Neural networks as data

Emilien Dupont

Jesus College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

September 2022

Acknowledgements

First, I would like to sincerely thank my supervisors Yee Whye and Arnaud for their support and guidance throughout my DPhil. I am extremely grateful for the research freedom they gave me and for always encouraging me to explore any topic and idea I was interested in. I also want to thank the many amazing people I was lucky to collaborate with during my DPhil and internships, including the COIN group at Oxford, Qi, Josh, Miguel, Alex and Aditya at Apple and Hyunjik, Danilo, Ali and Dan at DeepMind.

I am also very grateful for all the friends that were there with me throughout the DPhil. There have been so many amazing people that I cannot possible list everyone.

To everyone in the stats department and office 1.17 in particular. To Jef, Bobs, Jin, Kaspar, Robert, Faaiz, Edwin, Fran, Charline, Sheh, Hyunjik, Andrew, Eduard, Carlo, Desi, Tyler, Qinyi, Adam F, K and G, Emile and many more. Thank you for making the office such a nice place to go to every day and for all the trips to the Japanese van and Gail's.

I also want to thank the many sports groups that helped me relax and have fun during my DPhil. Thank you to the table tennis club for all the training sessions, matches and trips to Incha and Zhang Ji. To Eric, Kritica, Ali, Jef, Abe, Jinlin, Zhongyi, Seb, Virginia and many more. Thank you to the Wolfson tennis crew, I loved every session even when we were playing in zero degrees and snow. Thank you to the beach volleyball crew and the Sunday badminton group, for all the fun games and many dinners.

Thank you also to many other friends in Oxford and outside of Oxford. In particular, Abdul, Kun, Lisa, Michael and William - thank you for being there for me. To Grace and Thu, thank you for making my internships so much more fun.

A special thank you to Jinlin for her continuous love and support - thank you for all the good times and for making the tough times a lot easier. Oxford life was better with you there. Finally, thank you to my family - Mor, Papa, Marie and Jesús - for always being there for me. Merci og tak.

Abstract

In machine learning, neural networks are typically treated as models to be used for various tasks such as classification, regression or generative modeling. In this thesis, we take a different perspective and treat neural networks as *data* instead of *models*.

Indeed, in deep learning, data is often represented by arrays, such as a 2D grid of pixels for images. However, the underlying signal represented by these arrays is often continuous, such as the scene depicted in an image. A powerful continuous alternative to discrete arrays is then to represent such signals with an implicit neural representation (INR), a neural network trained to output the appropriate signal value for any input spatial location. An image for example, can be parameterized by a neural network mapping pixel locations to RGB values. In this thesis, we investigate the consequences and compelling properties of using such neural networks as data. We motivate and explore this approach through two main applications: compression, where we store data as neural networks, and generative modeling, where we train generative models directly on datasets of neural networks.

For data compression, we propose to store the quantized weights of an INR as a compressed code for a given signal (such as an image), instead of directly compressing the discretized signal. We show how this allows us to build a single neural codec that is seamlessly applicable to multiple data modalities, from images and audio to medical and climate data. For generative modeling, we propose to learn distributions of INRs, either implicitly by training on array data or explicitly by directly training models on datasets of INRs. We demonstrate how such an approach leads to compelling algorithms for a range of machine learning tasks, including 3D shape inference and generative modeling of neural radiance fields.

Contents

List of Figures

1 Introduction

1.1 Motivation

Consider capturing a scene such as the one shown in Figure 1.1 in an image. Even though the scene itself is spatially continuous, the corresponding visual signal is typically discretized into a set of pixels in order to store it on a computer. The image then forms an approximate representation of the true visual signal depicting the scene. To obtain a more accurate representation of this scene, we can increase the spatial resolution of the image as shown in Figure 1.1. However, increasing the resolution also increases the amount of memory required to store the signal. Indeed, the number of pixels grows quadratically with resolution.

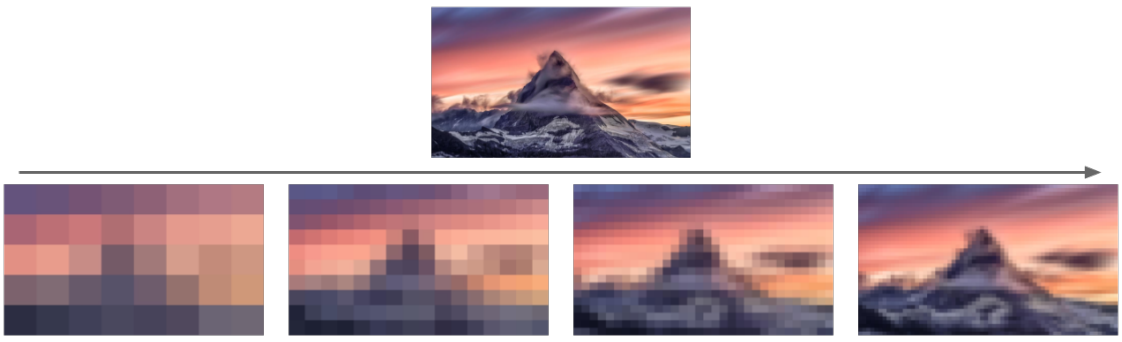

Figure 1.1: To obtain a more accurate representation of a scene (top) we can increase the spatial resolution of the image representing it (bottom) at the cost of increased memory.

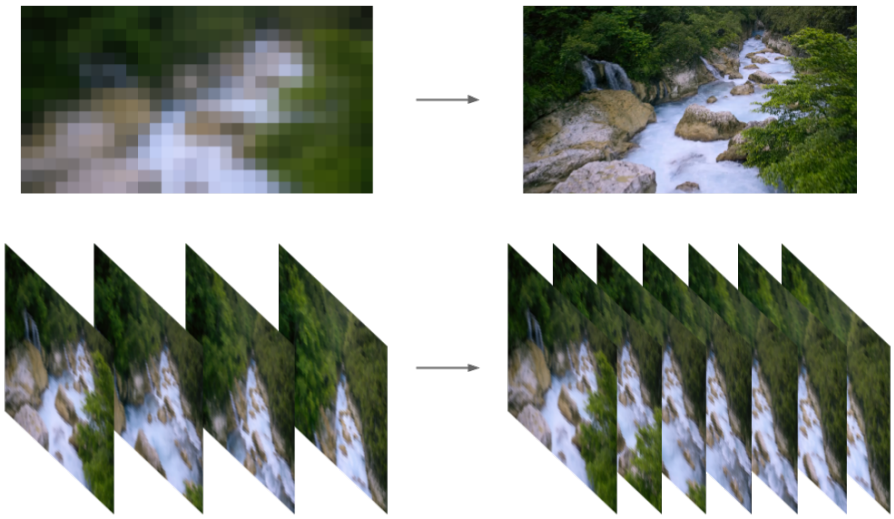

Figure 1.2: We can increase resolution both in space (top) and time (bottom) to obtain a more accurate representation of a dynamic scene.

Similarly, we can consider observing a dynamic scene which is continuous both in space and time. To store this signal on a computer, we discretize it in space (into pixels) and time (into frames) into a video (see Figure 1.2). Again, finer discretization will yield a more accurate representation at the cost of increasing memory. For videos, the memory required grows very quickly. For example, storing 1 second of raw HD video requires

$$\underbrace{30}_{\text{fps}} \times \underbrace{1920}_{\text{width}} \times \underbrace{1080}_{\text{height}} \times \underbrace{3}_{\text{RGB}} \times \underbrace{1}_{\text{1 byte/color}} = 186\text{MB}. \tag{1.1}$$

Storing 27 seconds of raw HD video then requires 5GB, which is roughly the same amount of memory as a typical 2 hour HD movie [Netflix, 2022]. This stark difference arises because videos are almost always compressed: there is a large amount of redundancy in video data (e.g. two consecutive frames are often nearly identical) and video codecs typically obtain large compression rates [Bross et al., 2021]. For this reason, videos are very rarely consumed in raw form. However, the majority of deep learning models do consume video as highly inefficient raw arrays (see Figure 1.3), and are therefore severely limited in the length and resolution of the videos on which classifiers or generative models can be trained [Karpathy et al., 2014, Wang et al., 2017, Zhou et al., 2020a, Skorokhodov et al., 2022, Ho et al., 2022].

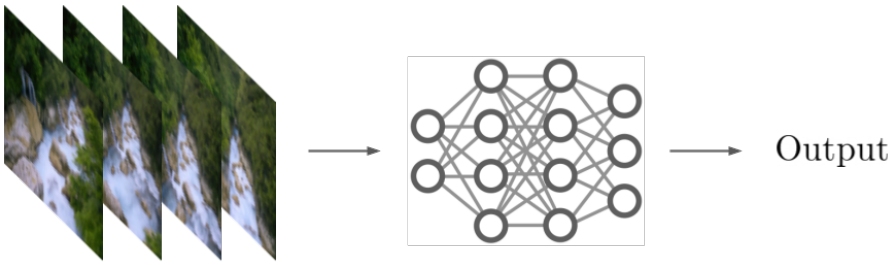

Figure 1.3: Typically, videos are fed to neural networks as uncompressed arrays of frames.

As videos (and images) are highly compressible, it would therefore seem reasonable to also train neural networks on compressed versions of these data modalities. For images, a series of works have considered training generative models on the compressed latent vector of autoencoders, with the VQ-VAE model family being the most prominent example of this [Van Den Oord et al., 2017, Razavi et al., 2019]. However, these still rely on training an autoencoder on the data in raw format (as well as having several other limitations discussed in Section 1.3). Other works have built classifiers and generative models directly on the coefficients obtained from JPEG compression [Ehrlich and Davis, 2019, Nash et al., 2021a]. However, these models are specific to a single codec and are not trivial to apply more generally.

Instead of training on compressed formats, it may therefore be interesting to consider training models on data that has not been discretized onto a grid and therefore is not directly tied to the quadratic or cubic memory scaling resulting from this. For example, in order to build generative models of large scale video, memory restrictions imply that we will likely need to eventually abandon discretized grids of pixels and frames.

The issue of memory is even more severe for 3D scenes where the scaling of discretized representations is often cubic, even for static scenes. As a result, neural networks trained on 3D scenes are typically restricted to low resolution, small scale signals (see [Park et al., 2019] for a discussion of this). This scaling issue may also be part of the reason why the majority of models in computer vision are trained on images even though images are a static two dimensional projection of

a dynamic three dimensional world. Indeed, learning, for example, distributions over dynamic 3D scenes is important for building models that reason or perform inference in the real world. However, using traditional discretized representations may be too restrictive to achieve this goal.

An interesting question is then whether we can represent data without explicitly discretizing it, in a way that is compatible with deep learning models? Inspired by recent success (particularly in the field of 3D computer vision) in representing signals with continuous neural networks [Park et al., 2019, Mescheder et al., 2019, Chen and Zhang, 2019, Sitzmann et al., 2019b, Mildenhall et al., 2020], we explore the idea of treating the continuous neural networks themselves as data, with the hope of eventually enabling deep learning on data modalities that are not currently tractable.

1.2 Background

In this section, we review the main ideas around representing data as neural networks as well as laying the foundations for the rest of the work presented in this thesis. Most of the ideas as they are presented in this section were initially introduced in [Mescheder et al., 2019, Park et al., 2019, Chen and Zhang, 2019, Sitzmann et al., 2020b].

1.2.1 Definition

In this thesis, we restrict ourselves to representing data that can be expressed in terms of sets of coordinates $\mathbf{x} \in \mathcal{X}$ and features $\mathbf{y} \in \mathcal{Y}$. An image for example, can be described by a set of pixel locations in $\mathbf{x} = (x, y)$ in $\mathbb{R}^2$ and their corresponding RGB values $\mathbf{y} = (r, g, b)$ in $\{0, 1, ..., 255\}^3$. A video can be described in the same way by adding a time index t to the coordinates, that is $\mathbf{x} = (x, y, t)$. Similarly, a 3D shape can be defined by sets of 3D coordinates $\mathbf{x} = (x, y, z)$ and their corresponding occupancy values $p \in \{0, 1\}$ indicating whether a point is inside or outside the 3D shape. This definition even encompasses data lying on manifolds: we can for example describe climate measurements on the Earth (as a sphere), by a set of

latitude and longitude coordinates $\mathbf{x} = (\lambda, \varphi)$ and the corresponding measurements $\mathbf{y} = (T, P)$, such as temperature T, pressure P and so on.

We then define a single data point $\mathbf{d}$ as a collection of coordinate and feature pairs $\mathbf{d} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (for example an image as a collection of n pixel locations and RGB values). As the previous examples highlight, this is a very general description and encompasses a large class of data modalities.

We are then interested in representing a data point $\mathbf{d}$ by a function $f : \mathcal{X} \to \mathcal{Y}$ mapping coordinates $\mathbf{x} \in \mathcal{X}$ to features $\mathbf{y} \in \mathcal{Y}$. For images for example, we have $f : \mathbb{R}^2 \to \{0, 1, ..., 255\}^3$ where $f(x, y) = (r, g, b)$ returns the RGB values at pixel location (x, y). In practice, we parameterize such a function f_θ by an MLP with weights θ, often referred to as an *implicit neural representation* [1] (INR). To fit an INR f_θ to a datapoint $\mathbf{d}$, we simply minimize some loss $\mathcal{L}(\theta, \mathbf{d})$, usually the mean squared error

$$\mathcal{L}(\theta, \mathbf{d}) = \sum_{i=1}^{n} \|f_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2 \tag{1.2}$$

over the parameters θ. The function f_θ (or equivalently the parameters θ) then form a representation of the data point $\mathbf{d}$.

We can note a few properties about this representation. For images for example, the sum in Equation 1.2 is over all pixel locations. While the optimization problem depends on the number of pixels n, the representation f_θ itself is, remarkably, *independent* of the number of pixels. The representation f_θ therefore, unlike most image representations, does not depend on the resolution of the image. Indeed, a core property of INRs is that they scale with signal *complexity* and not with signal resolution [Sitzmann et al., 2020b]. Indeed, the memory required to store data scales quadratically with resolution for images and cubically for voxel grids. In contrast, for functional representations, the memory requirements scale directly with signal complexity: to represent a more complex signal, we increase the capacity of the function f_θ, for example by increasing the number of layers of a neural network.

Figure 1.4: Modeling an image with a neural network with (right) and without (left) Fourier features.

1.2.2 Learning high frequency functions

While we have defined the loss function for fitting an INR f_θ in Equation 1.2, we have not yet chosen a specific parameterization for f_θ. The naive choice for this is a standard MLP with e.g. ReLU activations. However, it has recently been shown that learning function representations by minimizing equation (1.2) with a regular MLP is biased towards learning low frequency functions [Mildenhall et al., 2020, Sitzmann et al., 2020b, Tancik et al., 2020]. Indeed, minimizing Equation 1.2 and specifically learning the high frequency components of the function is prohibitively slow even with arbitrarily large MLPs [Ronen et al., 2019, Bietti and Mairal, 2019, Tancik et al., 2020].

Several approaches have been proposed to alleviate this problem, including the use of positional encodings [Mildenhall et al., 2020], random Fourier features [Tancik et al., 2020] and sinusoidal activation functions [Sitzmann et al., 2020b]. In this thesis, we use both random Fourier features and sinusoidal activations and describe these in detail below.

Random Fourier features

The use of random Fourier features was proposed by Tancik et al. [2020] to ease the learning of high frequency details for INRs. Specifically, given a coordinate

[1]There are several commonly used names for this concept, see Section 1.2.3 for a discussion.

$\mathbf{x} \in \mathbb{R}^d$, an encoding function $\gamma : \mathbb{R}^d \to \mathbb{R}^{2m}$ is defined as

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi B\mathbf{x}) \\ \sin(2\pi B\mathbf{x}) \end{pmatrix}, \tag{1.3}$$

where $B \in \mathbb{R}^{m \times d}$ is a (potentially learnable) random matrix whose entries are typically sampled from $\mathcal{N}(0, \sigma^2)$. The number of frequencies m and the variance σ^2 of the entries of B are hyperparameters. To learn high frequency functions, we simply encode $\mathbf{x}$ before passing it through the MLP, $f_\theta(\gamma(\mathbf{x}))$, and minimize equation (1.2). As can be seen in Figure 1.4, fitting an INR to an image with a ReLU MLP fails to capture high frequency detail whereas using random Fourier features followed by a ReLU MLP allows us to faithfully reproduce the image.

Sinusoidal activation functions

An alternative to random Fourier features called SIREN was proposed by Sitzmann et al. [2020b]. The SIREN approach consists in replacing the activation functions in the MLP with sine functions as well as using an appropriate initialization of the MLP weights, allowing the model to fit high frequency data such as natural images. More specifically, a SIREN layer is defined by an elementwise sin applied to a hidden feature vector $\mathbf{h} \in \mathbb{R}^k$ as

$$\text{SIREN}(\mathbf{h}) = \sin(\omega_0(W\mathbf{h} + \mathbf{b})) \tag{1.4}$$

where $W \in \mathbb{R}^{k \times k}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^k$ a bias vector and $\omega_0 \in \mathbb{R}^+$ a positive scaling factor. While we found that SIRENs were usually less stable to train than MLPs with random Fourier features, we also found that they were generally more compact (obtaining lower losses with fewer parameters) and as such are useful for compression.

1.2.3 Naming conventions

As neural networks mapping coordinates to features have been referred to by several names in the literature, and there is still no single well established naming

convention, we briefly discuss the various names here. The three[2] most common names are *implicit neural representations*, *neural fields* and *coordinate-based MLPs*. Implicit neural representations (INR) initially gained their name as they were used to represent 3D objects implicitly [Mescheder et al., 2019, Park et al., 2019, Chen and Zhang, 2019]. For example, a 3D surface can be implicitly represented by the 0-level set of a signed distance function. However, INRs are often used to represent data *explicitly*, e.g. the mapping from pixel locations to RGB values. The term neural fields has been proposed as an alternative [Xie et al., 2022], however the word field itself has several definitions within mathematics, physics and computer science[3]. Arguably, the most common definition of field is a map from a manifold to a vector space. According to this definition, many neural networks that would not be considered neural fields are then also fields, e.g. neural networks used for regression problems. Finally, coordinate-based MLP has been frequently used [Tancik et al., 2020]. However, the neural networks or models used to represent these are not always standard MLPs [Liang et al., 2021a, Fridovich-Keil et al., 2022, Chen et al., 2022].

While it is difficult to find a single term to accurately define these neural networks, we believe the key property is that they *represent* data as a *function* (or a field). Indeed, most neural networks are used as a *model* to solve some task, such as classification or generative modeling. In contrast, INRs/neural fields/coordinate-based MLPs are functional *representations* of data. An appropriate name could then be *neural functional representations* or *neural field representations* for example. However, to be consistent with the dominant term in the literature, we use mostly implicit neural representations (INRs) throughout this thesis. As naming conventions might change, we have deliberately left the title more ambiguous, using simply "neural networks".

[2]In very early work, INRs were referred to as Compositional Pattern Producing Networks [Stanley, 2007], but this is rarely used now.

[3]For example, the field of real numbers, vector fields, fields of sets and so on.

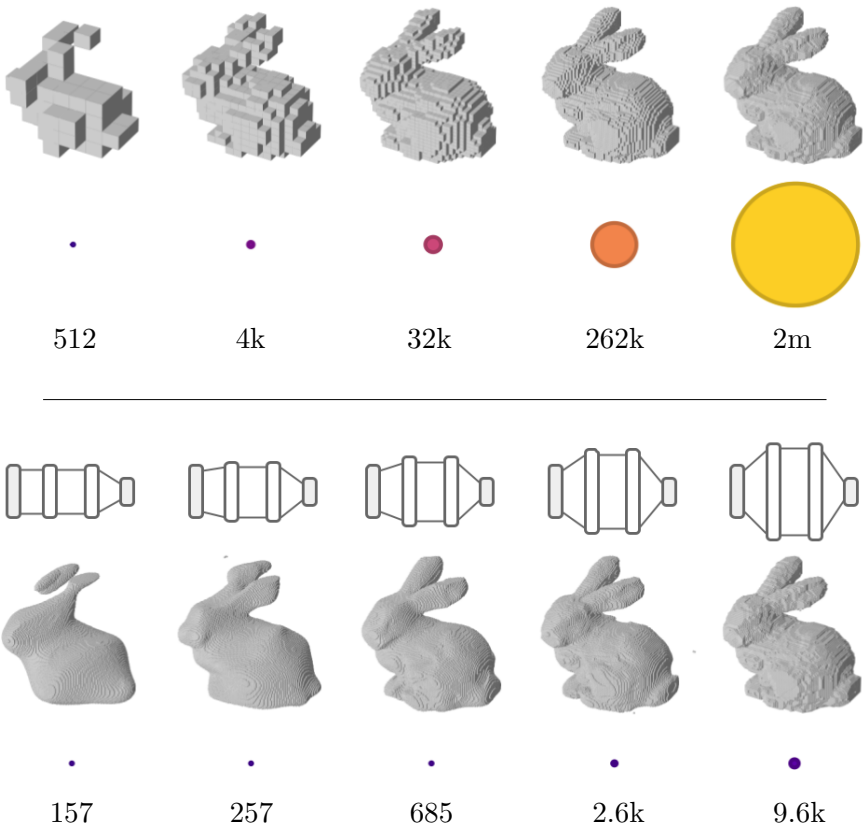

Figure 1.5: Top row: voxelized Stanford Bunny at various resolutions as well as the number of voxels required to represent each of them. Bottom row: INRs fit to each voxel grid, their corresponding rendering and number of parameters.

1.3 Properties of implicit neural representations

In this section we discuss various compelling properties of INRs and motivate their use as data points.

1.3.1 Scaling

One fundamentally desirable property of INRs is that they are adaptive. Indeed, their size scales with signal *complexity* and not signal *resolution*. In contrast, standard array based representations typically scale poorly with resolution[4] (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, particularly in 3D [Park et al., 2019, Mescheder et al., 2019].

[4]This phenomenon is sometimes referred to as the *curse of discretization* [Mescheder, 2020].

We can illustrate this point with a simple experiment. We first discretize the Stanford Bunny (which we consider a spatially continuous 3D object, in a similar way that the scene represented in an image is spatially continuous), to various voxel grid resolutions (from 8^3 to 128^3). Given these voxel grids we then fit the smallest possible INR (in terms of number of parameters) to each resolution and measure how the number of voxels grows compared to the number of parameters for an INR. Results of this experiment are shown in Figure 1.5. As can be seen, INRs scale much more gracefully. For example, for the largest grid, we require $128^3 \approx 2$million voxels, while we require less than 10k parameters for the INR. Treating INRs as data points can then potentially lead to more efficient training of neural networks for downstream tasks. Indeed, high dimensional but low entropy data are particularly amenable to being represented with INRs, precisely because they are very compressible.

We note that we used voxel grids which are famously memory inefficient for this experiment. There are a plethora of more efficient methods to represent 3D geometry, including point clouds and meshes. However, similar experiments can be carried out with these and the same results hold both for meshes [Davies et al., 2020] and point clouds [Isik et al., 2021].

1.3.2 Optimization and inverse problems

While there are several standard methods for representing signals continuously (using e.g. a Fourier or wavelet basis), one of the main advantages of using INRs over these is the ease with which they can be optimized, particularly for certain classes of inverse problems. INRs have for example been applied to inverse problems in medical imaging [Vasconcelos et al., 2022] and famously for inverse rendering with NeRF [Mildenhall et al., 2020]. Indeed, NeRF parameterizes 3D scenes with INRs and infers the 3D scene structure from a set of posed 2D images by solving an inverse rendering problem. This has led to a flurry of work [Dellaert and Yen-Chen, 2020] and highlights the potential of using INRs to solve highly complex inverse problems. As a recent review paper on INRs succinctly put it [Xie et al., 2022]: "By their analytic differentiability, gradient descent, and over-parameterization,

neural fields are effective at regressing complex signals through optimizations for problems that are otherwise ill-posed."

1.3.3 Easing downstream tasks

Treating INRs as data may also ease certain downstream tasks. For example, we can consider the task of generative modeling of 3D scenes (an important problem in computer vision) by learning distributions of NeRF scenes. Using a standard approach, learning generative models of NeRF scenes would require simultaneously learning a distribution over 3D scenes while inferring 3D structure from 2D images (i.e. simultaneously solving an inverse problem and modeling the distribution over the results of the inverse problem). In contrast, using the framework we propose in this thesis, we first infer the 3D scene from 2D images by minimizing reconstruction error on views - a relatively easier tasks - and only after we obtain a dataset of 3D scenes as INRs, we model their distribution. Another advantage of our approach is that the generative model does not have to backpropagate through the expensive volume rendering operation in NeRF. We therefore hope this perspective could be useful for scaling generative models to larger 3D scenes, as it is not memory limited by the rendering operation.

1.3.4 Moving away from fixed resolutions

Datasets of array representations (such as images) are typically stored at a fixed resolution and deep learning models often can only ingest data at a single 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 and so on. On the other hand, we can easily convert data at a variety of resolutions to INRs simply via optimization. As the INRs themselves do not directly depend on resolution, a downstream model trained on datasets of INRs would therefore also be able to handle data at a range of resolutions.

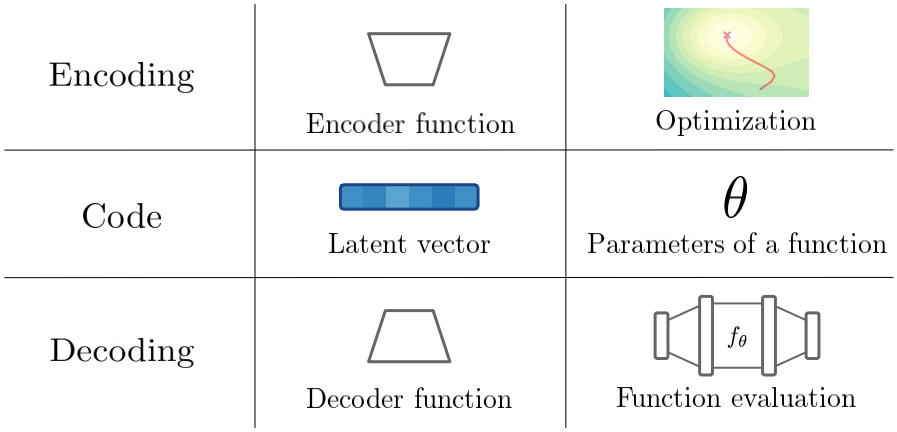

Figure 1.6: Comparison of autoencoders (middle) and functional representations (right).

1.3.5 Signals that are 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 can be more difficult to discretize. For example, NeRF uses volumetric rendering which requires the neural radiance field to be evaluated at arbitrary spatial locations [Mildenhall et al., 2020]. In addition, in many scientific disciplines we are interested in representing fields or densities which may not be trivial to discretize. Further, physical fields often require continuous derivatives to solve differential equations of motion. INRs then provide a natural way to express data continuously and as such bypass the need for discretization when training models on downstream tasks.

1.3.6 Multimodality and INRs vs autoencoders

It is standard practice in deep learning to use specialized encoder and decoder architectures for different data modalities. However, designing encoders and decoders is typically difficult. While excellent encoder and decoder architectures exist for images (such as ResNets [He et al., 2016] or Vision Transformers [Dosovitskiy et al., 2020]), this is not necessarily the case for other modalities. For example, designing encoders for NeRF scenes is challenging and requires aggregating a collection of images and poses into a radiance field [Kosiorek et al., 2021]. Building encoders for 3D meshes is also difficult, although recent works have had some

success tackling this [Zhou et al., 2020b]. Similarly, data lying on manifolds require highly specialized architectures [Cohen et al., 2018]. In contrast, INRs can be used to encode and decode a wide variety of data modalities through a generic optimization procedure. In addition, partial observations are easily handled by INRs (we simply fit the signal to the partial observations only), which is typically not the case for autoencoders. For example, it is not trivial to obtain a meaningful latent representation from a partially occluded image.

We also note that obtaining a compact latent representation through an autoencoder requires training on a distribution of data to learn an encoder. With INRs, we can in principle use as little as a single data point to obtain a representation. This property has been used for video compression for example [Chen et al., 2021]. The multimodal property of INRs has also been explored in several recent works (including the ones presented in this thesis). A notable example is GEM [Du et al., 2021], which learns a common latent space for multiple modalities using INRs, allowing the model to jointly learn a latent space for video and audio for example.

Finally, it is interesting to directly compare each component of encoder/decoders and INRs. This is particularly instructive for understanding where differences in performance and expressiveness come from, particularly in the context of neural compression where autoencoders are ubiquitous. A summary of these comparisons can be found in Figure 1.6 and can be split into 3 parts: encoding, the code and decoding.

Encoding. For autoencoders, the encoder is a neural network mapping the input data to a latent vector. This encoder can in principle be made arbitrarily expressive by increasing the size of the neural network. For INRs, encoding is optimization. Given a powerful enough decoder, the optimization procedure can in principle also be made arbitrarily "expressive" (i.e. we assume that we have an optimization procedure that can find the best possible encoding given the decoder).

Code. For autoencoders, the code is a latent vector output by the encoder and fed to the decoder. The size of this code can be changed irrespective of the size of both the encoder and decoder. For INRs, the code is a set of parameters specifying

the functional representation (for example the weights of an MLP). For naive parameterizations, the size of this code is *not* independent of the decoder, since the decoder is the INR itself. Increasing the capacity of the decoder then also increases the size of the code which is not desirable. However, INR parameterizations exist where the size of the decoder and the code become independent. We introduce one such parameterization in Chapter 4.

Decoding. For autoencoders, the decoder is a neural network mapping a latent vector to a reconstruction of the data. This decoder can also be made arbitrarily expressive by increasing the capacity of the neural network. For INRs, the decoder is the INR itself and decoding simply corresponds to evaluating the function. The decoder can also be made arbitrarily expressive by increasing the size of the INR (which can be done independently of code size as mentioned in the code section).

For both encoding and decoding, there is therefore in principle no reason why INRs could not match the performance of autoencoders. However, in practice, particularly for images, autoencoders typically obtain better reconstructions using smaller latent vectors (see Chapters 3 and 4). This suggests that the way INRs are currently parameterized and optimized is suboptimal. An open question is then how to parameterize and design optimization algorithms for INRs such that they can match the compression performance of autoencoders. If this were possible, we would, thanks to the multimodal property of INRs, potentially be able to build compression algorithms that are generally applicable to a wide variety of modalities. Building such a "universal codec" is one of our main motivations for exploring compression with INRs.

1.4 Thesis outline

While INRs have various compelling properties, they are still predominantly viewed as *models*. In this thesis, we take a different perspective and treat INRs as *data*. We then view INRs as first class citizens, to be used as any other data structure or representation in deep learning.

We explore the feasibility and properties of this perspective mainly through two applications: compression (where we store data as neural networks) and generative modeling (where we learn distributions over neural networks).

In Chapter 2, we introduce GASP, a generative model of functions which, in addition to being agnostic to discretization, can be seamlessly applied to a range of different data modalities from images and 3D shapes to climate data. By acting only on functions and their input and output pairs, the proposed model can be trained end-to-end without the need for grids or arrays.

In Chapter 3, we introduce COIN, an alternative to traditional neural compression methods. Instead of directly compressing the RGB values for each pixel of an image, we compress the weights of a neural network overfitted to the image. We show that such an approach has various interesting properties which could make it a compelling alternative to autoencoder based neural compression.

In Chapter 4, we introduce COIN++, which improves upon COIN in several ways. More specifically, COIN++ uses meta-learning to reduce compression time by several orders of magnitude, a base network with modulations to encode shared features across datasets, as well as an improved quantization and entropy coding scheme. In addition, we demonstrate that COIN++ can compress a wide range of data modalities, including images, audio, medical and climate data.

In Chapter 5, we introduce the functa approach, which consists in training models on datasets of INRs. We use a parameterization similar to COIN++ to build datasets of INRs, which are then used as training data for various machine learning tasks, including generative modeling, classification and data imputation. We show that it is feasible to train models on datasets of INRs, while also demonstrating why this might be a useful approach, for example for generative modeling of 3D NeRF scenes or 3D shape inference.

In Chapter 6, we discuss the potential and limitations of our approach, as well as suggesting directions for future work. In addition, we include various preliminary experimental results highlighting interesting research directions for improving both the learning of datasets of INRs as well as the training of models on datasets of INRs.

1.5 Related concepts

In this section, we briefly discuss some concepts that are tangentially related, but not directly relevant, to this thesis. We do not intend to provide an in depth review of each of these topics, as several excellent reviews already exist which we refer to in the relevant sections. For ideas directly related to this thesis, we provide in-depth surveys and literature reviews within each chapter, covering the material that is relevant to the topics explored in each specific part.

Functional data analysis. Functional data analysis (FDA) is a branch of statistics concerned with analysing, modeling and building theory for functional data (i.e. data where each sample is a function) [Wang et al., 2016]. The two most prominent tools of FDA are functional principal component analysis [Jolliffe, 2002] and functional linear regression [Ramsay and Dalzell, 1991] although nonlinear methods also exist [Wang et al., 2016]. As FDA is generally concerned with functional data, our work can be seen as a special case of this field. In practice, however, the tools we use (implicit neural representations) and our goals (performing deep learning on functions parameterized by INRs) are quite different. For a recent review of the field of functional data analysis we refer the reader to Wang et al. [2016].

Neural processes. Neural processes [Garnelo et al., 2018a,b] are a family of models that learn (typically conditional) distributions of functions, usually in the context of uncertainty quantification and meta-learning. While neural processes have been successful in modeling 1D functions as well as small scale images in a continuous manner [Garnelo et al., 2018b, Kim et al., 2019, Gordon et al., 2019], they currently do not scale to data modalities handled by INRs, including large scale 3D geometry and NeRF scenes (although adding attention [Kim et al., 2019] and translation equivariance [Gordon et al., 2019] help alleviate some of the scaling issues). Further, neural processes are used to learn conditional distributions of functions and it is not immediately clear how to apply them more generally to the tasks we are concerned with here, such as classification and compression. For a review of neural processes, we refer the reader to Jha et al. [2022].

Neural differential equations and operators. Neural differential equations [Chen et al., 2018] are a class of models that operate in continuous time (or space), typically by parameterizing the vector field of a differential equation by a neural network. Similarly to our work, variants of these models can act on functional data, such as continuous time series [Rubanova et al., 2019, Kidger et al., 2020]. Neural operators [Li et al., 2020b,a, Lu et al., 2021a, Kovachki et al., 2021] are a related class of models that learn mappings between function spaces (as opposed to the typical finite dimensional spaces on which neural networks act) and as a result are invariant to the discretization of the underlying space. However, these models are typically applied to modeling dynamics (such as ordinary or partial differential equations) in a resolution invariant manner and not with solving general deep learning tasks. Further, it is not clear how to use these frameworks on modalities like NeRF scenes for which INRs perform well. Nonetheless, we believe there may be interesting ways to combine INRs and neural differential equations and operators, particularly for modeling dynamic scenes [Pumarola et al., 2021]. We refer the reader to Kidger [2022] for a survey of neural differential equations and Kovachki et al. [2021] for a review of neural operators.

1.6 Papers

1.6.1 Papers included in this thesis

Each chapter of this thesis is based on a paper. The list of these is included in chronological order here for completeness.

1. Generative Models as Distributions of Functions, ***E. Dupont***, *Y. W. Teh, A. Doucet*, **AISTATS 2022**, [Dupont et al., 2022c]

2. COIN: COmpression with Implicit Neural representations, ***E. Dupont****, *A. Goliński*, M. Alizadeh, Y. W. Teh, A. Doucet*, **ICLR 2021 Neural Compression Workshop**, [Dupont et al., 2021]

3. COIN++: Neural Compression Across Modalities, ***E. Dupont****, *H. Loya**, *M. Alizadeh, A. Goliński, Y. W. Teh, A. Doucet*, **TMLR 2022**, [Dupont et al., 2022b]

4. From data to functa: Your data point is a function and you can treat it like one, ***E. Dupont****, *H. Kim**, *A. Eslami, D. Rezende, D. Rosenbaum*, **ICML 2022**, [Dupont et al., 2022a]

1.6.2 Papers omitted from this thesis

To keep the thesis unified and concise, I have omitted several papers that I published during my DPhil. A list of these is included in chronological order here for completeness.

1. Learning Disentangled Joint Continuous and Discrete Representations, ***E. Dupont***, **NeurIPS 2018**, [Dupont, 2018]

2. Probabilistic Semantic Inpainting with Pixel Constrained CNNs, ***E. Dupont***, *S. Suresha*, **AISTATS 2019**, [Dupont and Suresha, 2019]

3. Augmented Neural ODEs, ***E. Dupont***, *A. Doucet, Y. W. Teh*, **NeurIPS 2019**, [Dupont et al., 2019]

4. Equivariant Neural Rendering, ***E. Dupont***, *M. Bautista, A. Colburn, A. Sankar, J. Susskind, Q. Shan*, **ICML 2020**, [Dupont et al., 2020]

5. LieTransformer: Equivariant self-attention for Lie groups, *M. Hutchinson**, *C. Le Lan**, *S. Zaidi**, ***E. Dupont***, *Y. W. Teh, H. Kim* **ICML 2021**, [Hutchinson et al., 2021]

2

Generative models as distributions of functions

Abstract

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

2.1 Introduction

In generative modeling, data is often represented by discrete arrays. Images are represented by two dimensional grids of RGB values, 3D scenes are represented by three dimensional voxel grids and audio as vectors of discretely sampled waveforms. However, the true underlying signal is often continuous. We can therefore also consider representing such signals by continuous functions taking as input grid coordinates and returning features. In the case of images for example, we can define a function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ mapping pixel locations to RGB values using a neural network. Such representations, typically referred to as implicit neural representations, coordinate-based neural representations or neural function representations, have the remarkable property that they are independent of signal resolution [Park et al., 2019, Mescheder et al., 2018, Chen and Zhang, 2019, Sitzmann et al., 2020b].

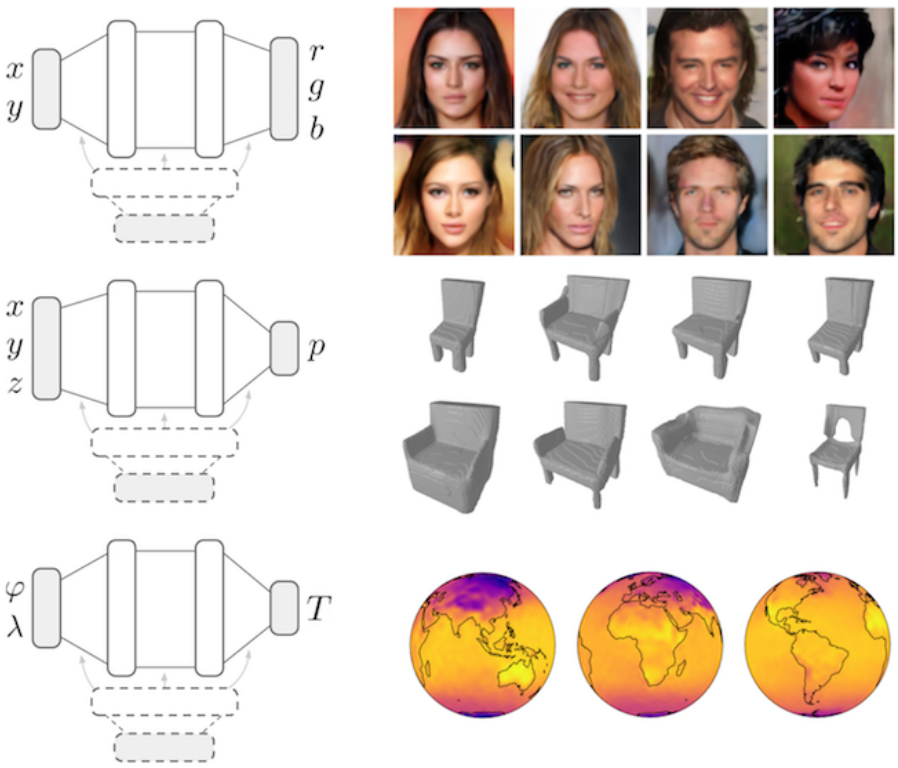

Figure 2.1: By representing data as continuous functions, we can use the same model to learn distributions of images, 3D shapes and climate data, irrespective of any underlying grid or discretization.

In this paper, we build generative models that inherit the attractive properties of implicit representations. By framing generative modeling as learning distributions of functions, we are able to build models that act entirely on continuous spaces, independently of resolution. We achieve this by parameterizing a distribution over neural networks with a hypernetwork [Ha et al., 2017] and training this distribution with an adversarial approach [Goodfellow et al., 2014], using a discriminator that acts directly on sets of coordinates (e.g. pixel locations) and features (e.g. RGB values). Crucially, this allows us to train the model irrespective of any underlying discretization or grid and avoid the *curse of discretization* [Mescheder, 2020].

Indeed, standard convolutional generative models act on discretized grids, such as images or voxels, and as a result scale quadratically or cubically with resolution, which quickly becomes intractable at high resolutions, particularly in 3D [Park et al., 2019]. In contrast, our model learns distributions on continuous spaces and is agnostic to discretization. This allows us to not only build models that act independently of resolution, but also to learn distributions of functions on manifolds where discretization can be difficult.

To validate our approach, we train generative models on various image, 3D shape and climate datasets. Remarkably, we show that, using our framework, we can learn rich function distributions on these varied datasets using the *same model*. Further, by taking advantage of recent advances in representing high frequency functions with neural networks [Mildenhall et al., 2020, Tancik et al., 2020, Sitzmann et al., 2020b], we also show that, unlike current approaches for generative modeling on continuous spaces [Garnelo et al., 2018a, Mescheder et al., 2019, Kleineberg et al., 2020], we are able to generate sharp and realistic samples.

2.2 Representing data as functions

In this section we review implicit neural representations, using images as a guiding example for clarity.

Representing a single image with a function. Let I be an image such that $I[x, y]$ corresponds to the RGB value at pixel location (x, y). We are interested in

representing this image by a function $f : \mathbb{R}^2 \to \mathbb{R}^3$ where $f(x, y) = (r, g, b)$ returns the RGB values at pixel location (x, y). To achieve this, we parameterize a function f_θ by an MLP with weights θ, often referred to as an *implicit neural representation*. We can then learn this representation by minimizing

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

where the sum is over all pixel locations. Remarkably, the representation f_θ is *independent* of the number of pixels. The representation f_θ therefore, unlike most image representations, does not depend on the resolution of the image [Mescheder et al., 2019, Park et al., 2019, Sitzmann et al., 2020b].

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

$$\min_\theta \sum_{i=1}^n \| f_\theta(\mathbf{x}_i) - \mathbf{y}_i \|_2^2. \tag{2.1}$$

A core property of these representations is that they scale with signal *complexity* and not with signal resolution [Sitzmann et al., 2020b]. Indeed, the memory required to store data scales quadratically with resolution for images and cubically for voxel grids. In contrast, for function representations, the memory requirements scale directly with signal complexity: to represent a more complex signal, we would need to increase the capacity of the function f_θ, for example by increasing the number of layers of a neural network.

Representing high frequency functions. Recently, it has been shown that learning function representations by minimizing equation (2.1) is biased towards learning low frequency functions [Mildenhall et al., 2020, Sitzmann et al., 2020b,

Figure 2.2: Modeling an image with a function with (right) and without (left) Fourier features.

Tancik et al., 2020]. While several approaches have been proposed to alleviate this problem, we use the random Fourier feature (RFF) encoding proposed by Tancik et al. [2020] as it is not biased towards on axis variation (unlike Mildenhall et al. [2020]) and does not require specialized initialization (unlike Sitzmann et al. [2020b]). Specifically, given a coordinate $\mathbf{x} \in \mathbb{R}^d$, the encoding function $\gamma : \mathbb{R}^d \to \mathbb{R}^{2m}$ is defined as

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

where $B \in \mathbb{R}^{m \times d}$ is a (potentially learnable) random matrix whose entries are typically sampled from $\mathcal{N}(0, \sigma^2)$. The number of frequencies m and the variance σ^2 of the entries of B are hyperparameters. To learn high frequency functions, we simply encode $\mathbf{x}$ before passing it through the MLP, $f_\theta(\gamma(\mathbf{x}))$, and minimize equation (2.1). As can be seen in Figure 2.2, learning a function representation of an image with a ReLU MLP fails to capture high frequency detail whereas using an RFF encoding followed by a ReLU MLP allows us to faithfully reproduce the image.

2.3 Learning distributions of functions

In generative modeling, we are typically given a set of data, such as images, and are interested in approximating the distribution of this data. As we represent data points by functions, we would therefore like to learn a distribution over functions. In the case of images, standard generative models typically sample some noise and

feed it through a neural network to output n pixels [Goodfellow et al., 2014, Kingma and Welling, 2014, Rezende et al., 2014]. In contrast, we sample the weights of a neural network to obtain a function which we can probe at arbitrary coordinates. Such a representation allows us to operate entirely on coordinates and features irrespective of any underlying grid representation that may be available. To train the function distribution we use an adversarial approach and refer to our model as a *Generative Adversarial Stochastic Process* (GASP).

2.3.1 Data representation

While our goal is to learn a distribution over functions, we typically do not have access to the ground truth functions representing the data. Instead, each data point is typically given by some *set* of coordinates and features $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$. For an image for example, we do not have access to a function mapping pixel locations to RGB values but to a collection of n pixels. Such a set of coordinates and features corresponds to input/output pairs of a function, allowing us to learn function distributions without operating directly on the functions. A *single* data point then corresponds to a *set* of coordinates and features (e.g. an image is a set of n pixels). We then assume a dataset is given as samples $\mathbf{s} \sim p_{\text{data}}(\mathbf{s})$ from a distribution over sets of coordinate and feature pairs. Working with sets of coordinates and features is very flexible - such a representation is agnostic to whether the data originated from a grid and at which resolution it was sampled.

Crucially, formulating our problem entirely on sets also lets us split individual data points into subsets and train on those. Specifically, given a single data point $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, such as a collection of n pixels, we can randomly subsample K elements, e.g. we can select K pixels among the n pixels in the entire image. Training on such subsets then removes any direct dependence on the resolution of the data. For example, when training on 3D shapes, instead of passing an entire voxel grid to the model, we can train on subsets of the voxel grid, leading to large memory savings (see Section 2.5.2). This is not possible with standard convolutional models which are directly tied to the resolution of the grid. Further, training on sets of coordinates

and features allows us to model more exotic data, such as distributions of functions on manifolds (see Section 2.5.3). Indeed, as long as we can define a coordinate system on the manifold (such as polar coordinates on a sphere), our method applies.

2.3.2 Function generator

Learning distributions of functions with an adversarial approach requires us to define a generator that generates fake functions and a discriminator that distinguishes between real and fake functions. We define the function generator using the commonly applied hypernetwork approach [Ha et al., 2017, Sitzmann et al., 2019b, 2020b, Anokhin et al., 2021, Skorokhodov et al., 2021]. More specifically, we assume the structure (e.g. the number and width of layers) of the MLP f_θ representing a single data point is fixed. Learning a distribution over functions f_θ is then equivalent to learning a distribution over weights $p(\theta)$. The distribution $p(\theta)$ is defined by a latent distribution $p(\mathbf{z})$ and a second function $g_\phi : \mathcal{Z} \to \Theta$, itself with parameters ϕ, mapping latent variables to the weights θ of f_θ (see Figure 2.3). We can then sample from $p(\theta)$ by sampling $\mathbf{z} \sim p(\mathbf{z})$ and mapping $\mathbf{z}$ through g_ϕ to obtain a set of weights $\theta = g_\phi(\mathbf{z})$. After sampling a function f_θ, we then evaluate it at a set of coordinates $\{\mathbf{x}_i\}$ to obtain a set of generated features $\{\mathbf{y}_i\}$ which can be used to train the model. Specifically, given a latent vector $\mathbf{z}$ and a coordinate $\mathbf{x}_i$, we compute a generated feature as $\mathbf{y}_i = f_{g_\phi(\mathbf{z})}(\gamma(\mathbf{x}_i))$ where γ is an RFF encoding allowing us to learn high frequency functions.

2.3.3 Point cloud discriminator

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

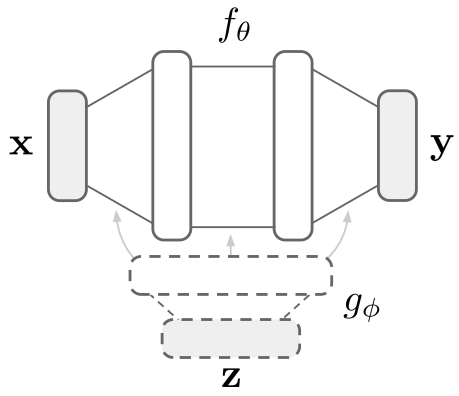

Figure 2.3: Diagram of a neural function distribution architecture. A latent vector $\mathbf{z}$ is mapped through a hypernetwork g_ϕ (in dashed lines) to obtain the weights of a function f_θ (in solid lines) mapping coordinates $\mathbf{x}$ to features $\mathbf{y}$.

As the core idea of our paper is to build generative models that are independent of discretization, we therefore cannot follow the naive approach of using convolutional discriminators. Instead, our discriminator should be able to distinguish between real and fake sets of coordinate and feature pairs. Specifically, we need to define a function D which takes in an *unordered set* $\mathbf{s}$ and returns the probability that this set represents input/output pairs of a real function. We therefore need D to be permutation invariant with respect to the elements of the set $\mathbf{s}$. The canonical choice for set functions is the PointNet [Qi et al., 2017] or DeepSets [Zaheer et al., 2017] model family. However, we experimented extensively with such functions and found that they were not adequate for learning complex function distributions (see Section 2.3.5). Indeed, while the input to the discriminator is an unordered set $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, there is an underlying notion of distance between points $\mathbf{x}_i$ in the coordinate space. We found that it is crucial to take this into account when training models on complex datasets. Indeed, we should not consider the coordinate and feature pairs as sets but rather as *point clouds* (i.e. sets with an underlying notion of distance).

While several works have tackled the problem of point cloud classification [Qi et al., 2017, Li et al., 2018, Thomas et al., 2019], we leverage the PointConv framework introduced by Wu et al. [2019] for several reasons. Firstly, PointConv layers are translation equivariant (like regular convolutions) and permutation invariant by construction. Secondly, when sampled on a regular grid, PointConv

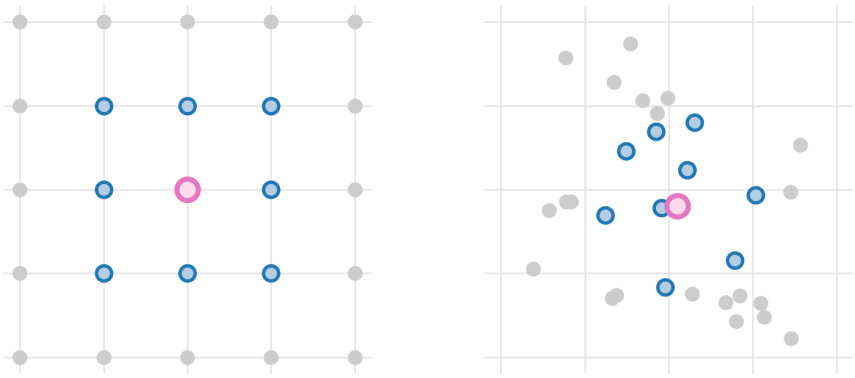

Figure 2.4: Convolution neighborhood for regular convolutions (left) and PointConv (right).

networks closely match the performance of regular CNNs. Indeed, we can loosely think of PointConv as a continuous equivalent of CNNs and, as such, we can build PointConv architectures that are analogous to typical discriminator architectures.

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

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

where $N_{\mathbf{x}}$ is a set of neighbors of $\mathbf{x}$ over which to perform the convolution (see Figure 2.4). Interestingly, this neighborhood is found by a nearest neighbor search with respect to some metric on the coordinate space. We therefore have more flexibility in defining the convolution operation as we can choose the most appropriate notion of distance for the space we want to model (our implementation supports fast computation on the GPU for any ℓ_p norm).

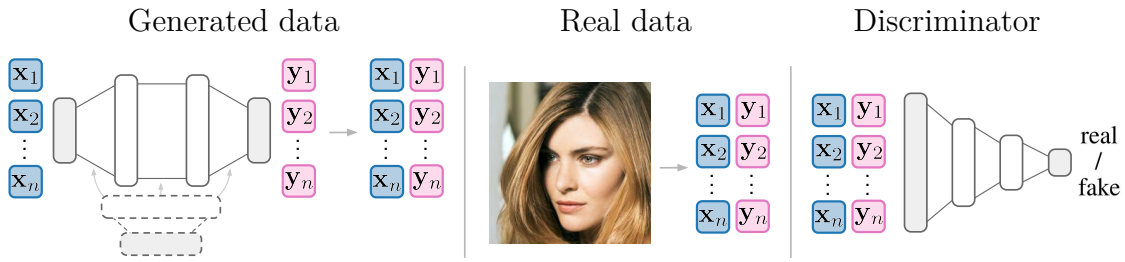

Figure 2.5: Training procedure for GASP: 1. Sample a function and evaluate it at a set of coordinate locations to generate fake point cloud. 2. Convert real data sample to point cloud. 3. Discriminate between real and fake point clouds.

2.3.4 Training

We use the traditional (non saturating) GAN loss [Goodfellow et al., 2014] for training and illustrate the entire procedure for a single training step in Figure 2.5. To stabilize training, we define an equivalent of the R_1 penalty from Mescheder et al. [2018] for point clouds. For images, R_1 regularization corresponds to penalizing the gradient norm of the discriminator with respect to the input image. For a set $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, we define the penalty as

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

that is we penalize the gradient norm of the discriminator with respect to the features. Crucially, our entire modeling procedure is then independent of discretization. Indeed, the generator, discriminator and loss all act directly on continuous point clouds.

2.3.5 How not to learn distributions of functions

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

Set discriminators. As described in Section 2.3.3, the canonical choice for set functions is the PointNet/DeepSet model family [Qi et al., 2017, Zaheer et al., 2017]. Indeed, Kleineberg et al. [2020] use a similar approach to ours to learn signed distance functions for 3D shapes using such a set discriminator. However, we

found both theoretically and experimentally that PointNet/DeepSet functions were not suitable as discriminators for complex function distributions (such as natural images). Indeed, these models do not directly take advantage of the metric on the space of coordinates, which we conjecture is crucial for learning rich function distributions. In addition, we show in the appendix that the Lipschitz constant of set functions can be very large, leading to unstable GAN training [Arjovsky et al., 2017, Roth et al., 2017, Mescheder et al., 2018]. We provide further theoretical and experimental insights on set discriminators in the appendix.

Auto-decoders. A common method for embedding functions into a latent space is the auto-decoder framework used in DeepSDF [Park et al., 2019]. This framework and variants of it have been extensively used in 3D computer vision [Park et al., 2019, Sitzmann et al., 2019b]. While auto-decoders excel at a variety of tasks, we show in the appendix that the objective used to train these models is not appropriate for generative modeling. We provide further analysis and experimental results on auto-decoders in the appendix.

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

2.4 Related work

Implicit representations. Implicit representations were initially introduced in the context of evolutionary algorithms as compositional pattern producing networks [Stanley, 2007]. In pioneering work, Ha [2016] built generative models of such networks for MNIST. Implicit representations for 3D geometry were initially (and concurrently) proposed by [Park et al., 2019, Mescheder et al., 2019, Chen and Zhang, 2019]. A large body of work has since taken advantage of these representations for inverse rendering [Sitzmann et al., 2019b, Mildenhall et al., 2020, Niemeyer et al., 2020, Yu et al., 2020], modeling dynamic scenes [Niemeyer et al., 2019,

Pumarola et al., 2021], modeling 3D scenes [Atzmon and Lipman, 2020, Jiang et al., 2020, Gropp et al., 2020], superresolution [Chen et al., 2020a] and learning convolutional kernels [Romero et al., 2021b,a].

Continuous models of image distributions. In addition to the work of Ha [2016], neural processes [Garnelo et al., 2018a,b] are another family of models that can learn (conditional) distributions of images as functions. However, the focus of these is on uncertainty quantification and meta-learning rather than generative modeling. Further, these models do not scale to large datasets, although adding attention [Kim et al., 2019] and translation equivariance [Gordon et al., 2019] helps alleviate this. Gradient Origin Networks [Bond-Taylor and Willcocks, 2021] model distributions of implicit representations using an encoder free model, instead using gradients of the latents as an encoder. In concurrent work, Skorokhodov et al. [2021], Anokhin et al. [2021] use an adversarial approach to learn distributions of high frequency implicit representations for images. Crucially, *these both use standard image convolutional discriminators* and as such do not inherit several advantages of implicit representations: they are restricted to data lying on a grid and suffer from the curse of discretization. In contrast, GASP is entirely continuous and independent of resolution and, as a result, we are able to train on a variety of data modalities.

Continuous models of 3D shape distributions. Mescheder et al. [2019] use a VAE to learn distributions of occupancy networks for 3D shapes, while Chen and Zhang [2019] train a GAN on embeddings of a CNN autoencoder with an implicit function decoder. Park et al. [2019], Atzmon and Lipman [2021] parameterize families of 3D shapes using the auto-decoder framework, which, as shown in Section 2.3.5, cannot be used for sampling. Kleineberg et al. [2020] use a set discriminator to learn distributions of signed distance functions for 3D shape modeling. However, we show both theoretically (see appendix) and empirically (see Section 2.5) that using such a set discriminator severely limits the ability of the model to learn complex function distributions. Cai et al. [2020] represent functions implicitly by gradient fields and use Langevin sampling to generate point clouds. Spurek et al. [2020] learn a function mapping a latent vector to a point cloud coordinate, which is

used for point cloud generation. In addition, several recent works have tackled the problem of learning distributions of NeRF scenes [Mildenhall et al., 2020], which are special cases of implicit representations. This includes GRAF [Schwarz et al., 2020] which concatenates a latent vector to an implicit representation and trains the model adversarially using a patch-based convolutional discriminator, GIRAFFE [Niemeyer and Geiger, 2021] which adds compositionality to the generator and pi-GAN [Chan et al., 2021b] which models the generator using modulations to the hidden layer activations. Finally, while some of these works show basic results on small scale image datasets, GASP is, to the best of our knowledge, the first work to show how function distributions can be used to model a very general class of data, including images, 3D shapes and data lying on manifolds.

2.5 Experiments

We evaluate our model on CelebAHQ [Karras et al., 2018] at 64×64 and 128×128 resolution, on 3D shapes from the ShapeNet [Chang et al., 2015] chairs category and on climate data from the ERA5 dataset [Hersbach et al., 2019]. For all datasets we use the *exact same model* except for the input and output dimensions of the function representation and the parameters of the Fourier features. Specifically, we use an MLP with 3 hidden layers of size 128 for the function representation and an MLP with 2 hidden layers of size 256 and 512 for the hypernetwork. Remarkably, we find that such a simple architecture is sufficient for learning rich distributions of images, 3D shapes and climate data.

The point cloud discriminator is loosely based on the DCGAN architecture [Radford et al., 2015]. Specifically, for coordinates of dimension d, we use 3^d neighbors for each PointConv layer and downsample points by a factor of 2^d at every pooling layer while doubling the number of channels. We implemented our model in PyTorch [Paszke et al., 2019] and performed all training on a single 2080Ti GPU with 11GB of RAM. The code can be found at `https://github.com/EmilienDupont/neural-function-distributions`.

Figure 2.6: Samples from our model trained on CelebAHQ 64×64 (top) and 128×128 (bottom). Each image corresponds to a function which was sampled from our model and then evaluated on the grid. To produce this figure we sampled 5 batches and chose the best batch by visual inspection.

2.5.1 Images

We first evaluate our model on the task of image generation. To generate images, we sample a function from the learned model and evaluate it on a grid. As can be seen in in Figure 2.6, GASP produces sharp and realistic images both at 64×64 and 128×128 resolution. While there are artifacts and occasionally poor samples (particularly at 128×128 resolution), the images are generally convincing and show that the model has learned a meaningful distribution of functions representing the data. To the best of our knowledge, this is the first time data of this complexity has been modeled in an entirely continuous fashion.

As the representations we learn are independent of resolution, we can examine the continuity of GASP by generating images at higher resolutions than the data

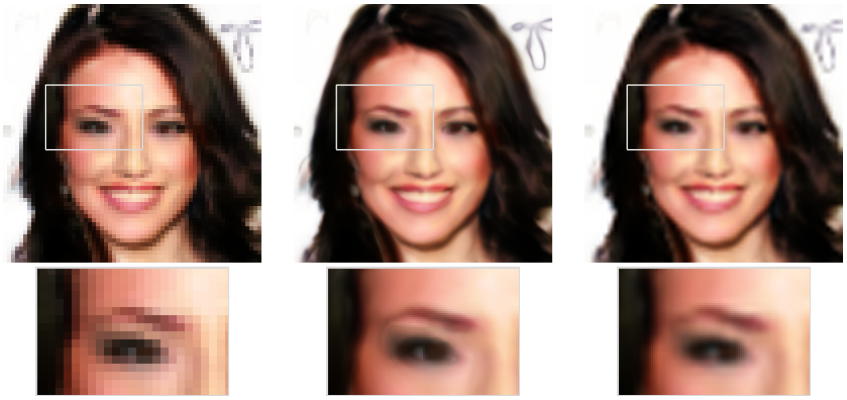

Figure 2.7: Superresolution. The first column corresponds to the original resolution, the second column to 4× the resolution and the third column to bicubic upsampling.

on which it was trained. We show examples of this in Figure 2.7 by first sampling a function from our model, evaluating it at the resolution on which it was trained and then evaluating it at a 4× higher resolution. As can be seen, our model generates convincing 256×256 images even though it has only seen 64×64 images during training, confirming the continuous nature of GASP (see appendix for more examples).

We compare GASP against three baselines: a model trained using the auto-decoder (AD) framework (similar to DeepSDF [Park et al., 2019]), a model trained with a set discriminator (SD) (similar to Kleineberg et al. [2020]) and a convolutional neural process (ConvNP) [Gordon et al., 2019]. To the best of our knowledge, these are the only other model families that can learn generative models in a continuous manner, without relying on a grid representation (which is required for regular CNNs). Results comparing all three models on CelebAHQ 32×32 are shown in Figure 2.8. As can be seen, the baselines generate blurry and incoherent samples, while our model is able to generate sharp, diverse and plausible samples. Quantitatively, our model (Table 2.1) outperforms all baselines, although it lags behind state of the art convolutional GANs specialized to images [Lin et al., 2019].

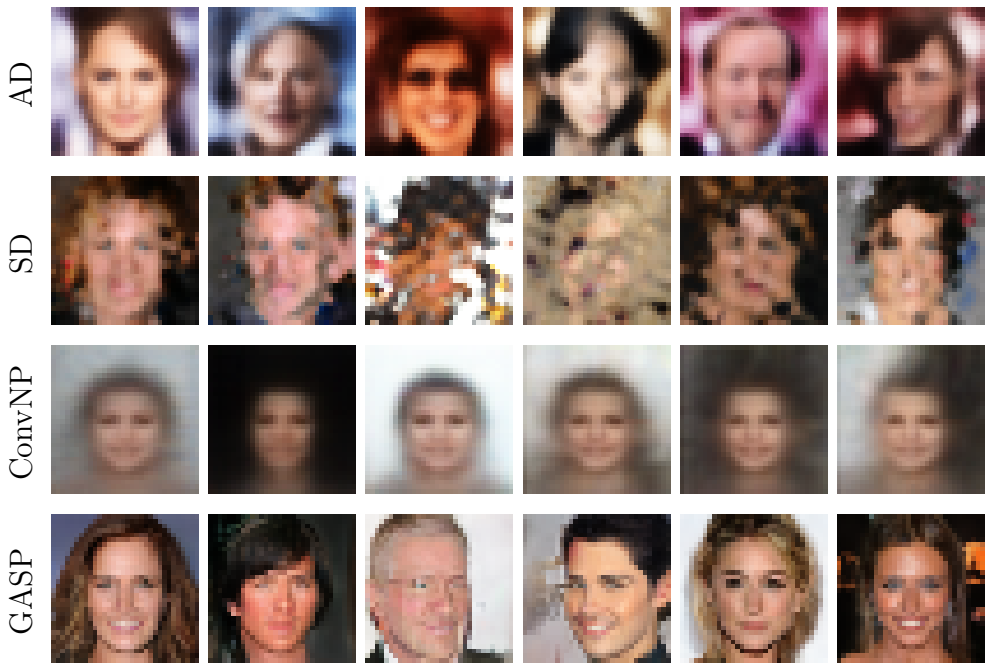

Figure 2.8: Baseline comparisons on CelebAHQ 32×32. Note that the ConvNP model was trained on CelebA (not CelebAHQ) and as such has a different crop.

	CelebAHQ64	CelebAHQ128
SD	236.82	-
AD	117.80	-
GASP	**7.42**	19.16
Conv	4.00	5.74

Table 2.1: FID scores (lower is better) for various models on CelebAHQ datasets, including a standard convolutional image GAN [Lin et al., 2019].

2.5.2 3D scenes

To test the versatility and scalability of GASP, we also train it on 3D shapes. To achieve this, we let the function representation $f_\theta : \mathbb{R}^3 \to \mathbb{R}$ map x, y, z coordinates to an occupancy value p (which is 0 if the location is empty and 1 if it is part of an object). To generate data, we follow the setup from Mescheder et al. [2019]. Specifically, we use the voxel grids from Choy et al. [2016] representing the chairs category from ShapeNet [Chang et al., 2015]. The dataset contains 6778 chairs each of dimension 32^3. As each 3D model is large (a set of $32^3 = 32,768$ points), we uniformly subsample $K = 4096$ points from each object during training, which leads to large memory savings (Figure 2.9) and allows us to train with

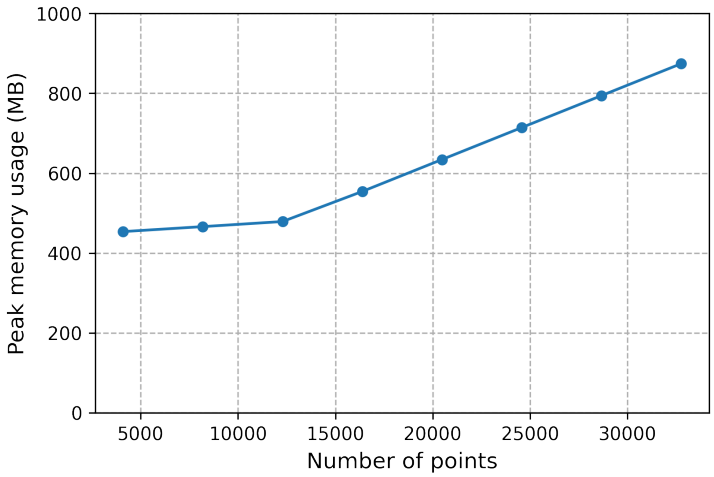

Figure 2.9: GPU memory consumption as a function of the number of points K in voxel grid.

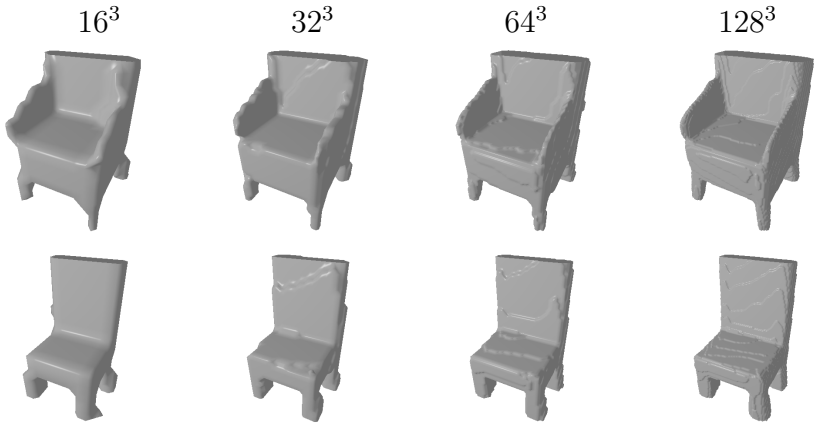

Figure 2.10: Evaluating the same function at different resolutions. As samples from our model can be probed at arbitrary coordinates, we can increase the resolution to render smoother meshes.

large batch sizes even on limited hardware. Crucially, *this is not possible with convolutional discriminators* and is a key property of our model: we can train the model independently of the resolution of the data.

In order to visualize results, we convert the functions sampled from GASP to meshes we can render (see appendix for details). As can be seen in Figure 2.10, the continuous nature of the data representation allows us to sample our model at high resolutions to produce clean and smooth meshes. In Figure 2.11, we compare our model to two strong baselines for continuous 3D shape modeling:

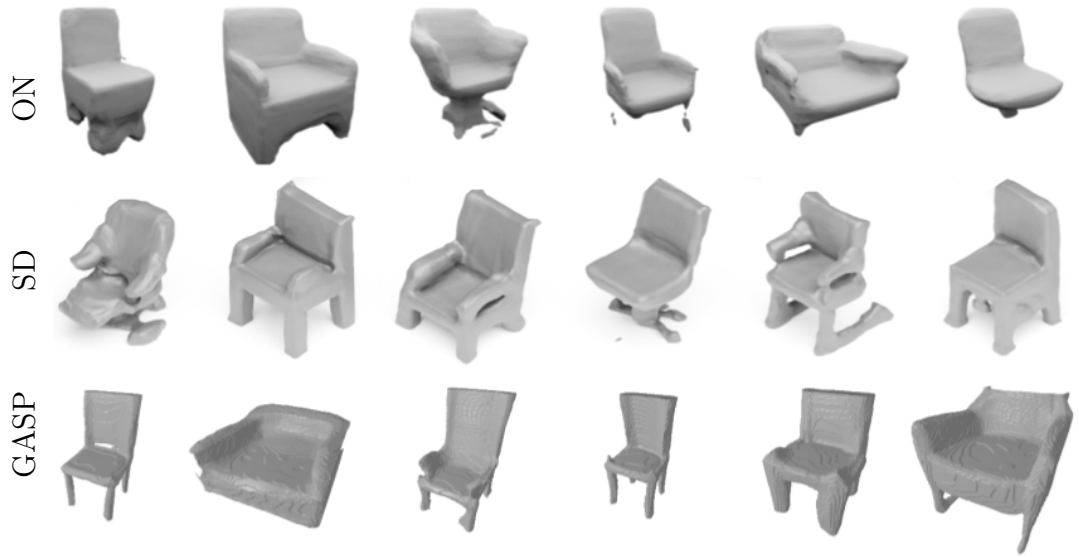

Figure 2.11: Samples from occupancy networks trained as VAEs (ON), DeepSDF with set discriminators (SD) and GASP trained on ShapeNet chairs. The top row samples were taken from Mescheder et al. [2019] and the middle row samples from Kleineberg et al. [2020].

occupancy networks trained as VAEs [Mescheder et al., 2019] and DeepSDFs trained with a set discriminator approach [Kleineberg et al., 2020]. As can be seen, GASP produces coherent and fairly diverse samples, which are comparable to both baselines specialized to 3D shapes.

2.5.3 Climate data

As we have formulated our framework entirely in terms of continuous coordinates and features, we can easily extend GASP to learning distributions of functions on manifolds. We test this by training GASP on temperature measurements over the last 40 years from the ERA5 dataset [Hersbach et al., 2019], where each datapoint is a 46×90 grid of temperatures T measured at evenly spaced latitudes λ and longitudes φ on the globe (see appendix for details). The dataset is composed of 8510 such grids measured at different points in time. We then model each datapoint by a function $f : S^2 \to \mathbb{R}$ mapping points on the sphere to temperatures. We treat the temperature grids as i.i.d. samples and therefore do not model any

temporal correlation, although we could in principle do this by adding time t as an input to our function.

To ensure the coordinates lie on a manifold, we simply convert the latitude-longitude pairs to spherical coordinates before passing them to the function representation, i.e. we set $\mathbf{x} = (\cos\lambda\cos\varphi, \cos\lambda\sin\varphi, \sin\lambda)$. We note that, in contrast to standard discretized approaches which require complicated models to define convolutions on the sphere [Cohen et al., 2018, Esteves et al., 2018], we only need a coordinate system on the manifold to learn distributions.

While models exist for learning conditional distributions of functions on manifolds using Gaussian processes [Borovitskiy et al., 2021, Jensen et al., 2020], we are not aware of any work learning unconditional distributions of such functions for sampling. As a baseline we therefore compare against a model trained directly on the grid of latitudes and longitudes (thereby ignoring that the data comes from a manifold). Samples from our model as well as comparisons with the baseline and an example of interpolation in function space are shown in Figure 2.12. As can be seen, GASP generates plausible samples, smooth interpolations and, unlike the baseline, is continuous across the sphere.

2.6 Scope, limitations and future work

Limitations. While learning distributions of functions gives rise to very flexible generative models applicable to a wide variety of data modalities, GASP does not outperform state of the art specialized image and 3D shape models. We strived for simplicity when designing our model but hypothesize that standard GAN tricks [Karras et al., 2018, 2019, Arjovsky et al., 2017, Brock et al., 2019] could help narrow this gap in performance. In addition, we found that training could be unstable, particularly when subsampling points. On CelebAHQ for example, decreasing the number of points per example also decreases the quality of the generated images (see appendix for samples and failure examples), while the 3D model typically collapses to generating simple shapes (e.g. four legged chairs) even if the data contains complex shapes (e.g. office chairs). We conjecture that this is due to the

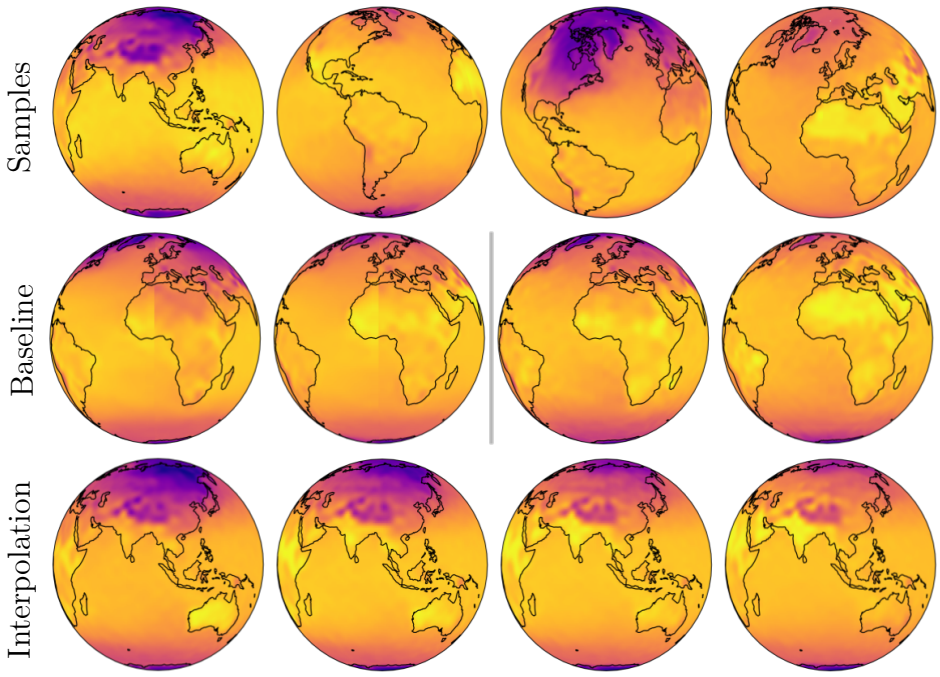

Figure 2.12: Results on climate data. The top row shows samples from our model. The middle row shows comparisons between GASP (on the right) and a baseline (on the left) trained on a grid. As can be seen, the baseline generates discontinuous samples at the grid boundary unlike GASP which produces smooth samples. The bottom row shows a latent interpolation corresponding roughly to interpolating between summer and winter in the northern hemisphere.

nearest neighbor search in the discriminator: when subsampling points, a nearest neighbor may lie very far from a query point, potentially leading to unstable training. More refined sampling methods and neighborhood searches should help improve stability. Finally, determining the neighborhood for the point cloud convolution can be expensive when a large number of points is used, although this could be mitigated with faster neighbor search [Johnson et al., 2019].

Future work. As our model formulation is very flexible, it would be interesting to apply GASP to geospatial [Jean et al., 2016], geological [Dupont et al., 2018], meteorological [Sønderby et al., 2020] or molecular [Wu et al., 2018] data which typically do not lie on a regular grid. In computer vision, we hope our approach will help scale generative models to larger datasets. While our model in its current form could not scale to truly large datasets (such as room scale 3D scenes), framing generative models entirely in terms of coordinates and features could be a first

step towards this. Indeed, while grid-based generative models currently outperform continuous models, we believe that, at least for certain data modalities, continuous models will eventually surpass their discretized counterparts.

2.7 Conclusion

In this paper, we introduced GASP, a method for learning generative models that act entirely on continuous spaces and as such are independent of signal discretization. We achieved this by learning distributions over functions representing the data instead of learning distributions over the data directly. Through experiments on images, 3D shapes and climate data, we showed that our model learns rich function distributions in an entirely continuous manner. We hope such a continuous approach will eventually enable generative modeling on data that is not currently tractable, either because discretization is expensive (such as in 3D) or difficult (such as on non-euclidean data).

Acknowledgements

We thank William Zhang, Yuyang Shi, Jin Xu, Valentin De Bortoli, Jean-Francois Ton and Kaspar Märtens for providing feedback on an early version of the paper. We also thank Charline Le Lan, Jean-Francois Ton and Bobby He for helpful discussions. We thank Yann Dubois for providing the ConvNP samples as well as helpful discussions around neural processes. We thank Shahine Bouabid for help with the ERA5 climate data. Finally, we thank the anonymous reviewers and the ML Collective for providing constructive feedback that helped us improve the paper. Emilien gratefully acknowledges his PhD funding from Google DeepMind.

Appendix

2.A Experimental details

In this section we provide experimental details necessary to reproduce all results in the paper. All the models were implemented in PyTorch Paszke et al. [2019] and trained on a single 2080Ti GPU with 11GB of RAM. The code to reproduce all experiments can be found at `https://github.com/EmilienDupont/neural-function-distributions`.

2.A.1 Single image experiment

To produce Figure 2.2, we trained a ReLU MLP with 2 hidden layers each with 256 units, using tanh as the final non-linearity. We trained for 1000 iterations with Adam using a learning rate of 1e-3. For the RFF encoding we set $m = 256$ and $\sigma = 10$.

2.A.2 GASP experiments

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

For the point cloud discriminator, we used 3^d neighbors in each convolution layer and followed every convolution by an average pooling layer reducing the number of points by 2^d. We applied a sigmoid as the final non-linearity. We used an MLP with 4 hidden layers each of size 16 to parameterize all weight MLPs. Unless stated otherwise, we use Adam with a learning rate of 1e-4 for the hypernetwork

weights and 4e-4 for the discriminator weights with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ as is standard for GAN training. For each dataset, we trained for a large number of epochs and chose the best model by visual inspection.

MNIST

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

CelebAHQ 64x64

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

CelebAHQ 128x128

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

ShapeNet voxels

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

ERA5 climate data

- Dimensions: $d = 2$, $k = 1$
- Fourier features: $m = 128$, $\sigma = 2$
- Discriminator channels: $64, 128, 256, 512$
- Batch size: 64
- Epochs: 300

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

- We initially let the function representation f_θ have 2 hidden layers of size 256, instead of 3 layers of size 128. However, we found that this did not work well, particularly for more complex datasets. We hypothesize that this is because the number of weights in a single $256 \to 256$ linear layer is $4\times$ the number of weights in a single $128 \to 128$ layer. As such, the number of weights in four $128 \to 128$ layers is the same as a single $256 \to 256$, even though such a 4-layer network would be much more expressive. Since the hypernetwork needs to output all the weights of the function representation, the final layer of the hypernetwork will be extremely large if the number of function weights is large. It is therefore important to make the network as expressive as possible with as few weights as possible, i.e. by making the network thinner and deeper.

- As the paper introducing the R_1 penalty [Mescheder et al., 2018] does not use batchnorm [Ioffe and Szegedy, 2015] in the discriminator, we initially ran experiments without using batchnorm. However, we found that using batchnorm both in the weight MLPs and between PointConv layers was crucial for stable training. We hypothesize that this is because using standard initializations for the weights of PointConv layers would result in PointConv outputs (which correspond to the weights in regular convolutions) that are large. Adding batchnorm fixed this initialization issue and resulted in stable training.

- In the PointConv paper, it was shown that the number of hidden layers in the weight MLPs does not significantly affect classification performance [Wu et al., 2019]. We therefore initially experimented with single hidden layer MLPs for the weights. However, we found that it is crucial to use deep networks for the weight MLPs in order to build discriminators that are expressive enough for the datasets we consider.

- We experimented with learning the frequencies of the Fourier features (i.e. learning B) but found that this did not significantly boost performance and generally resulted in slower training.

2.A.4 ERA5 climate data

We extracted the data used for the climate experiments from the ERA5 database [Hersbach et al., 2019]. Specifically, we used the monthly averaged surface temperature at 2m, with reanalysis by hour of day. Each data point then corresponds to a set of temperature measurements on a 721 x 1440 grid (i.e. 721 latitudes and 1440 longitudes) across the entire globe (corresponding to measurements every 0.25 degrees). For our experiments, we subsample this grid by a factor of 16 to obtain grids of size 46 x 90. For each month, there are a total of 24 grids, corresponding to each hour of the day. The dataset is then composed of temperature measurements for all months between January 1979 and December 2020, for a total of 12096

datapoints. We randomly split this dataset into a train set containing 8510 grids, a validation set containing 1166 grids and a test set containing 2420 grids. Finally, we normalize the data to lie in $[0, 1]$ with the lowest temperature recorded since 1979 corresponding to 0 and the highest temperature to 1.

2.A.5 Quantitative experiments

We computed FID scores using the `pytorch-fid` library [Seitzer, 2020]. We generated 30k samples for both CelebAHQ 64×64 and 128×128 and used default settings for all hyperparameters. We note that the FID scores for the convolutional baselines in the main paper were computed on CelebA (not the HQ version) and are therefore not directly comparable with our model. However, convolutional GANs would also outperform our model on CelebAHQ.

2.A.6 Rendering 3D shapes

In order to visualize results for the 3D experiments, we convert the functions sampled from GASP to meshes we can render. To achieve this, we first sample a function from our model and evaluate it on a high resolution grid (usually 128^3). We then threshold the values of this grid at 0.5 (we found the model was robust to choices of threshold) so voxels with values above the threshold are occupied while the rest are empty. Finally, we use the marching cubes algorithm [Lorensen and Cline, 1987] to convert the grid to a 3D mesh which we render with PyTorch3D [Ravi et al., 2020].

2.A.7 Baseline experiments

The baseline models in Section 2.5.1 were trained on CelebAHQ 32×32, using the same generator as the one used for the CelebAHQ 64×64 experiments. Detailed model descriptions can be found in Section 2.B and hyperparameters are provided below.

Auto-decoders. We used a batch size of 64 and a learning rate of 1e-4 for both the latents and the generator parameters. We sampled the latent initializations

Figure 2.13: Left: Samples from an auto-decoder model trained on MNIST. Right: Samples from an auto-decoder model trained on CelebAHQ 32×32.

from $\mathcal{N}(0, 0.01^2)$. We trained the model for 200 epochs and chose the best samples based on visual inspection.

Set Discriminators. We used a batch size of 64, a learning rate of 1e-4 for the generator and a learning rate of 4e-4 for the discriminator. We used an MLP with dimensions [512, 512, 512] for the set encoder layers and an MLP with dimensions [256, 128, 64, 32, 1] for the final discriminator layers. We used Fourier features with $m = 128$, $\sigma = 2$ for both the coordinates and the features before passing them to the set discriminator. We trained the model for 200 epochs and chose the best samples based on visual inspection.

2.B Models that are not suitable for learning function distributions

2.B.1 Auto-decoders

We briefly introduce auto-decoder models following the setup in [Park et al., 2019] and describe why they are not suitable as generative models. As in the GASP case, we assume we are given a dataset of N samples $\{\mathbf{s}^{(i)}\}_{i=1}^{N}$ (where each sample $\mathbf{s}^{(i)}$ is a set). We then associate a latent vector $\mathbf{z}^{(i)}$ with each sample $\mathbf{s}^{(i)}$. We further parameterize a probabilistic model $p_\theta(\mathbf{s}^{(i)}|\mathbf{z}^{(i)})$ (similar to the decoder in variational autoencoders) by a neural network with learnable parameters θ (typically returning the mean of a Gaussian with fixed variance). The optimal parameters are then estimated as

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

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

We hypothesize that this is why the prior is required to have very low variance for the auto-decoder model to work well [Park et al., 2019]. Indeed, if the norm of the $\mathbf{z}^{(i)}$'s is so small that they are barely changed during training, they will remain close to their initial Gaussian distribution. While this trick is sufficient to learn distributions of simple datasets such as MNIST, we were unable to obtain good results on more complex and high frequency datasets such as CelebAHQ. Results of our best models are shown in Figure 2.13.

We also note that auto-decoders were not necessarily built to act as generative models. Auto-decoders have for example excelled at embedding 3D shape data into a latent space [Park et al., 2019] and learning distributions over 3D scenes for inverse rendering [Sitzmann et al., 2019b]. Our analysis therefore does not detract from the usefulness of auto-decoders, but instead shows that auto-decoders may not be suitable for the task of generative modeling.

2.B.2 Set discriminators

In this section, we analyse the use of set discriminators for learning function distributions. Given a datapoint $\mathbf{s} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ represented as a set, we build a permutation invariant set discriminator as a PointNet/DeepSet [Qi et al., 2017,

Zaheer et al., 2017] function

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

where ρ and φ are both MLPs and γ_x and γ_y are RFF encodings for the coordinates and features respectively. Recall that the RFF encoding function γ is defined as

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

where B is a (potentially learnable) random matrix of frequencies. While the RFF encodings are not strictly necessary, we were unable to learn high frequency functions without them. Note also that we normalize the sum over set elements by $\sqrt{n}$ instead of n as is typical - as shown in Section 2.B.3 this is to make the Lipschitz constant of the set discriminator independent of n.

We experimented extensively with such models, varying architectures and encoding hyperparameters (including not using an encoding) but were unable to get satisfactory results on CelebAHQ, even at a resolution of 32×32. Our best results are shown in Figure 2.14. As can be seen, the model is able to generate plausible samples for MNIST but fails on CelebAHQ.

While PointNet/DeepSet functions are universal approximators of set functions [Zaheer et al., 2017], they do not explicitly model set element interactions. As such, we also experimented with Set Transformers [Lee et al., 2019b] which model interactions using self-attention. However, we found that using such architectures did not improve performance. As mentioned in the main paper, we therefore conjecture that explicitly taking into account the metric on the coordinate space (as is done in PointConv) is crucial for learning complex neural distributions. We note that Set Transformers have also been used as a discriminator to model sets [Stelzner et al., 2020], although this was only done for small scale datasets.

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

Figure 2.14: Left: Samples from a set discriminator model trained on MNIST. Right: Samples from a set discriminator model trained on CelebAHQ 32×32.

2.B.3 The Lipschitz constant of set discriminators

Several works have shown that limiting the Lipschitz constant (or equivalently the largest gradient norm) of the discriminator is important for stable GAN training [Arjovsky et al., 2017, Gulrajani et al., 2017, Roth et al., 2017, Miyato et al., 2018, Mescheder et al., 2018]. This is typically achieved either by penalizing the gradient norm or by explicitly constraining the Lipschitz constant of each layer in the discriminator. Intuitively, this ensures that the gradients of the discriminator with respect to its input do not grow too large and hence that gradients with respect to the weights of the generator do not grow too large either (which can lead to unstable training). In the following subsections, we show that the Lipschitz constant of set discriminators and specifically the Lipschitz constant of RFF encodings are large in most realistic settings.

Architecture

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

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

See Section 2.C for a proof. In the case where the RFF encoding is fixed, imposing gradient penalties on D would therefore reduce the Lipschitz constant of ρ and φ but not of γ_x and γ_y. If the RFF encoding is learned, its Lipschitz constant could also be penalized. However, as shown in Tancik et al. [2020], learning high frequency functions typically requires large frequencies in the matrix

B. We show in the following section that the Lipschitz constant of γ is directly proportional to the spectral norm of B.

Lipschitz Constant of Random Fourier Features

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

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

See Section 2.C for a proof. There is therefore a fundamental tradeoff between how much high frequency detail the discriminator can learn (requiring a large Lipschitz constant) and its training stability (requiring a low Lipschitz constant). In practice, for the settings we used in this paper, the spectral norm of B is on the order of 100s, which is too large for stable GAN training.

2.C Proofs

2.C.1 Prerequisites

We denote by $\|\cdot\|_2$ the ℓ_2 norm for vectors and by $\|\cdot\|$ the spectral norm for matrices (i.e. the matrix norm induced by the ℓ_2 norm). The spectral norm is defined as

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

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

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

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

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

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

Note that when composing two functions f and g we have

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

We will also make use of the following lemmas.

Spectral norm of concatenation

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

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

Proof.[1]

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

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

Inequality for ℓ_1 and ℓ_2 norm

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

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

Proof.

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

where we used Cauchy-Schwarz in the second line. Note that this is an extension of the well-known inequality $\|\mathbf{x}\|_1 \leq \sqrt{n}\|\mathbf{x}\|_2$ to the case where each component of the vector $\mathbf{x}$ is the ℓ_2 norm of another vector.

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

Lipschitz constant of sum of identical functions

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

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

Proof.

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

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

Lipschitz constant of concatenation

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

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

Proof.

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

where we used the definition of the ℓ_2 norm in the second and last line.

2.C.2 Lipschitz constant of Fourier feature encoding

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.C.3 Lipschitz constant of set discriminator

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

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

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

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

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

Proof. Write

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

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

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

We can further write

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

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

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

We can then write

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

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

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

Putting everything together we finally obtain

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

2.D Failure examples

Figure 2.15: Left: Samples from model trained on CelebAHQ 64×64 using $K = 2048$ pixels (50%). Right: Samples from model trained using $K = 3072$ pixels (75%).

Figure 2.16: Selected samples highlighting failure modes of our model, including generation of unrealistic and incoherent samples.

2.E Additional results

2.E.1 Additional evaluation on ERA5 climate data

As metrics like FID are not applicable to the ERA5 data, we provide additional experimental results to strengthen the evaluation of GASP on this data modality. Figure 2.22 shows comparisons between samples from GASP and the training data. As can be seen, the samples produced from our model are largely indistinguishable from real samples. To ensure the model has not memorized samples from the training set, but rather has learned a smooth manifold of the data, we show examples of latent interpolations in Figure 2.23. Finally, Figure 2.24 shows a histogram comparing the distribution of temperatures in the test set and the distribution of temperatures obtained from GASP samples.

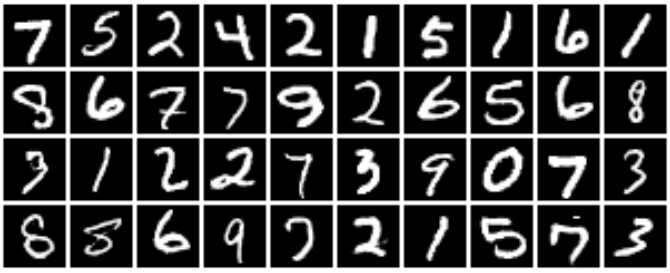

Figure 2.17: Additional MNIST samples.

Figure 2.18: Additional CelebAHQ 64×64 samples.

Figure 2.19: Additional CelebAHQ 128×128 samples.

Figure 2.20: Additional superresolution samples. Left column shows superresolution from $64 \times 64 \rightarrow 256 \times 256$ and right column shows superresolution from $64 \times 64 \rightarrow 512 \times 512$

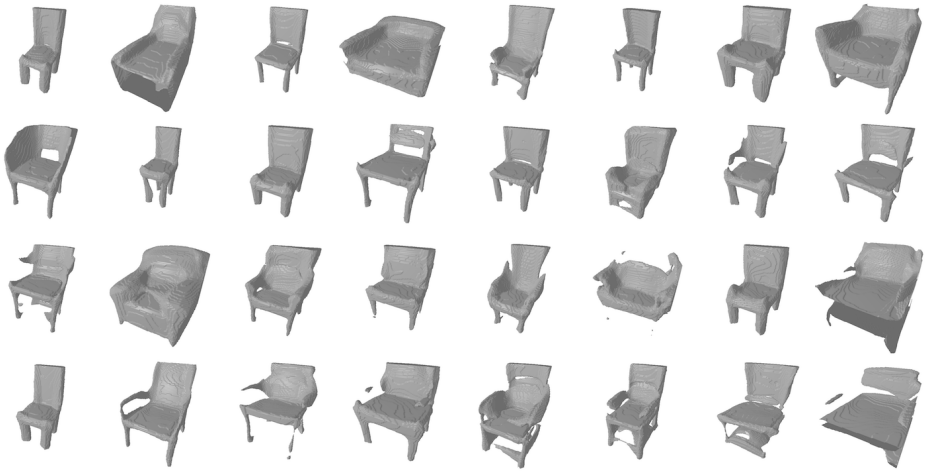

Figure 2.21: Additional Shapenet chairs samples.

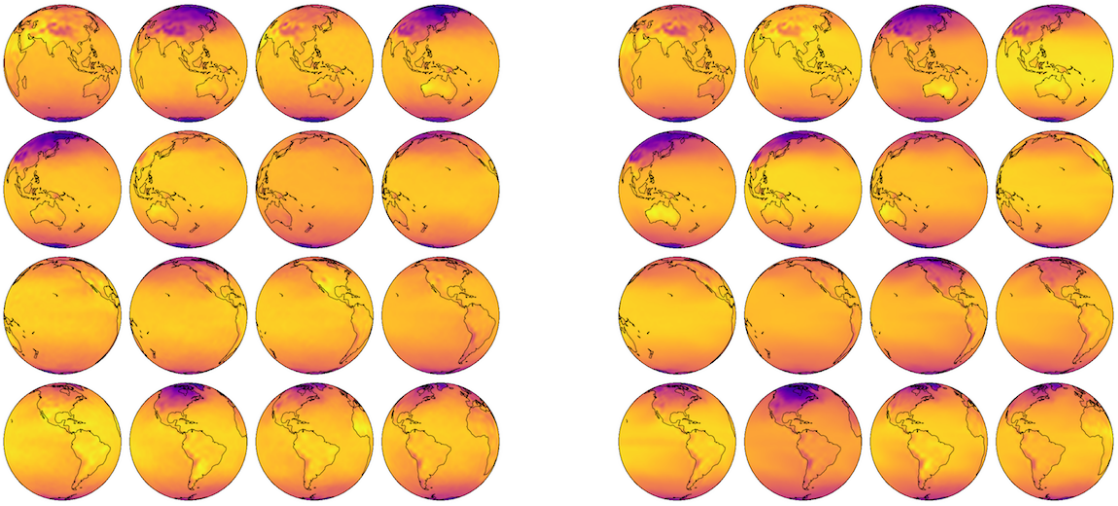

Figure 2.22: Random samples from GASP (left) and the training data (right).

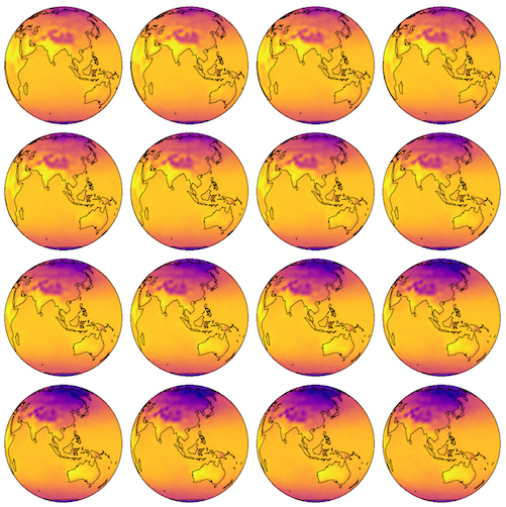

Figure 2.23: Latent (function space) interpolation between two random samples from GASP. As can be seen the the model has learned a smooth latent space for the data.

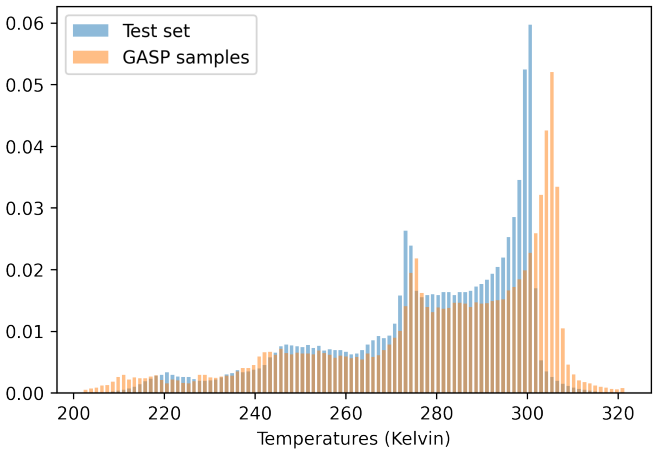

Figure 2.24: Distribution of temperatures in test set and from GASP samples. As can be seen, the distribution of temperatures from GASP roughly matches the distribution in the test set.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Generative Models as Distributions of Functions
Publication Status	X Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	*Emilien Dupont, Yee Whye Teh, Arnaud Doucet*, Generative Models as Distributions of Functions, AISTATS 2022

Student Confirmation

Student Name:	Emilien Dupont		
Contribution to the Paper	I had the idea for the project which was refined in discussions with Arnaud and Yee Whye. I wrote the code, performed all experiments and wrote the paper. Arnaud and Yee Whye provided feedback and edits on the write up.		
Signature		Date	September 19, 2022

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: *Professor Yee Whye Teh*			
Supervisor comments			
Signature		Date	23 Sep 2022

3

COIN: COmpression with Implicit Neural representations

Abstract

We propose a new simple approach for image compression: instead of storing the RGB values for each pixel of an image, we store the weights of a neural network overfitted to the image. Specifically, to encode an image, we fit it with an MLP which maps pixel locations to RGB values. We then quantize and store the weights of this MLP as a code for the image. To decode the image, we simply evaluate the MLP at every pixel location. We found that this simple approach outperforms JPEG at low bit-rates, *even without entropy coding or learning a distribution over weights.* While our framework is not yet competitive with state of the art compression methods, we show that it has various attractive properties which could make it a viable alternative to other neural data compression approaches.

3.1 Introduction

Neural image compression methods typically operate in an autoencoder setup [Ballé et al., 2018, Minnen et al., 2018, Lee et al., 2019a]. The sender uses an encoder to map the input data to a discretized latent code, which is then entropy coded into a bitstream according to a learned latent distribution. The bitstream is transmitted to the receiver that decodes it into a latent code, which is finally passed through the decoder to reconstruct the image.

In this paper, we take a different approach: we encode an image by overfitting it with a small MLP mapping pixel locations to RGB values and then transmit the weights θ of this MLP as a code for the image (see Figure 3.1). While overfitting such MLPs, referred to as implicit neural representations, is difficult due to the high frequency information contained in natural images [Basri et al., 2019, Tancik et al., 2020], recent research has shown that this can be mitigated by using sinusoidal encodings and activations [Mildenhall et al., 2020, Tancik et al., 2020, Sitzmann et al., 2020b]. In this work, we show that using MLPs with sine activations, often referred to as SIRENs [Sitzmann et al., 2020b], we can fit large images (393k pixels) with surprisingly small networks (8k parameters).

We evaluate our method on standard image compression tasks and show that we outperform JPEG at low bit-rates, even without entropy coding or learning a distribution over weights. As implicit representations have successfully been applied in the context of generative modeling [Dupont et al., 2022c], it is likely that

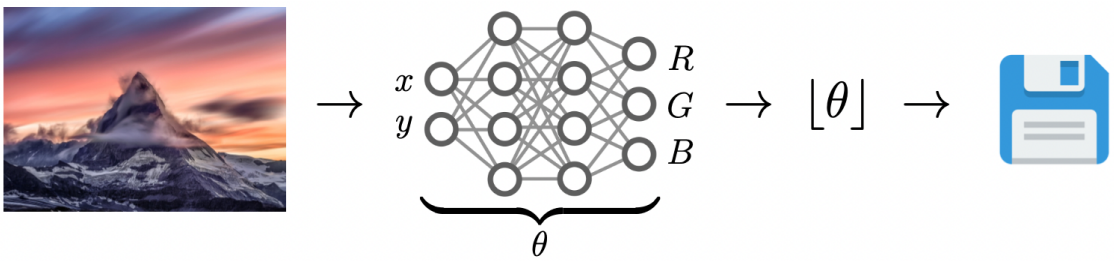

Figure 3.1: Compressed implicit neural representations. We overfit an image with a neural network mapping pixel locations (x, y) to RGB values (often referred to as an implicit neural representation). We then quantize the weights θ of this neural network to a lower bit-width and transmit them.

combining our approach with a learned weight distribution could lead to promising new approaches for neural data compression. Further, by treating our image as a function from pixel locations to RGB values, we can perform progressive decoding simply by evaluating our function at progressively higher resolutions, which is particularly attractive for resource constrained receiving devices.

3.2 Method

In this section, we describe COmpressed Implicit Neural representations (COIN), our proposed method for image compression. The encoding step consists in overfitting an MLP to the image, quantizing its weights and transmitting these. At decoding time, the transmitted MLP is evaluated at all pixel locations to reconstruct the image.

3.2.1 Encoding

Let I denote the image we wish to encode, such that $I[x, y]$ returns the RGB values at pixel location (x, y). We define a function $f_\theta : \mathbb{R}^2 \to \mathbb{R}^3$ with parameters θ mapping pixel locations to RGB values in the image, i.e., $f_\theta(x, y) = (r, g, b)$. We can then encode the image by overfitting f_θ to the image under some distortion measure. In this paper, we use the mean squared error, resulting in the following optimization problem

$$\min_\theta \sum_{x,y} \| f_\theta(x, y) - I[x, y] \|_2^2, \tag{3.1}$$

where the sum is over all pixel locations.

Choosing the parameterization of f_θ is crucial. Indeed, parameterizing f_θ by an MLP with standard activation functions results in underfitting, even when using a large number of parameters [Tancik et al., 2020, Sitzmann et al., 2020b]. This problem can be overcome in multiple ways, e.g. by encoding pixel coordinates with Fourier features [Tancik et al., 2020] or by using sine activation functions [Sitzmann et al., 2020b]. Empirically, we found that the latter option yielded better results for a given parameter budget.

Minimizing equation (3.1) is trivial given a large enough MLP. However, we store the parameters θ of the MLP as the compressed description of the image, so restricting the number of weights will improve the compression rate. The goal is therefore to fit f_θ to I (i.e., minimize distortion) using the fewest parameters possible (i.e., minimizing rate). Our approach then effectively converts a data compression problem to a model compression problem.

To reduce the model size, we consider two approaches: architecture search and weight quantization. More specifically, we perform a hyperparameter sweep over the width and number of layers of the MLP and quantize the weights from 32-bit to 16-bit precision, which was sufficient to outperform the JPEG standard for low bit-rates. However, we believe that more sophisticated approaches to architecture search [Elsken et al., 2019] and especially model compression [Ullrich et al., 2017, Havasi et al., 2019, van Baalen et al., 2020] will further improve results.

3.2.2 Decoding

Given the stored quantized weights θ, decoding simply consists in evaluating the function f_θ at every pixel location to reconstruct the image. This decoding approach gives us extra flexibility: we can progressively decode the image, e.g. by decoding parts of the image or a low resolution image first, simply by evaluating the function at various pixel locations. Partially decoding images in this way is difficult with autoencoder based methods, showing a further advantage of the COIN approach.

3.3 Related Work

Implicit neural representations. Representing data with neural networks was originally proposed by [Stanley, 2007] but has seen a recent surge in interest in the 3D vision community [Park et al., 2019, Niemeyer et al., 2019, Chen and Zhang, 2019]. Motivated by the fact that memory requirements of deep voxel representations grow cubically [Nguyen-Phuoc et al., 2018, Sitzmann et al., 2019a, Dupont et al., 2020], implicit representations were proposed to compactly encode high resolution signals [Mildenhall et al., 2020, Tancik et al., 2020, Sitzmann et al., 2020b]. While

the MLPs used to represent images typically have a relatively small number of parameters [Dupont et al., 2022c], we take this even further and show that by carefully choosing the architecture of the MLP and quantizing the weights, we can fit images with MLPs that take up significantly less space than storing RGB values.

Neural data compression. Learned image compression methods are commonly based on hierarchical variational autoencoders [Ballé et al., 2018, Minnen et al., 2018, Lee et al., 2019a] with a learned prior and latent variables being discretized for the purpose of entropy coding. In a similar vein to work in the latent variable model literature [Hjelm et al., 2016, Krishnan et al., 2018, Kim et al., 2018, Marino et al., 2018], several works [Campos et al., 2019, Guo et al., 2020, Yang et al., 2020] attempt to close the amortization gap [Cremer et al., 2018] by performing iterative gradient-based optimization steps on top of the use of amortized inference networks. [Yang et al., 2020] additionally identify and attempt to close the discretization gap stemming from the quantization of the latent variables, also by inference time per-instance optimization. [van Rozendaal et al., 2021] take the idea of per-instance optimization of the model further: they perform per-instance finetuning of the decoder and transmit the quantized decoder parameter updates along with the latent code, leading to improved rate-distortion performance. In this paper, we take a different, and even more extreme from the per-instance optimization perspective, approach: we optimize an MLP to overfit a single image and transmit its weights as the compressed description of the image.

Model compression. While it is known that the problems of data and model compression are very closely related, COIN explicitly casts the problem of data compression into a problem of model compression. There exists a rich body of literature on model compression, [Ullrich et al., 2017, Louizos et al., 2017, Havasi et al., 2019, van Baalen et al., 2020, Krishnamoorthi, 2018b, Jacob et al., 2018], which could likely be used to improve COIN's performance.

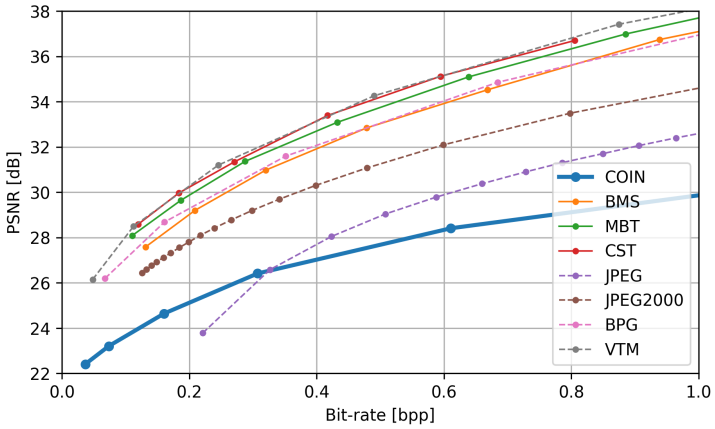

Figure 3.2: Rate distortion plots on the Kodak dataset. Solid lines represent learned compression methods and dashed lines represent standard image codes.

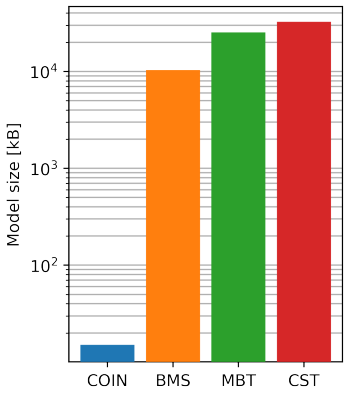

Figure 3.3: Model sizes at 0.3bpp.

3.4 Experiments

We perform experiments on the Kodak image dataset [Kodak, 1991] consisting of 24 images of size 768×512. We compare our model against three autoencoder based neural compression baselines which we refer to as BMS [Ballé et al., 2018], MBT [Minnen et al., 2018], and CST [Cheng et al., 2020b]. We also compare against the JPEG, JPEG2000, BPG and VTM image codecs. To benchmark our model, we use the CompressAI library [Bégaint et al., 2020] and the pre-trained models provided therein. We implement our model in PyTorch [Paszke et al., 2019] and perform all experiments on a single RTX2080Ti GPU.

Rate-distortion plots. To determine the best model architectures for a given parameter budget (measured in bits per pixel or bpp[1]), we first find valid combinations of depth and width for the MLPs representing an image. For example, for 0.3bpp using 16-bit weights, valid networks include MLPs with 10 layers of width 28, 7 layers of width 34 and so on. We then select the best architecture by running a hyperparameter search over learning rates and valid architectures on a single image using Bayesian optimization (we found that the results of the architecture search transferred well to other images). The resulting model is trained on each image in the

[1] bits-per-pixel $= \frac{\#\text{parameters} \times \text{bits-per-parameter}}{\#\text{pixels}}$

dataset at 32-bit precision and converted to 16-bit precision after training. We note that decreasing the weights' precision from 32-bit to 16-bit after training resulted in almost no distortion increase, but decreasing them further to 8-bit incurred significant amount of distortion, outweighing the benefit of halving the bpp.

Results of this procedure for various bpp levels are shown in Figure 3.2. As can be seen, at low bit-rates our model improves upon JPEG *even without using entropy coding*. While our approach is still far from the state of the art compression methods, we believe the performance of such a simple approach is promising for future work in this direction.

Model size. In contrast to most other neural data compression algorithms, our method does not require a decoder at test time. Indeed, while the latent code representing the compressed image in such methods is small, the decoder model is large (typically much larger than an uncompressed image). As such, the memory required on the decoding device is also large. In our case, we only require the weights of a (very small) MLP on the decoder side, leading to memory requirements that are orders of magnitude smaller. As can be seen in Figure 3.3, at 0.3bpp, our method requires 14kB, whereas other baselines require between 10MB and 40MB.

Encoding optimization dynamics. We show an example of the overfitting procedure in Figure 3.4. As can be seen, COIN outperforms JPEG after 15k iterations and continues improving beyond that. While the optimization can be noisy, we simply save the model with the best PSNR.

Architecture choice. In Figure 3.5, we show the performance of various valid architectures of size 0.3bpp. As can be seen, the quality of compression depends on the architecture choice, with different optimal architectures for different bpp values, see Appendix A for details.

3.5 Scope, limitations and future work

Limitations. The main limitation of our approach is that encoding is slow, because we have to solve an optimization problem for each encoded image. However, paying a significant computational cost upfront to compress content for delivery to many

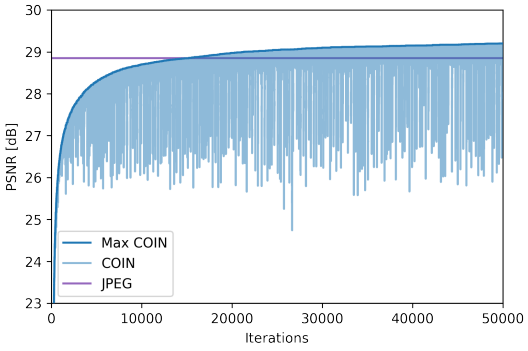

Figure 3.4: Model training on image 15 in the Kodak dataset. Max COIN represents the max PSNR achieved at any point during training.

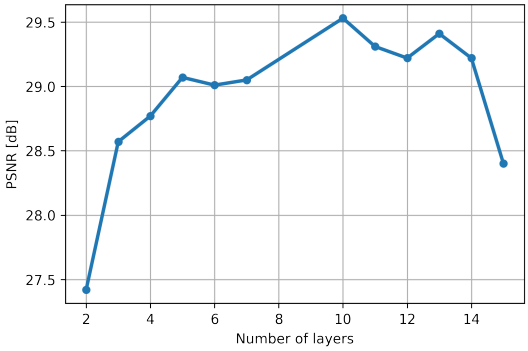

Figure 3.5: Plot of maximum PSNR for networks of the same size (0.3bpp) with different architectures.

receivers is a standard practice in the setting of one-to-many media distribution, e.g., at Netflix [Aaron et al., 2015]. Nevertheless, this limitation could likely be sidestepped with meta-learning [Sitzmann et al., 2020a, Tancik et al., 2021] or amortized inference [Trevithick and Yang, 2020, Yu et al., 2020] approaches. Further, at decoding time, we are required to evaluate the network at every pixel location to decode the full image. However, this computation can be embarrassingly parallelized to the point of a single forward pass for all pixels. Finally, our method performs worse than state of the art compression methods. However, we believe there are several promising directions to reduce this gap.

Future work. Recent work in generative modeling of implicit representations [Dupont et al., 2022c] suggests that learning a distribution over the function weights could translate to significant compression gains for our approach. In addition, exploring meta-learning or other amortization approaches for faster encoding could be an important direction for future work [Sitzmann et al., 2020a, Tancik et al., 2021]. Refining the architectures of the functions representing the images (through neural architecture search or pruning for example) is another promising avenue. While we simply converted weights to half-precision in this paper, large gains in performance could likely be made by using more advanced model compression [Havasi et al., 2019, van Baalen et al., 2020, Jacob et al., 2018]. Finally, as implicit representations map arbitrary coordinates to arbitrary features [Tancik et al., 2020,

Sitzmann et al., 2020b, Dupont et al., 2022c], it would be interesting to apply our method to different types of data, such as video or audio.

3.6 Conclusion

In this paper, we proposed COIN, a new method for compressing images by fitting neural networks to pixels and storing the weights of the resulting models. We showed through experiments that this simple approach can outperform JPEG at low bit-rates, even without the use of entropy coding. We hope that further work in this area will lead to a novel class of methods for neural data compression.

Appendix

3.A Experimental details

All models were trained using Adam for 50k iterations. We used MLPs with 2 input dimensions (corresponding to (x, y) coordinates) and 3 output dimensions (corresponding to RGB values). The coordinates were normalized to lie in $[-1, 1]$ and the RGB values were normalized to lie in $[0, 1]$. We used sine non-linearities at every layer except the last and used the initialization described in [Sitzmann et al., 2020b]. We used a learning rate of 2e-4. Below we describe the architectures for each bpp level.

- 0.07bpp. Number of layers: 5, width of layers: 20.
- 0.15bpp. Number of layers: 5, width of layers: 30.
- 0.3bpp. Number of layers: 10, width of layers: 28.
- 0.6bpp. Number of layers: 10, width of layers: 40.
- 1.2bpp. Number of layers: 13, width of layers: 49.

The code to reproduce all experiments in the paper can be found at `https://github.com/EmilienDupont/coin`.

3.B Additional results

In Figure 3.6, we plot the performance of COIN and JPEG at 0.3bpp (since COIN and JPEG perform similarly at this bit-rate) for all images in the Kodak dataset. As can be seen, the distortion values closely follow each other: images that are difficult for COIN to encode are also difficult for JPEG to encode.

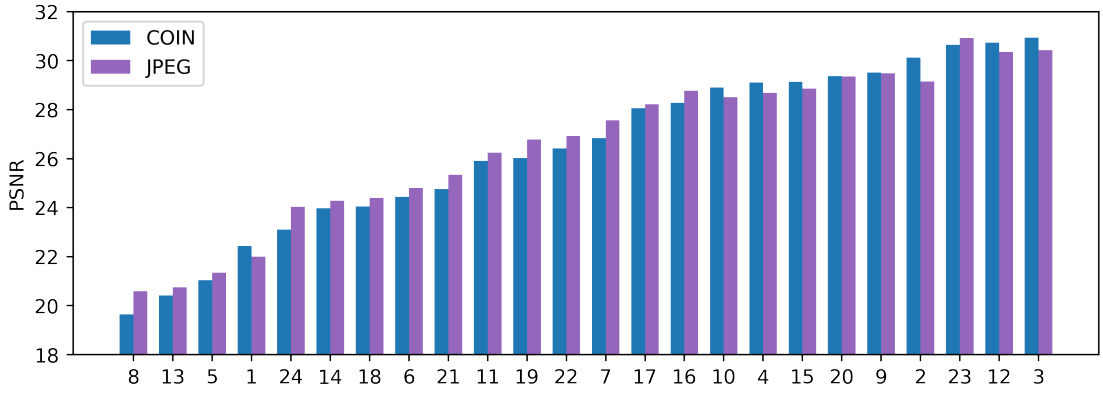

Figure 3.6: Histogram of PSNR for all images in the Kodak dataset for COIN and JPEG.

3.C Qualitative results

We include qualitative results comparing the compression artifacts from COIN and JPEG on the Kodak dataset.

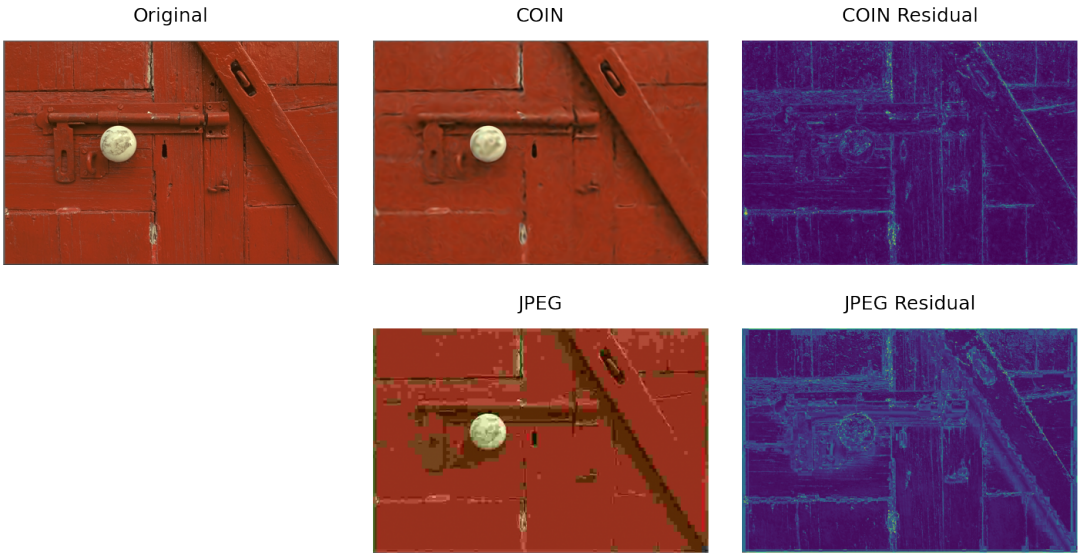

Figure 3.7: Comparison between COIN and JPEG on image 2 at 0.15bpp. The PSNRs are 28.69dB and 24.67dB respectively.

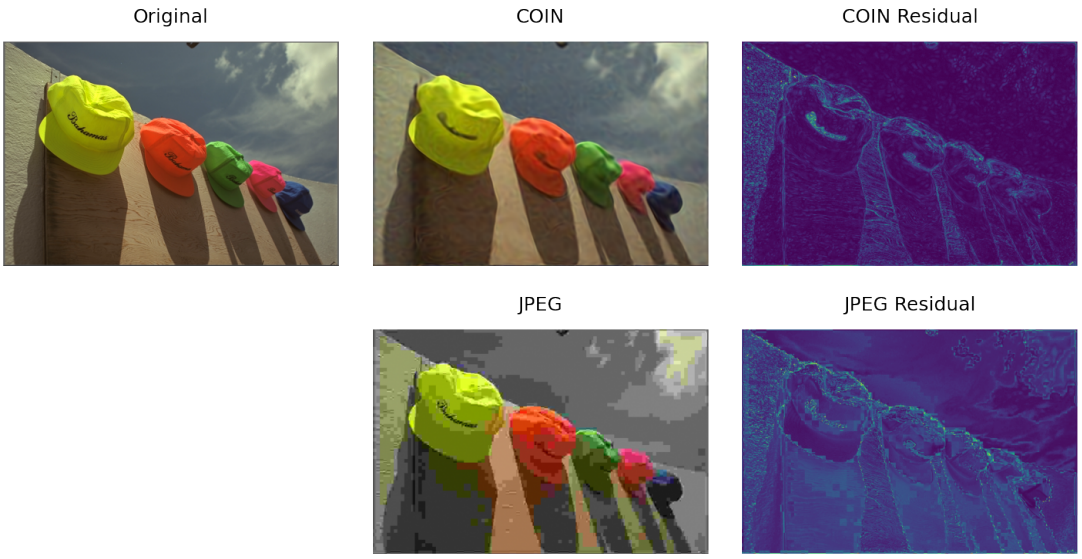

Figure 3.8: Comparison between COIN and JPEG on image 3 at 0.15bpp. The PSNRs are 29.02dB and 23.63dB respectively.

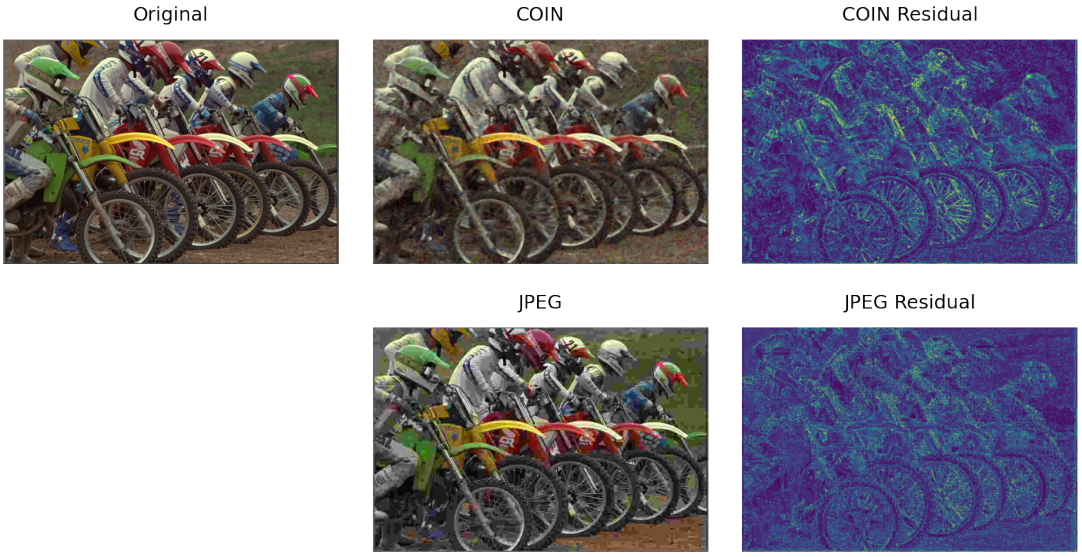

Figure 3.9: Comparison between COIN and JPEG on image 5 at 0.3bpp. The PSNRs are 20.97dB and 21.34dB respectively.

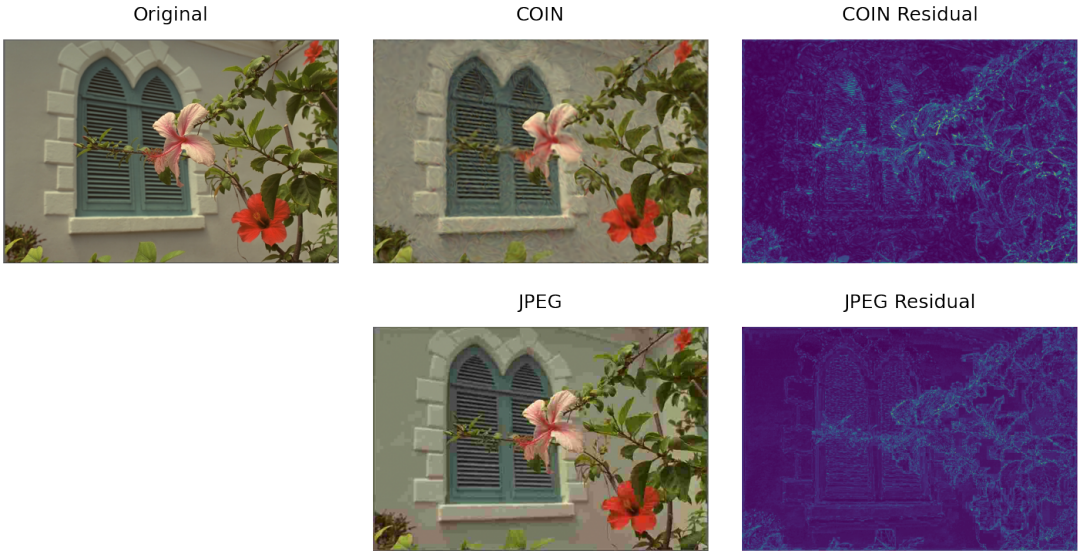

Figure 3.10: Comparison between COIN and JPEG on image 7 at 0.3bpp. The PSNRs are 26.92dB and 27.56dB respectively.

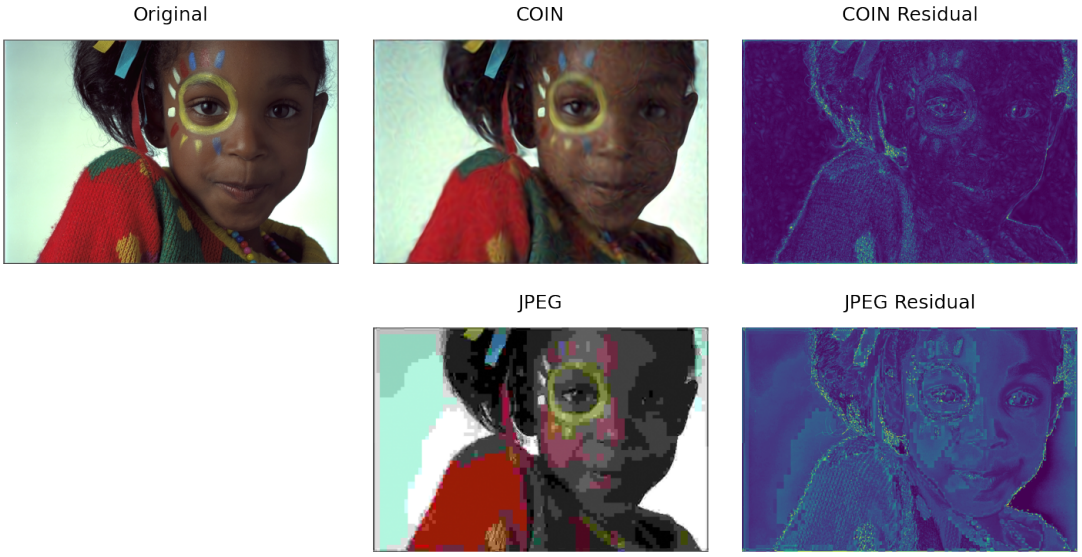

Figure 3.11: Comparison between COIN and JPEG on image 15 at 0.15bpp. The PSNRs are 27.35dB and 21.74dB respectively.

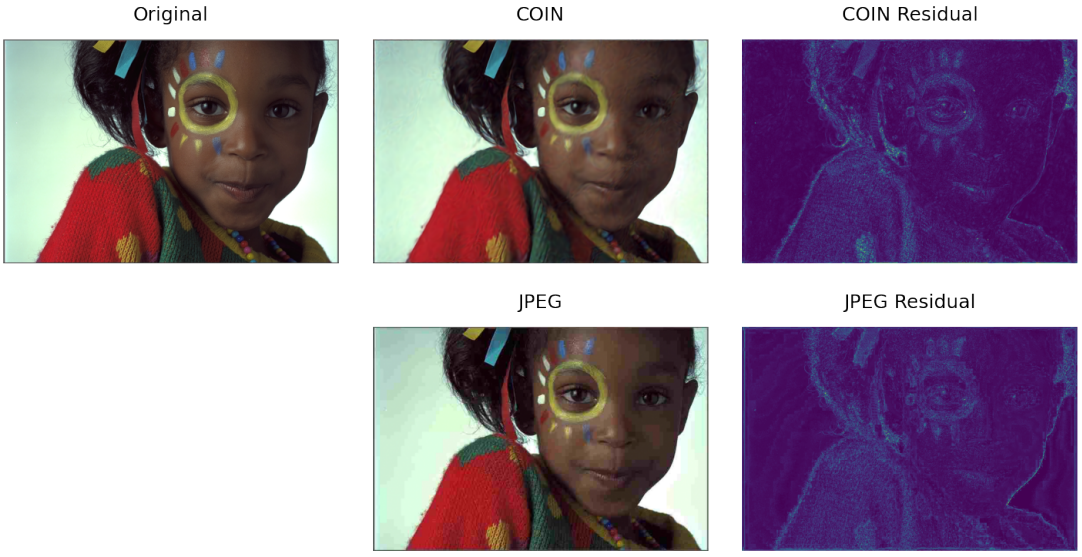

Figure 3.12: Comparison between COIN and JPEG on image 15 at 0.3bpp. The PSNRs are 29.31dB and 28.85dB respectively.

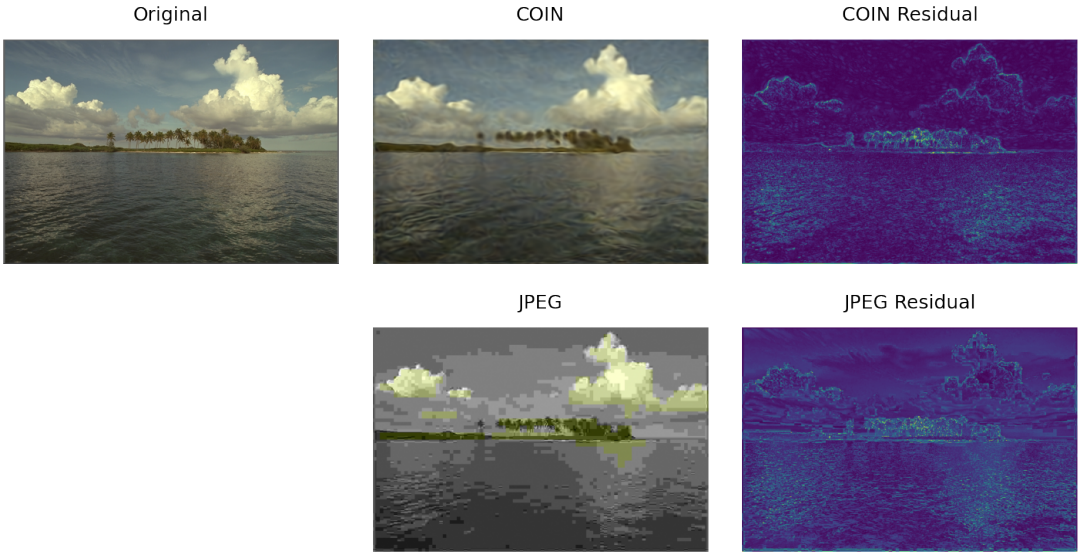

Figure 3.13: Comparison between COIN and JPEG on image 16 at 0.15bpp. The PSNRs are 27.19dB and 24.16dB respectively.

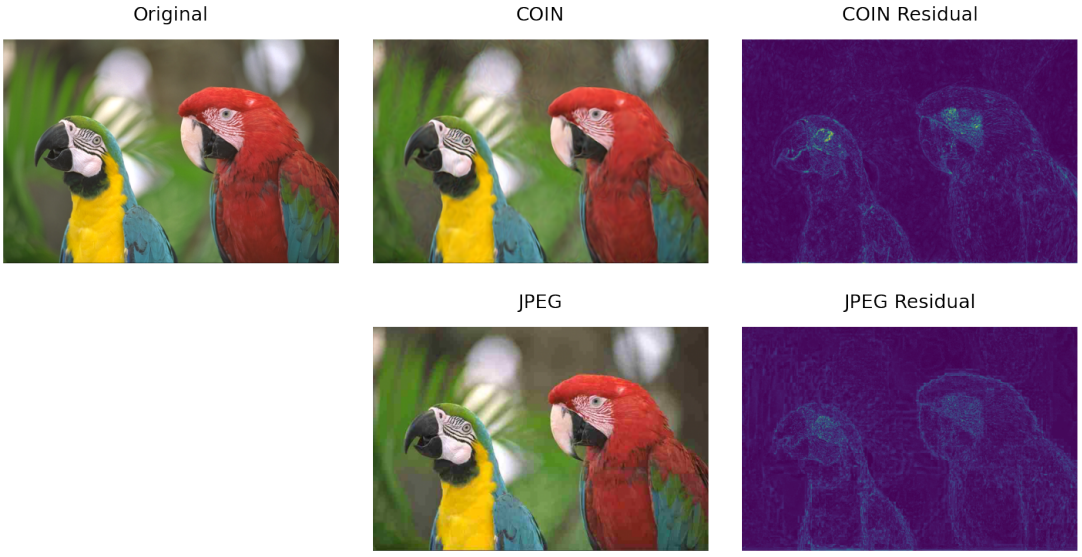

Figure 3.14: Comparison between COIN and JPEG on image 23 at 0.3bpp. The PSNRs are 31.08dB and 30.92dB respectively.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	COIN: COmpression with Implicit Neural representations
Publication Status	X Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	*Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, Arnaud Doucet*, COIN: COmpression with Implicit Neural representations, ICLR 2021 Workshop on Neural Compression

Student Confirmation

Student Name:	Emilien Dupont		
Contribution to the Paper	I had the idea for the project which was refined in discussions with Adam. I wrote the initial codebase and ran experiments, with Adam also helping out with the code and experiments. I wrote the paper and Adam, Milad, Yee Whye and Arnaud provided feedback and edits for the manuscript.		
Signature 		Date	September 19, 2022

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: *Professor Yee Whye Teh*		
Supervisor comments		
Signature 	Date	23 Sep 2022

4

COIN++: Neural compression across modalities

Abstract

Neural compression algorithms are typically based on autoencoders that require specialized encoder and decoder architectures for different data modalities. In this paper, we propose COIN++, a neural compression framework that seamlessly handles a wide range of data modalities. Our approach is based on converting data to implicit neural representations, i.e. neural functions that map coordinates (such as pixel locations) to features (such as RGB values). Then, instead of storing the weights of the implicit neural representation directly, we store modulations applied to a meta-learned base network as a compressed code for the data. We further quantize and entropy code these modulations, leading to large compression gains while reducing encoding time by two orders of magnitude compared to baselines. We empirically demonstrate the effectiveness of our method by compressing various data modalities, from images and audio to medical and climate data.

4.1 Introduction

It is estimated that several exabytes of data are created everyday [Domo, 2018]. This data is comprised of a wide variety of data modalities, each of which could benefit from compression. However, the vast majority of work in neural compression has focused only on image and video data [Ma et al., 2019]. In this paper, we introduce a new approach for neural compression, called COIN++, which is applicable to a wide range of data modalities, from images and audio to medical and climate data (see Figure 4.1).

Most neural compression algorithms are based on autoencoders [Ballé et al., 2018, Minnen et al., 2018, Lee et al., 2019a]. An encoder maps an image to a latent representation which is quantized and entropy coded into a bitstream. The bitstream is then transmitted to a decoder that reconstructs the image. The parameters of the encoder and decoder are trained to jointly minimize reconstruction error, or *distortion*, and the length of the compressed code, or *rate*. To achieve good performance, these algorithms heavily rely on encoder and decoder architectures that are specialized to images [Cheng et al., 2020b, Xie et al., 2021]. Applying these models to new data modalities then requires designing new encoders and decoders which is usually challenging.

Recently, a new framework for neural compression, called COIN (COmpression with Implicit Neural representations), was proposed which bypasses the need for specialized encoders and decoders [Dupont et al., 2021]. Instead of compressing images directly, COIN fits a neural network mapping pixel locations to RGB values to an image and stores the quantized weights of this network as a compressed

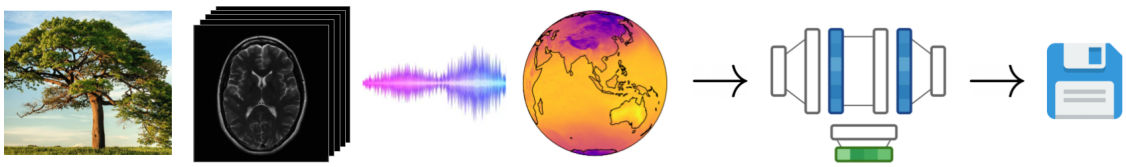

Figure 4.1: COIN++ converts a wide range of data modalities to neural networks via optimization and then stores the parameters of these neural networks as compressed codes for the data.

code for the image. While Dupont et al. [2021] only apply COIN to images, it holds promise for storing other data modalities. Indeed, neural networks mapping coordinates (such as pixel locations) to features (such as RGB values), typically called *implicit neural representations* (INR), have been used to represent signed distance functions [Park et al., 2019], voxel grids [Mescheder et al., 2019], 3D scenes [Sitzmann et al., 2019b, Mildenhall et al., 2020], temperature fields [Dupont et al., 2022c], videos [Li et al., 2021b], audio [Sitzmann et al., 2020b] and many more. COIN-like approaches that convert data to INRs and compress these are therefore promising for building flexible neural codecs applicable to a range of modalities.

In this paper, we identify and address several key problems with COIN and propose a compression algorithm applicable to multiple modalities, which we call COIN++. More specifically, we identify the following issues with COIN: *1.* Encoding is slow: compressing a single image can take up to an hour, *2.* Lack of shared structure: as each image is compressed independently, there is no shared information between networks, *3.* Performance is well below state of the art (SOTA) image codecs. We address these issues by: *1.* Using meta-learning to reduce encoding time by more than two orders of magnitude to less than a second, compared to minutes or hours for COIN, *2.* Learning a base network that encodes shared structure and applying modulations to this network to encode instance specific information, *3.* Quantizing and entropy coding the modulations. While our method significantly exceeds COIN both in terms of compression and speed, it only partially closes the gap to SOTA codecs on well-studied modalities such as images. However, COIN++ is applicable to a wide range of data modalities where traditional methods cannot be used, making it a promising tool for neural compression in non-standard domains.

4.2 Method

In this paper, we consider compressing data that can be expressed in terms of sets of coordinates $\mathbf{x} \in \mathcal{X}$ and features $\mathbf{y} \in \mathcal{Y}$. An image for example can be described by a set of pixel locations $\mathbf{x} = (x, y)$ in $\mathbb{R}^2$ and their corresponding RGB values

$\mathbf{y} = (r, g, b)$ in $\{0, 1, ..., 255\}^3$. Similarly, an MRI scan can be described by a set of positions in 3D space $\mathbf{x} = (x, y, z)$ and an intensity value $\mathbf{y} \in \mathbb{R}^+$. Given a single datapoint as a collection of coordinate and feature pairs $\mathbf{d} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ (for example an image as a collection of n pixel locations and RGB values), the COIN approach consists in fitting a neural network $f_\theta : \mathcal{X} \to \mathcal{Y}$ with parameters θ to the datapoint by minimizing

$$\mathcal{L}(\theta, \mathbf{d}) = \sum_{i=1}^{n} \|f_\theta(\mathbf{x}_i) - \mathbf{y}_i\|_2. \tag{4.1}$$

The weights θ are then quantized and stored as a compressed representation of the datapoint $\mathbf{d}$. The neural network f_θ is parameterized by a SIREN [Sitzmann et al., 2020b], i.e. an MLP with sine activation functions, which is necessary to fit high frequency data such as natural images [Mildenhall et al., 2020, Tancik et al., 2020, Sitzmann et al., 2020b]. More specifically, a SIREN layer is defined by an elementwise sin applied to a hidden feature vector $\mathbf{h} \in \mathbb{R}^d$ as

$$\text{SIREN}(\mathbf{h}) = \sin(\omega_0(W\mathbf{h} + \mathbf{b})) \tag{4.2}$$

where $W \in \mathbb{R}^{d \times d}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^d$ a bias vector and $\omega_0 \in \mathbb{R}^+$ a positive scaling factor.

While this approach is very general, there are several key issues. Firstly, as compression involves minimizing equation 4.1, encoding is extremely slow. For example, compressing a single image from the Kodak dataset [Kodak, 1991] takes nearly an hour on a 1080Ti GPU [Dupont et al., 2021]. Secondly, as each datapoint $\mathbf{d}$ is fitted with a separate neural network f_θ, there is no information shared across datapoints. This is clearly suboptimal: natural images for example share a lot of common structure that does not need to be repeatedly stored for each individual image. In the following sections, we show how our proposed approach, COIN++, addresses these problems while maintaining the generality of COIN.

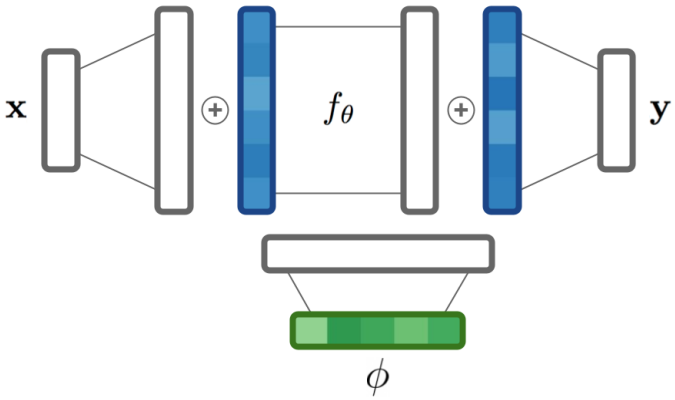

Figure 4.2: COIN++ architecture. Latent modulations ϕ (in green) are mapped to modulations (in blue) which are added to activations of the base network f_θ (in white) to parameterize a single function that is evaluated at coordinates $\mathbf{x}$ to obtain features $\mathbf{y}$.

4.2.1 Storing modulations

While COIN stores each image as a separate neural network, we instead train a base network shared across datapoints and apply *modulations* to this network to parameterize individual datapoints. Given a base network, such as a multi-layer perceptron (MLP), we use FiLM layers [Perez et al., 2018], to modulate the hidden features $\mathbf{h} \in \mathbb{R}^d$ of the network by applying elementwise scales $\boldsymbol{\gamma} \in \mathbb{R}^d$ and shifts $\boldsymbol{\beta} \in \mathbb{R}^d$ as

$$\text{FiLM}(\mathbf{h}) = \boldsymbol{\gamma} \odot \mathbf{h} + \boldsymbol{\beta}. \tag{4.3}$$

Given a fixed base MLP, we can therefore parameterize families of neural networks by applying different scales and shifts at each layer. Each neural network function is therefore specified by a set of scales and shifts, which are collectively referred to as modulations [Perez et al., 2018]. Recently, the FiLM approach has also been applied in the context of INRs. Chan et al. [2021c] parameterize the generator in a generative adversarial network by a SIREN network and generate samples by applying modulations to this network as $\sin(\boldsymbol{\gamma} \odot (W\mathbf{h} + \mathbf{b}) + \boldsymbol{\beta})$. Similarly, Mehta et al. [2021] parameterize families of INRs using a scale factor via $\boldsymbol{\alpha} \odot \sin(W\mathbf{h} + \mathbf{b})$. Both of these approaches can be modified to use a low dimensional latent vector mapped to a set of modulations instead of directly applying modulations. Chan

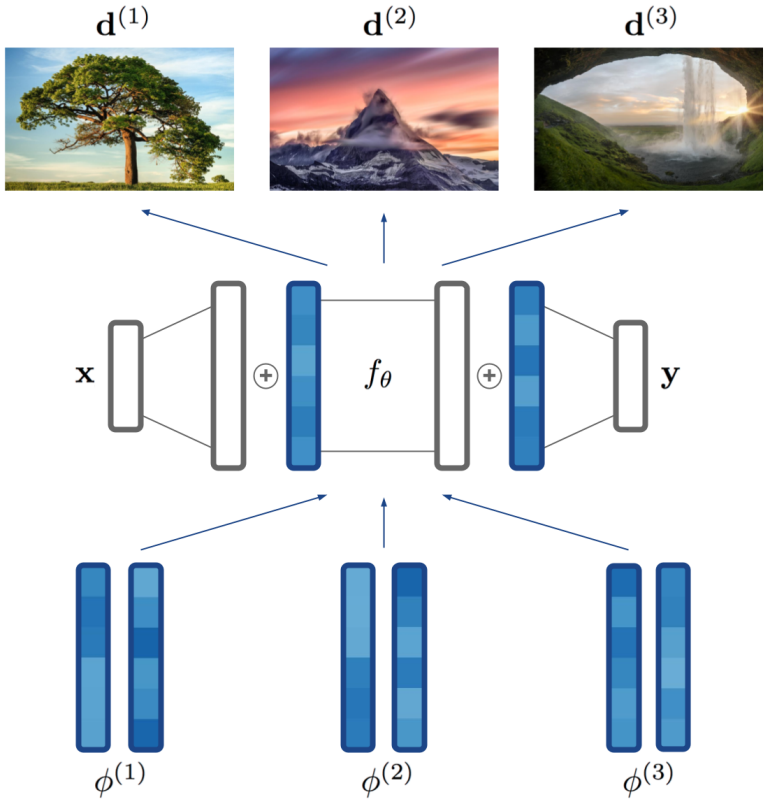

Figure 4.3: By applying modulations $\phi^{(1)}$, $\phi^{(2)}$, $\phi^{(3)}$ to a base network f_θ, we obtain different functions that can be decoded into datapoints $\mathbf{d}^{(1)}$, $\mathbf{d}^{(2)}$, $\mathbf{d}^{(3)}$ by evaluating the functions at various coordinates. While we show images in this figure, the same principle can be applied to a range of data modalities.

et al. [2021c] map a latent vector to scales and shifts with an MLP, while Mehta et al. [2021] map the latent vector through an MLP of the same shape as the base network and use the hidden activations of this network as modulations. However, we found that both of these approaches performed poorly in terms of compressibility, requiring a large number of modulations to achieve satisfying reconstructions.

Instead, we propose a new parameterization of modulations for INRs which, on top of yielding better compression rates, is also more stable to train. More specifically, given a base SIREN network, we only apply shifts $\boldsymbol{\beta} \in \mathbb{R}^d$ as modulations using

$$\sin(\omega_0(W\mathbf{h} + \mathbf{b} + \boldsymbol{\beta})) \tag{4.4}$$

at every layer of the MLP. To further reduce storage, we use a latent vector which is linearly mapped to the modulations as shown in Figure 4.2. In a slight overload

of notation, we also refer to this vector as modulations or latent modulations. Indeed, we found empirically that using only shifts gave the same performance as using both shifts and scales while using only scales yielded considerably worse performance. In addition, linearly mapping the latent vector to modulations worked better than using a deep MLP as in [Chan et al., 2021c]. Given this parameterization, we then store a datapoint d (such as an image) as a set of (latent) modulations ϕ. To decode the datapoint, we simply evaluate the modulated base network $f_\theta(\cdot;\phi)$ at every coordinate $\mathbf{x}$,

$$\mathbf{y} = f_\theta(\mathbf{x};\phi) \tag{4.5}$$

as shown in Figure 4.3. To fit a set of modulations ϕ to a datapoint $\mathbf{d}$, we keep the parameters θ of the base network fixed and minimize

$$\mathcal{L}(\theta,\phi,\mathbf{d}) = \sum_{i=1}^{n} \|f_\theta(\mathbf{x}_i;\phi) - \mathbf{y}_i\|_2 \tag{4.6}$$

over ϕ. In contrast to COIN, where each datapoint $\mathbf{d}$ is stored as a separate neural network f_θ, COIN++ only requires storing $O(n)$ modulations (or less when using latents) as opposed to $O(n^2)$ weights, where n is the width of the MLP. In addition, this approach allows us to store shared information in the base network and instance specific information in the modulations. For natural images for example, the base network encodes structure that is common to natural images while the modulations store the information required to reconstruct individual images.

4.2.2 Meta-learning modulations

Given a base network f_θ, we can encode a datapoint $\mathbf{d}$ by minimizing equation 4.6. However, we are still faced with two problems: *1.* We need to learn the weights θ of the base network, *2.* Encoding a datapoint via equation 4.6 is slow, requiring thousands of iterations of gradient descent. COIN++ solves both of these problems with meta-learning.

Recently, Sitzmann et al. [2020a], Tancik et al. [2021] have shown that applying MAML [Finn et al., 2017] to INRs can reduce fitting at test time to just a few gradient steps. Instead of minimizing $\mathcal{L}(\theta,\mathbf{d})$ directly via gradient descent from a

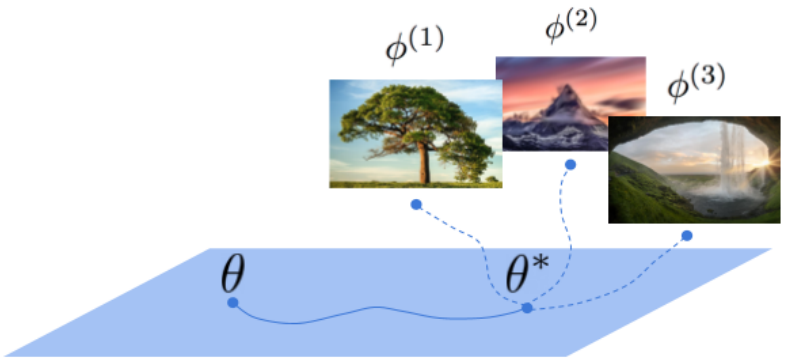

Figure 4.4: We meta-learn parameters θ^* of the base network such that modulations ϕ can easily be fit in a few gradient steps.

random initialization, we can meta-learn an initialization θ^* such that minimizing $\mathcal{L}(\theta, \mathbf{d})$ can be done in a few gradient steps. More specifically, assume we are given a dataset of N points $\{\mathbf{d}^{(j)}\}_{j=1}^N$. Starting from an initialization θ, a step of the MAML inner loop on a datapoint $\mathbf{d}^{(j)}$ is given by

$$\theta^{(j)} = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathbf{d}^{(j)}), \tag{4.7}$$

where α is the inner loop learning rate. We are then interested in learning a good initialization θ^* such that the loss $\mathcal{L}(\theta, \mathbf{d}^{(j)})$ is minimized after a few gradient steps across the entire set of datapoints $\{\mathbf{d}^{(j)}\}_{j=1}^N$. To update the initalization θ, we then perform a step of the outer loop, with an outer loop learning rate β, via

$$\theta \leftarrow \theta - \beta \nabla_\theta \textstyle\sum_{j=1}^N \mathcal{L}(\theta^{(j)}, \mathbf{d}^{(j)}). \tag{4.8}$$

In our case, MAML cannot be used directly since at test time we only fit the modulations ϕ and not the shared parameters θ. We therefore need to meta-learn an initialization for θ and ϕ such that, given a new datapoint, the *modulations* ϕ can rapidly be computed while keeping θ constant. Indeed, we only store the modulations for each datapoint and share the parameters θ across all datapoints. For COIN++, a single step of the inner loop is then given by

$$\phi^{(j)} = \phi - \alpha \nabla_\phi \mathcal{L}(\theta, \phi, \mathbf{d}^{(j)}), \tag{4.9}$$

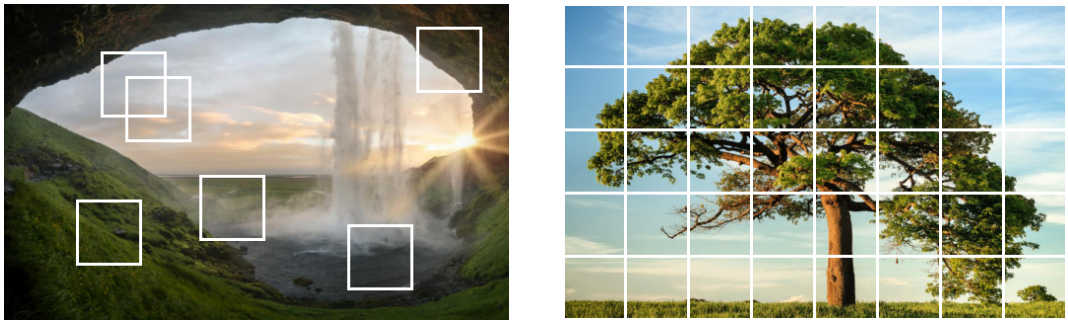

Figure 4.5: During training we sample patches randomly, while at test time we partition the datapoint into patches and fit modulations to each patch.

where θ is kept fixed. Performing the inner loop on a subset of parameters has previously been explored by Zintgraf et al. [2019] and is referred to as CAVIA. As observed in CAVIA, meta-learning the initialization for ϕ is redundant as it can be absorbed into a bias parameter of the base network weights θ. We therefore only need to meta-learn the shared parameter initialization θ. The update rule for the outer loop is then given by

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{j=1}^{N} \mathcal{L}(\theta, \phi^{(j)}, \mathbf{d}^{(j)}). \tag{4.10}$$

The inner loop then updates the modulations ϕ while the outer loop updates the shared parameters θ. This algorithm allows us to meta-learn a base network such that each set of modulations can easily and rapidly be fitted (see Figure 4.4). In practice, we find that as few as 3 gradient steps gives us compelling results, compared with thousands for COIN.

4.2.3 Patches, quantization and entropy coding for modulations

Patches for large scale data. While meta-learning the base network allows us to rapidly encode new datapoints into modulations, the training procedure is expensive, as MAML must take gradients through the inner loop [Finn et al., 2017]. For large datapoints (such as high resolution images or MRI scans), this can become prohibitively expensive. While first-order approximations exist [Finn et al., 2017, Nichol et al., 2018, Rajeswaran et al., 2019], we found that they severely

hindered performance. Instead, to reduce memory usage, we split datapoints into random patches during training. For large scale images for example, we train on 32×32 patches. At train time, we then learn a base network such that modulations can easily be fit to patches. At test time, we split a new image into patches and compute modulations for each of them. The image is then represented by the set of modulations for all patches (see Figure 4.5). We use a similar approach for other data modalities, e.g. MRI scans are split into 3D patches.

Quantization. While COIN quantizes the neural network weights from 32 bits to 16 bits to reduce storage, quantizing beyond this severely hinders performance [Dupont et al., 2021]. In contrast, we find that modulations are surprisingly quantizable. During meta-learning, modulations are represented by 32 bit floats. To quantize these to shorter bitwidths, we simply use uniform quantization. We first clip the modulations to lie within 3 standard deviations of their mean. We then split this interval into 2^b equally sized bins (where b is the number of bits). Remarkably, we found that reducing the number of bits from 32 to 5 (i.e. reducing the number of symbols from more than 10^9 to only 32) resulted only in small decreases in reconstruction accuracy. Simply applying uniform quantization then improves compression by a factor of 6 at little cost in reconstruction quality.

Entropy coding. A core component of almost all codecs is entropy coding, which allows for lossless compression of the quantized code, using e.g. arithmetic coding [Rissanen and Langdon, 1979]. This relies on a model of the distribution of the quantized codes. As with quantization, we use a very simple approach for modeling this distribution: we count the frequency of each quantized modulation value and use this distribution for arithmetic coding. In our experiments, this reduced storage 8-15% at no cost in reconstruction quality. While this simple entropy coding scheme works well, we expect more sophisticated methods to significantly improve performance, which is an exciting direction for future work.

4.3 Related Work

Neural compression. Learned compression approaches are typically based on autoencoders that jointly minimize rate and distortion, as initially introduced in Ballé et al. [2017]. Ballé et al. [2018] extend this by adding a hyperprior, while Mentzer et al. [2018], Minnen et al. [2018], Lee et al. [2019a] use an autoregressive model to improve entropy coding. Cheng et al. [2020b] improve the accuracy of the entropy models by adding attention and Gaussian mixture models for the distribution of latent codes, while Xie et al. [2021] use invertible convolutional layers to further enhance performance. While most of these are optimized on traditional distortion metrics such as MSE or SSIM, other works have explored the use of generative adversarial networks for optimizing perceptual metrics [Agustsson et al., 2019, Mentzer et al., 2020]. Neural compression has also been applied to video [Lu et al., 2019, Goliński et al., 2020, Agustsson et al., 2020] and audio [Kleijn et al., 2018, Valin and Skoglund, 2019, Yang et al., 2019, Zeghidour et al., 2021].

Implicit neural representations and compression. In addition to COIN, several recent works have explored the use of INRs for compression. Davies et al. [2020] encode 3D shapes with neural networks and show that this can reduce memory usage compared with traditional decimated meshes. Chen et al. [2021] represent videos by convolutional neural networks that take as input a time index and output a frame in the video. By pruning, quantizing and entropy coding the weights of this network, the authors achieve compression performance close to standard video codecs. Lee et al. [2021] meta-learn sparse and parameter efficient initializations for INRs and show that this can reduce the number of parameters required to store an image at a given reconstruction quality, although it is not yet competitive with image codecs such as JPEG. Lu et al. [2021b], Isik et al. [2021] explore the use of INRs for volumetric compression. Two concurrent works also use function representations for image [Strümpler et al., 2021] and video [Zhang et al., 2021] compression. Strümpler et al. [2021] meta-learn an MLP initialization and subsequently quantize and entropy code the weights of MLPs fitted to images, leading to large performance gains over COIN. However, their approach still requires

tens of thousands of iterations at test time to fully converge while underperforming image codecs like JPEG2000. Zhang et al. [2021] compress frames in videos using INRs (which are quantized and entropy coded) while learning a flow warping to model differences between frames. Results on video benchmarks are promising although the performance still lags behind standard video codecs. To the best of our knowledge, none of these works have considered INRs for building a unified compression framework across data modalities.

4.4 Experiments

We evaluate COIN++ on four data modalities: images, audio, medical data and climate data. We implement all models in PyTorch [Paszke et al., 2019] and train on a single GPU. We use SGD for the inner loop with a learning rate of 1e-2 and Adam for the outer loop with a learning rate of 1e-6 or 3e-6. We normalize coordinates $\mathbf{x}$ to lie in $[-1, 1]$ and features $\mathbf{y}$ to lie in $[0, 1]$. Full experimental details required to reproduce all the results can be found in the appendix. We train COIN++ using MSE between the compressed and ground truth data. As is standard, we measure reconstruction performance (or distortion) using PSNR (in dB), which is defined as $\text{PSNR} = -10 \log_{10}(\text{MSE})$. We measure the size of the compressed data (or rate) in terms of bits-per-pixel (bpp) which is given by $\frac{\text{number of bits}}{\text{number of pixels}}$[1] and kilobits per second (kpbs) for audio. We benchmark COIN++ against a large number of baselines including standard image codecs - JPEG [Wallace, 1992], JPEG2000 [Skodras et al., 2001], BPG [Bellard, 2014] and VTM [Bross et al., 2021] - autoencoder based neural compression - BMS [Ballé et al., 2018], MBT [Minnen et al., 2018] and CST [Cheng et al., 2020b] - standard audio codecs - MP3 [MP3, 1993] - and COIN [Dupont et al., 2021]. For clarity, we use consistent colors for different codecs and plot learned codecs with solid lines and standard codecs with dashed lines. The code to reproduce all experiments in the paper can be found at `https://github.com/EmilienDupont/coinpp`.

[1] For non image data a "pixel" corresponds to a single dimension of the data.

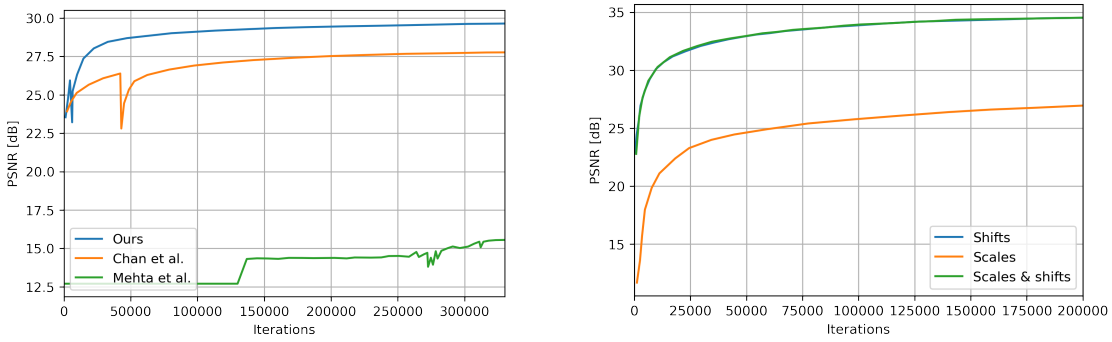

Figure 4.6: (Left) PSNR on CIFAR10 using a fixed number of modulations (see appendix for experimental details). (Right) Comparison of using shifts, scales and scales & shifts for modulations (note that shifts and scales & shifts overlap).

4.4.1 Comparisons to other INR parameterizations

We first compare our parameterization of INRs with the methods proposed by Chan et al. [2021c] and Mehta et al. [2021] as described in Section 4.2.1. As can be seen in Figure 4.6, our method significantly outperforms both in terms of compressibility, improving PSNR by 2dB with the same number of parameters[2]. Further, using shift modulations is more effective than using scales and shifts, and performs significantly better than scales alone. We also note that our method allows us to quickly fit INRs using only a few hundred parameters. This is in contrast to existing works on meta-learning for INRs [Sitzmann et al., 2020a, Tancik et al., 2021], which typically require fitting 3 orders of magnitude more parameters at test time. While we focus on using our parameterization for compression in this paper, we also believe it may be useful more generally for learning families of INRs.

4.4.2 Images: CIFAR10

We train COIN++ on CIFAR10 using 128, 256, 384, 512, 768 and 1024 latent modulations. As can be seen in Figure 4.7, COIN++ vastly outperforms COIN, JPEG and JPEG2000 while partially closing the gap to BPG, particularly at low bitrates. To the best of our knowledge, this is the first time compression with INRs has outperformed image codecs like JPEG2000. The remaining gap between

[2]Despite significant experimental effort, we were unable to achieve better performance using meta-learning with [Mehta et al., 2021].

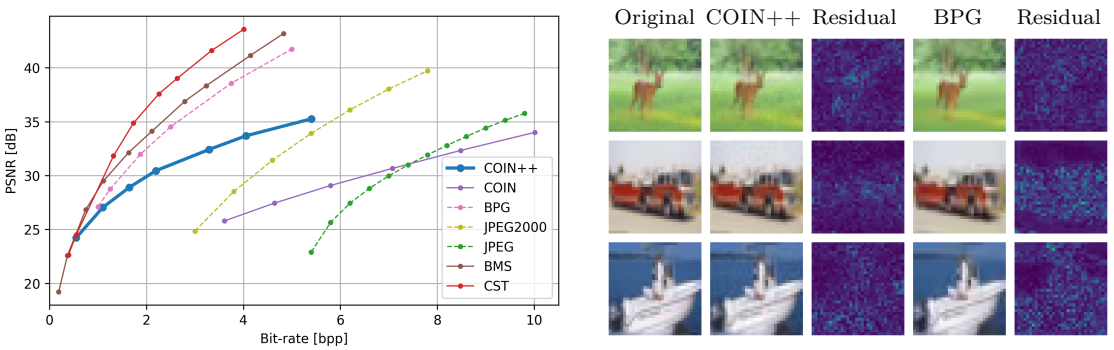

Figure 4.7: (Left) Rate distortion plot on CIFAR10. (Right) Qualitative comparison of compression artifacts for models at similar reconstruction quality. COIN++ achieves 32.4dB at 3.29 bpp while BPG achieves 31.9dB at 1.88 bpp.

COIN++ and SOTA codecs (BMS, CST) is likely due to entropy coding: we use the simple scheme described in Section 4.2.3, while BMS and CST use deep generative models. We hypothesize that using deep entropy coding for the modulations would significantly reduce or close this gap. Figure 4.7 shows qualitative comparisons between our model and BPG to highlight the types of compression artifacts obtained with COIN++. In order to thoroughly analyse and evaluate each component of COIN++, we perform a number of ablation studies.

Quantization bitwidth. Quantizing the modulations to a lower bitwidth yields more compressed codes at the cost of reconstruction accuracy. To understand the tradeoff between these, we show rate distortion plots when quantizing from 3 to 8 bits in Figure 4.8a. As can be seen, the optimal bitwidths are surprisingly low: 5 bits is optimal at low bitrates while 6 is optimal at higher bitrates. Qualitative artifacts obtained from quantizing the modulations are shown in Figure 4.15 in the appendix.

Quantization COIN vs COIN++. We compare the drop in PSNR due to quantization for COIN and COIN++ in Figure 4.8b. As can be seen, modulations are remarkably quantizable: when quantizing the COIN weights directly, performance decreases significantly around 14 bits, whereas quantizing modulations yields small drops in PSNR even when using 5 bits. However, as shown in Figure 4.8c, the drop in PSNR from quantization is larger for larger models.

Entropy coding. Figure 4.14 in the appendix shows rate distortion plots for full precision, quantized and entropy coded modulations. As can be seen, both

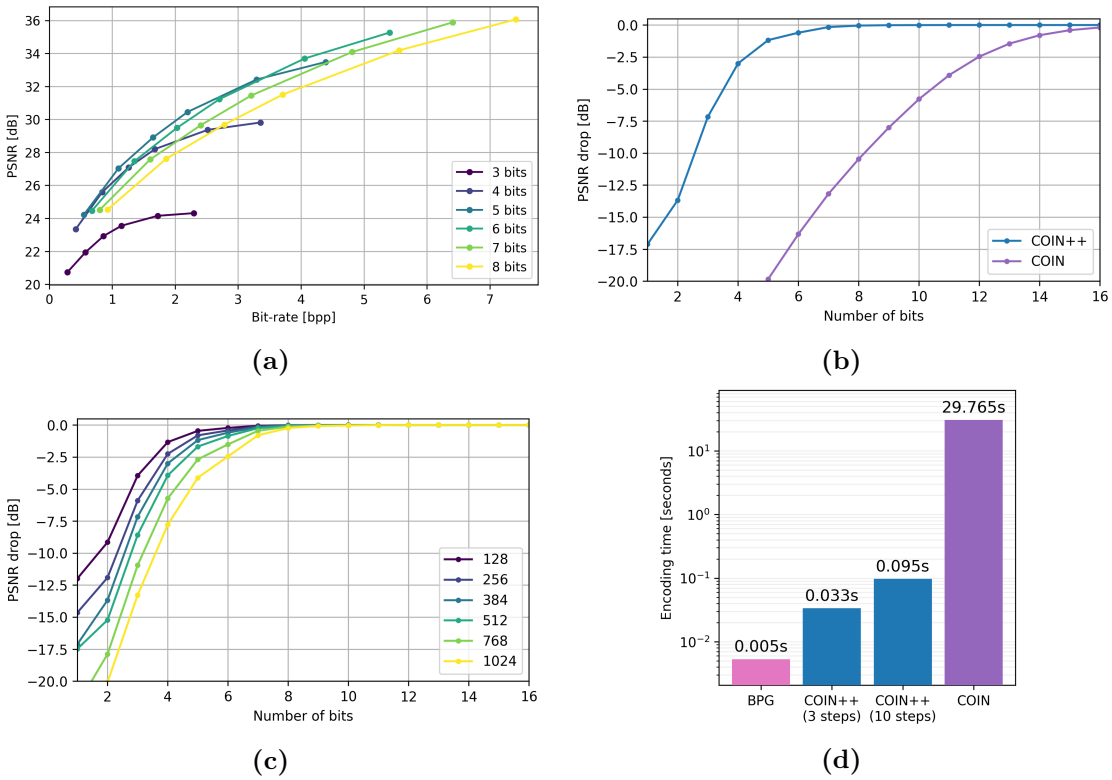

Figure 4.8: (a) Rate distortion plot on CIFAR10 when quantizing the modulations ϕ to various bitwidths. (b) Drop in PSNR for COIN and COIN++ quantization. (c) Drop in PSNR when quantizing the modulations ϕ to various bitwidths, for various latent dimensions. (d) Encoding time per image on CIFAR10 (log scale).

quantization and entropy coding significantly improve performance.

Encoding time. Figure 4.8d shows the average encoding time for COIN++, COIN and BPG on CIFAR10 (see appendix 4.B.1 for hardware details). As can be seen, COIN++ compresses images 300× faster than COIN while achieving a 4× better compression rate. Note that these results are obtained from compressing each image separately. When using batches of images, we can compress the entire CIFAR10 test set (10k images) in 4mins when using 10 inner loop steps (and in just over a minute when using 3 steps). In addition, as shown in Figure 4.16 in the appendix, COIN++ requires only 3 gradient steps to reach the same performance as COIN does in 10,000 steps, while using 4× less storage.

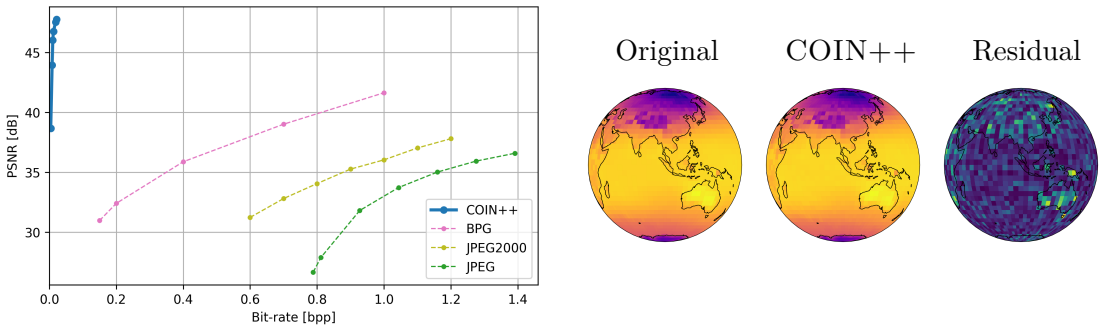

Figure 4.9: (Left) Rate distortion plot on ERA5. (Right) COIN++ compression artifacts on ERA5. See appendix 4.E.5 for more samples.

4.4.3 Climate data: ERA5 global temperature measurements

To demonstrate the flexibility of our approach, we also use COIN++ to compress data lying on a manifold. We use global temperature measurements from the ERA5 dataset [Hersbach et al., 2019] with the processing and splits from Dupont et al. [2022c]. The dataset contains 8510 train and 2420 test globes of size 46×90, with temperature measurements at equally spaced latitudes λ and longitudes φ on the Earth from 1979 to 2020. To model this data, we follow Dupont et al. [2022c] and use spherical coordinates $\mathbf{x} = (\cos\lambda\cos\varphi, \cos\lambda\sin\varphi, \sin\lambda)$ for the inputs. As a baseline, we compare COIN++ against JPEG, JPEG2000 and BPG applied to flat map projections of the data. As can be seen in Figure 4.9, COIN++ vastly outperforms all baselines. These strong results highlight the versatility of the COIN++ approach: unlike traditional codecs and autoencoder based methods (which would require spherical convolutions for the encoder), we can easily apply our method to a wide range of data modalities, including data lying on a manifold. Indeed, COIN++ achieves a 3000× compression rate while having an RMSE of 0.5°C, highlighting the potential for compressing climate data.

4.4.4 Compression with patches

To evaluate the patching approach from Section 4.2.3 and to demonstrate that COIN++ can scale to large data (albeit at a cost in performance), we test our model on images, audio and MRI data.

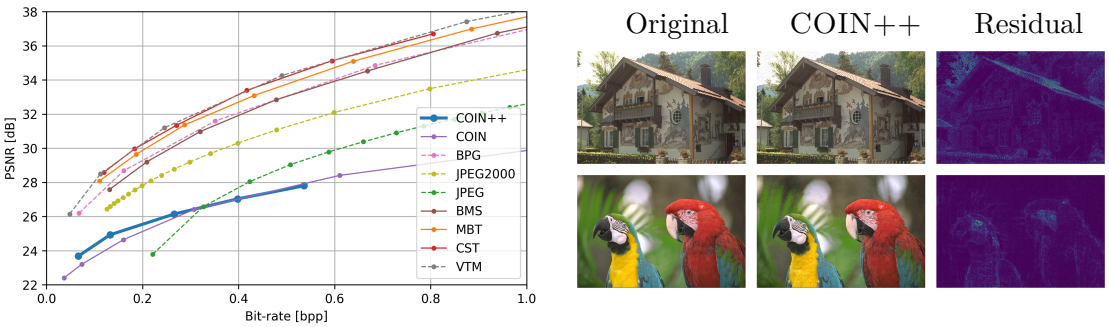

Figure 4.10: (Left) Rate distortion plot on Kodak. (Right) COIN++ compression artifacts on Kodak. See appendix 4.E.5 for more samples.

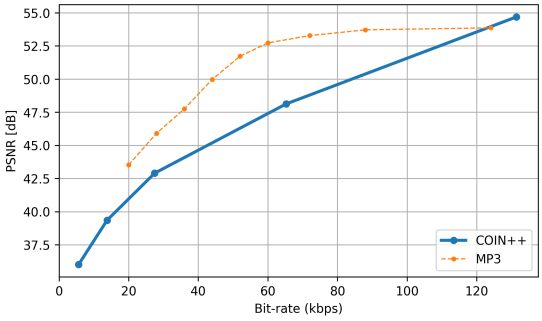

Figure 4.11: Rate distortion plot on LibriSpeech.

Large scale images: Kodak. The Kodak dataset [Kodak, 1991] contains 24 large scale images of size 768×512. To train the model, we use random 32×32 patches from the Vimeo90k dataset [Xue et al., 2019], containing 154k images of size 448×256. At evaluation time, each Kodak image is then split into 384 32×32 patches which are compressed independently. As we do not model the global structure of the image, we therefore expect a significant drop in performance compared to the case when no patching is required. As can be seen in Figure 4.10, the performance of COIN++ indeed drops, but still outperforms COIN and JPEG at low bitrates. We expect that this can be massively improved by modeling the global structure of the image (e.g. two patches of blue sky are nearly identical, but that information redundancy is not exploited in the current setup) but leave this to future work.

Audio: LibriSpeech. To evaluate COIN++ on audio, we use the LibriSpeech dataset [Panayotov et al., 2015] containing several hours of speech data recorded at 16kHz. As a baseline, we compare against the widely used MP3 codec [MP3,

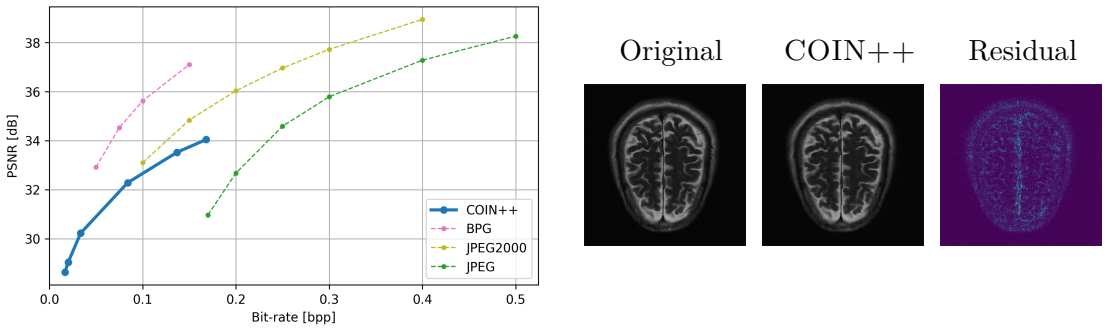

Figure 4.12: (Left) Rate distortion plot on FastMRI. (Right) COIN++ compression artifacts on FastMRI. See appendix 4.E.5 for more samples.

1993]. We split each audio sample into patches of varying size and compress each of these to obtain models at various bit-rates (we refer to appendix 4.B.5 for full experimental details). As can be seen in Figure 4.11, even though audio is a very different modality from the rest considered in this paper, COIN++ can still be used for compression, highlighting the versatility of our approach. However, in terms of performance, COIN++ still lags behind well-establised audio codecs such as MP3.

Medical data: brain MRI scans. Finally, we train our model on brain MRI scans from the FastMRI dataset [Zbontar et al., 2018]. The dataset contains 565 train volumes and 212 test volumes with sizes ranging from 16×320×240 to 16×384×384 (see appendix 4.A.2 for full dataset details). As a baseline, we compare our model against JPEG, JPEG2000 and BPG applied independently to each slice. Due to memory constraints, we train COIN++ on 16×16×16 patches. We therefore store roughly 400 independent patches at test time (as opposed to 16 slices for the image codecs). Even then COIN++ performs reasonably well, particularly at low bitrates (see Figure 4.12). As a large number of patches are nearly identical, especially close to the edges, we expect that large gains can be made from modeling the global structure of the data. Qualitatively, our model also performs well although it has patch artifacts at low bitrates (see Figure 4.12).

4.5 Conclusion, limitations and future work

Conclusion. We introduce COIN++, the first (to the best of our knowledge) neural codec applicable to multiple modalities. Our framework significantly improves performance compared to COIN both in terms of compression and encoding time, while being competitive with well-established codecs such as JPEG. While COIN++ does not match the performance of SOTA codecs, we hope our work will help expand the range of domains where neural compression is applicable.

Limitations. The main drawback of COIN++ is that, because of the second-order gradients required for MAML, training the model is memory intensive. This in turn limits scalability and requires us to use patches for large data. Devising effective first-order approximations or bypassing meta-learning altogether would mitigate these issues. In addition, training COIN++ can occasionally be unstable, although the model typically recovers from loss instabilities (see Figure 4.13 in the appendix). Further, there are several common modalities our framework cannot handle, such as text or tabular data, as these are not easily expressible as continuous functions. Finally, COIN++ still lags behind SOTA codecs. However, we believe there are several interesting directions for future work to close this gap.

Future work. In its current form, COIN++ employs very basic methods for both quantization and entropy coding - using more sophisticated techniques for these two steps could likely lead to large performance gains. Indeed, recent success in modeling distributions of functions [Schwarz et al., 2020, Anokhin et al., 2021, Skorokhodov et al., 2021, Dupont et al., 2022c] suggests that large gains could be made from using deep generative models to learn the distribution of modulations for entropy coding. Similarly, better post-training quantization [Nagel et al., 2019, Li et al., 2021a] or quantization-aware training [Krishnamoorthi, 2018a, Esser et al., 2020] would also improve performance. More generally, there are a plethora of methods from the model compression literature that could be applied to COIN++ [Cheng et al., 2020a, Liang et al., 2021b]. For large scale data, it would be interesting to model the global structure of patches instead of encoding and entropy coding them independently. Further, the field of INRs is progressing

rapidly and these advances are likely to improve COIN++ too. For example, Martel et al. [2021] use adaptive patches to scale INRs to gigapixel images - such a partition of the input is similar to the variable size blocks used in BPG [Bellard, 2014]. In addition, using better activation functions [Ramasinghe and Lucey, 2021] to increase PSNR and equilibrium models [Huang et al., 2021] to reduce memory usage are exciting avenues for future research.

Finally, as COIN++ replaces the encoder in traditional neural compression with a flexible optimization procedure and the decoder with a powerful functional representation, we believe compression with INRs has great potential. Advances in INRs, combined with more sophisticated entropy coding and quantization may allow COIN-like algorithms to equal or even surpass SOTA codecs, while potentially allowing for compression on currently unexplored modalities.

Acknowledgments

We would like to thank Hyunjik Kim, Danilo Rezende, Dan Rosenbaum and Ali Eslami for helpful discussions. We thank Jean-Francois Ton for helpful discussions around meta-learning and for reviewing an early version of the paper. Emilien gratefully acknowledges his PhD funding from Google DeepMind.

Appendix

4.A Dataset details

4.A.1 Vimeo90k

We use the Vimeo90k triplet dataset [Xue et al., 2019] containing 73,171 3-frame sequences from videos at a resolution of 448×256. We processed the dataset following Bégaint et al. [2020]. The resulting dataset contains 153,939 training images and 11,346 test images.

4.A.2 FastMRI

To generate the dataset, we use the validation split from the FastMRI brain multicoil database [Zbontar et al., 2018]. This contains 1378 fully sampled brain MRI images obtained through a variety of sources - T1, T1 post-contrast, T2 and FLAIR images. We then filter the dataset to only use scans from the T2 source. In addition, as the vast majority of volumes have 16 slices, we also filter by volumes with 16 slices. We then randomly split the filtered scans into a 565 training volumes and 212 testing volumes. The train dataset contains the following shapes (with their counts):

(16, 384, 384): 329

(16, 320, 320): 229

(16, 384, 312): 2

(16, 320, 260): 2

(16, 320, 240): 1

(16, 384, 342): 1

(16, 320, 270): 1

While the test dataset contains the following shapes (with their counts):

(16, 384, 384): 124

(16, 320, 320): 86

(16, 320, 260): 2

We also normalize the data to lie in [0, 1] (while COIN++ can handle data in any range, we cannot apply the image compression baselines if the data is not in [0, 1]). As the data contains outliers, we first compute a histogram of the data distribution and choose the maximum value such that 99.99% of the data has value less than this. We then normalize by the minimum and maximum value and clip any value lying outside this range (<0.01% of the data).

Disclaimer required when using the FastMRI dataset: *"Data used in the preparation of this article were obtained from the NYU fastMRI Initiative database (fastmri.med.nyu.edu) [Zbontar et al., 2018, Knoll et al., 2020]. As such, NYU fastMRI investigators provided data but did not participate in analysis or writing of this report. A listing of NYU fastMRI investigators, subject to updates, can be found at:fastmri.med.nyu.edu. The primary goal of fastMRI is to test whether machine learning can aid in the reconstruction of medical images."*

4.A.3 ERA5

The climate dataset was extracted from the ERA5 database [Hersbach et al., 2019], using the processing and splits from Dupont et al. [2022c] (see this reference for details). The resulting dataset contains 12,096 grids of size 46×90, with 8510 training examples, 1166 validation examples and 2420 test examples.

4.A.4 LibriSpeech

The LibriSpeech dataset [Panayotov et al., 2015] contains several hours of read English Speech recorded at 16kHz. For training, we use the train-clean-100 split containing 28,539 examples and the test-clean split containing 2,620 examples. We train and evaluate on the first 3 seconds of every example, corresponding to 48,000 audio samples per example.

4.B Experimental details

4.B.1 CIFAR10

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 64. All models were trained for 500 epochs (400k iterations). We used the following architectures:

- latent dim: 128, 10 layers of width 512
- latent dim: 256, 10 layers of width 512
- latent dim: 384, 10 layers of width 512
- latent dim: 512, 15 layers of width 512
- latent dim: 768, 15 layers of width 512
- latent dim: 1024, 15 layers of width 512

We used 10 inner steps at test time for all models.

INR baselines. We used a latent dimension of 384 for all models. For Chan et al. [2021c], we used a ReLU MLP with a single hidden layer with 512 units to map the latent vector to the modulations (we found that more layers performed worse). For Mehta et al. [2021], we used the exact model proposed in their paper and set the dimension of the $\mathbf{z}$ vector to 384.

COIN baseline. We manually searched for the best architecture for each bpp level. We followed all other hyperparameters from COIN [Dupont et al., 2021] and trained for 10k iterations (we found this was enough to converge on CIFAR10). Surprisingly, we found that for CIFAR10 depth did not improve performance and that increasing the width of the layers was better. This may be because the layers are already very small.

- bpp: 3.6, 2 layers of width 12
- bpp: 4.6, 2 layers of width 14

- bpp: 5.8, 2 layers of width 16
- bpp: 7.1, 2 layers of width 18
- bpp: 8.5, 2 layers of width 20
- bpp: 10.0, 2 layers of width 22

For the COIN quantization experiments, we used uniform quantization for the weights and biases separately. We chose the number of standard deviations k at which to define the quantization range using the formula $k = 3 + 3\frac{\text{number of bits}-1}{15}$. I.e. when using 1 bit, we use 3 standard deviations and when using 16 bits we use 6 standard deviations. Indeed, there is a tradeoff between how much data we are cutting off and how finely we can quantize the range. We found that this formula generally gave robust results across different bit values.

Autoencoder baselines. All autoencoder baselines were trained using the CompressAI implementations [Bégaint et al., 2020]. In order for these models to handle 32×32 images from the CIFAR10 dataset, we modified the architectures both for BMS and CST. Specifically, for BMS we changed the last two convolutional layers in the encoder from kernel size 5, stride 2 convolutions to kernel size 3 stride 1 convolutions, in order to preserve the spatial size (we made similar changes for the transposed convolutions in the decoder). For CST we replaced the first three residual blocks in the image encoder with stride 1 convolutions instead of stride 2, hence preserving the size of the image. Similarly, we replaced the upsampling operations in the decoder with stride 1 upsampling (i.e. dimension preserving convolutions) instead of stride 2. Otherwise, we used the default parameters provided by CompressAI, i.e. for BMS, we used N=128 and M=192 and for CST N=128. We trained all models for 500 epochs with a learning rate of 1e-4. We trained models for each of the following λ values: [0.0016, 0.0032, 0.0075, 0.015, 0.03, 0.05, 0.1, 0.15, 0.3, 0.5]. As particularly CST could be unstable to train, we trained two models for each value of λ and kept the best model for the rate distortion plot.

Standard image codec baselines. We use three image codec baselines: JPEG [Wallace, 1992], JPEG2000 [Skodras et al., 2001] and BPG [Bellard, 2014]. For each of these, we perform a search over either the quality, quantization level or compression ratio to find the best quality image (in terms of PSNR) at a given bpp level.

We use the JPEG implementation from Pillow version 8.1.0. We use the OpenJPEG version 2.4.0 implementation of JPEG2000, calling the binary file with

`opj_compress -i <in filepath> -r <compression ratio> -o <out filepath>`.

We use BPG version 0.9.8, calling the binary file with

`bpgenc -f 444 -q <quantization level> -o <out filepath> <in filepath>`.

Encoding time. We measure the encoding time of COIN and COIN++ on a 1080Ti GPU. For COIN we fit a separate neural network for each image in the CIFAR10 test set and report the average encoding time. For COIN++ we similarly fit modulations for each image in the test set and report the average encoding time. For BPG, we measured encoding time on an AMD Ryzen 5 3600 (12) at 3.600GHz with 32GB of RAM.

4.B.2 Kodak and Vimeo90k

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 1e-6 and batch size 64. All models were trained for 600 epochs (1.4 million iterations). We used the following architectures:

- latent dim: 16, 10 layers of width 512
- latent dim: 32, 10 layers of width 512
- latent dim: 64, 10 layers of width 512
- latent dim: 96, 10 layers of width 512
- latent dim: 128, 10 layers of width 512

We used 32×32 patches from the Vimeo90k dataset to train the model and evaluated on the full Kodak images. We used 3 inner steps for the latent dim 32 and 64 models and 10 inner steps for the latent dim 16, 96 and 128 models as this gave the best results. We quantized all modulations to 5 bits.

4.B.3 FastMRI

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 16. All models were trained for 32,000 epochs (1.1 million iterations). We used the following architectures:

- latent dim: 16, 10 layers of width 512
- latent dim: 32, 10 layers of width 512
- latent dim: 64, 10 layers of width 512
- latent dim: 128, 10 layers of width 512

We trained on 16×16×16 patches and evaluated on the full volumes. We used 10 inner steps at encoding time as this gave the best results. On the rate distortion plot, the first two points are the latent dim 16 model, quantized to 5 and 6 bits, then the latent dim 32 model, quantized to 5 bits, then the latent dim 64 model quantized to 6 bits and finally the latent dim 128 model, quantized to 5 bits and 6 bits.

4.B.4 ERA5

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 3e-6 and batch size 32. All models were trained for 800 epochs (210k iterations). We used the following architectures:

- latent dim: 4, 10 layers of width 384
- latent dim: 8, 10 layers of width 384
- latent dim: 12, 10 layers of width 384

We used 3 inner steps at encoding time as this gave the best results. On the rate distortion plot, the first two points are the latent dim 4 and 8 models quantized to 5 bits, then the latent dim 8 model quantized to 6 and 7 bits and finally the latent dim 12 model quantized to 7 and 8 bits.

4.B.5 LibriSpeech

For all models, we set $\omega_0 = 50$ and used an inner learning rate of 1e-2, an outer learning rate of 1e-6 and batch size 64. We further scaled the coordinates to lie in $[-5, 5]$ as we found this improved performance (similar observations were made by Sitzmann et al. [2020b]). All models were trained for 1000 epochs (445k iterations), except the latent dim 256 model which was trained for 2000 epochs (890k iterations). We used the following architectures:

- latent dim: 128, 10 layers of width 512, patch size 1600
- latent dim: 128, 10 layers of width 512, patch size 800
- latent dim: 128, 10 layers of width 512, patch size 400
- latent dim: 128, 10 layers of width 512, patch size 200
- latent dim: 256, 10 layers of width 512, patch size 200

We used 3 inner steps at encoding time. On the rate distortion plot, each point corresponds to one of the above models quantized to 5, 6, 6, 7 and 7 bits respectively.

Audio codec baselines. We use the MP3 implementation from LAME version 3.100, calling the binary file with

`lame -b <bit rate> <in filepath> <out filepath>`.

4.C Figure details

Figure 4.8b (COIN vs COIN++ quantization). The results in this figure are averaged across the entire CIFAR10 test set. We used COIN and COIN++ models

that achieve roughly the same PSNR (30-31dB), corresponding to the bpp 7.1 model for COIN and the latent dim 384 model for COIN++.

Figure 4.8d (Encoding time). The BPG model uses 1.25 bpp (PSNR: 28.7dB), the COIN++ model 1.14 bpp (PSNR: 28.9dB) and the COIN model 7.1 bpp (PSNR: 30.7dB).

Figure 4.10 (Kodak qualitative samples). The COIN++ model used for this plot has a bpp of 0.537 (latent dim 128).

Figure 4.12 (FastMRI qualitative samples). The COIN++ model used for this plot has a bpp of 0.168 (latent dim 128).

Figure 4.9 (ERA5 qualitative samples). The COIN++ model used for this plot has a bpp of 0.012 (latent dim 8).

Figure 4.15 (Qualitative quantization). This figure uses the COIN++ model with a latent dim of 768.

4.D Things we tried that didn't work

- As MAML is very memory intensive, we experimented with first-order approximations. We ran first-order MAML as described in Finn et al. [2017], but found that this severely hindered performance. Further, methods such as REPTILE [Nichol et al., 2018] are not applicable to our problem, as the weights updated in the inner and outer loop are not the same.

- Mehta et al. [2021] use a similar approach to us for fitting INRs (without meta-learning) by using overlapping patches in images. However, we found that using overlapping patches yielded a worse tradeoff between reconstruction accuracy and number of modulations and therefore used non-overlapping patches throughout.

- We experimented with using a deep MLP (and the architecture from Mehta et al. [2021]) for mapping the latent vector to modulations but found that this decreased performance. As MLPs are strictly more expressive than linear mappings, we hypothesize that this is due to optimization issues arising from

the meta-learning. If the base network is learned without meta-learning, it is likely a deep MLP would improve performance over a linear mapping.

4.E Additional results

4.E.1 Meta-learning curves

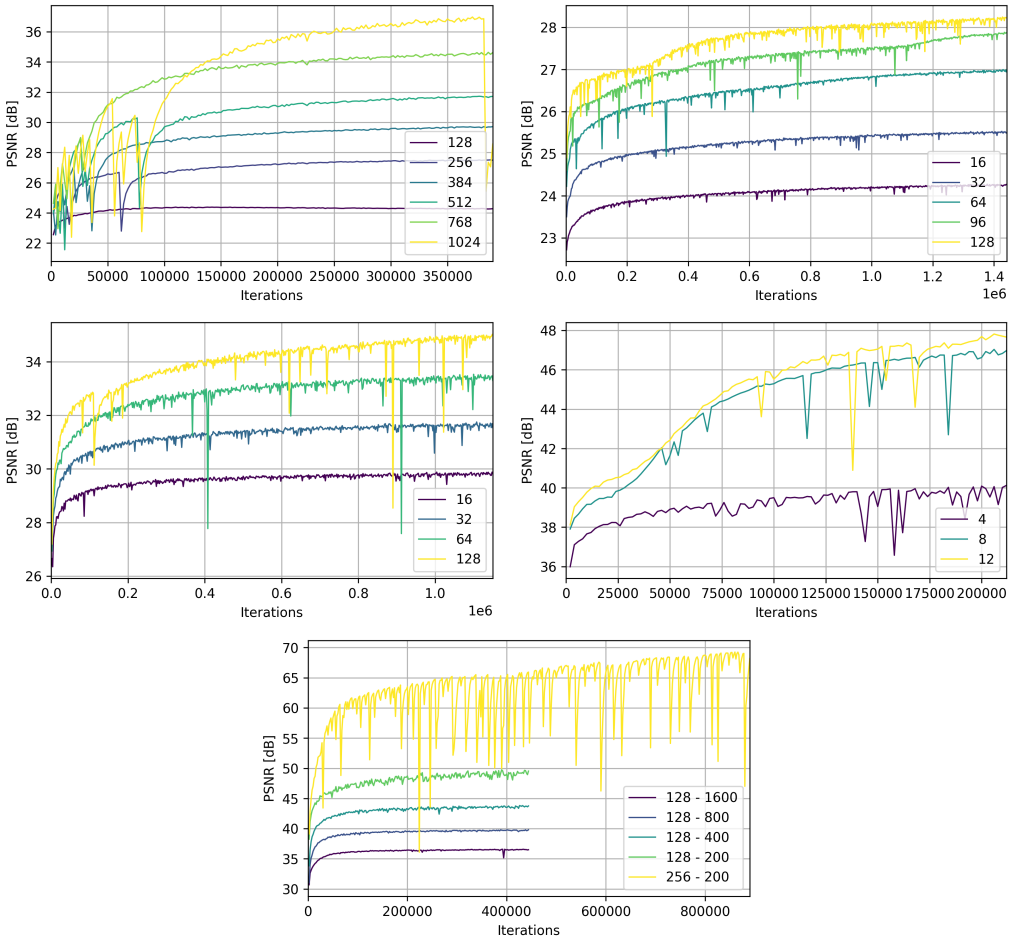

Figure 4.13: Validation PSNR (3 inner steps) during meta-learning on CIFAR10 (top left), Kodak (top right), FastMRI (middle left), ERA5 (middle right) and LibriSpeech (bottom). Note that for LibriSpeech, the legend corresponds to "latent dimension - patch size".

4.E.2 CIFAR10 ablations

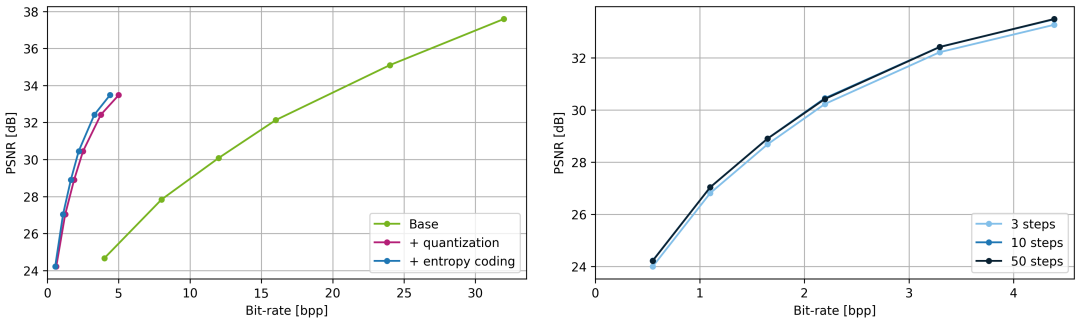

Figure 4.14: (Left) Effect of of quantization (to 5 bits) and entropy coding on CIFAR10. (Right) Effect of number of inner steps on CIFAR10 for a model that has been quantized to 5 bits, with entropy coding. While we use 3 inner steps for meta-learning, performing 10 steps at test time leads to an increase in reconstruction performance of 0.5-1.5dB, while fitting for more than 10 steps generally does not improve performance. Indeed, curves for 10 and 50 steps almost fully overlap.

4.E.3 Qualitative quantization results

Figure 4.15: Qualitative effects of quantization. The top row shows ground truth data from MNIST and CIFAR10, the second row shows the reconstructions from full precision (32 bit) modulations. The subsequent rows show reconstructions when quantizing to various bitwidths. As can be seen, with only 5 bits, reconstructions are nearly perfect. Using as few as 1 or 2 bits, the class of the object is generally recognizable.

4.E.4 Encoding curves

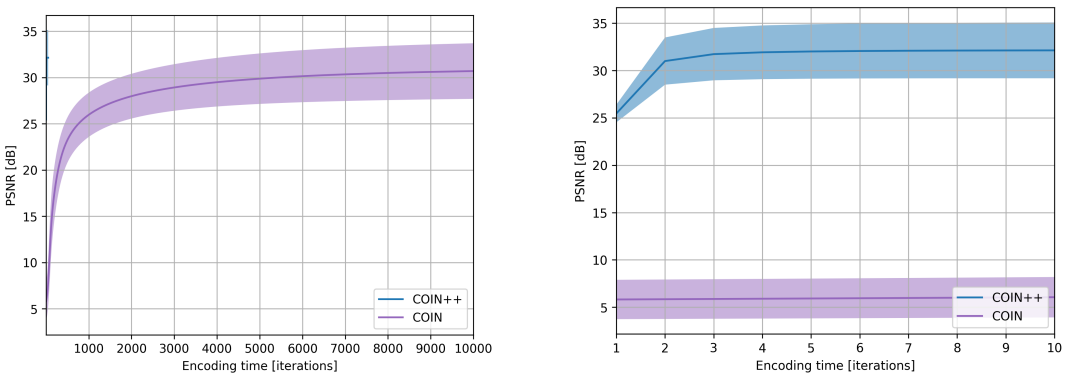

Figure 4.16: Encoding curves for COIN and COIN++ on CIFAR10 (full curve on the left, zoomed in version on the right). The COIN model has a bpp of 7.1, while COIN++ has a bpp of 2.2.

4.E.5 Additional qualitative results

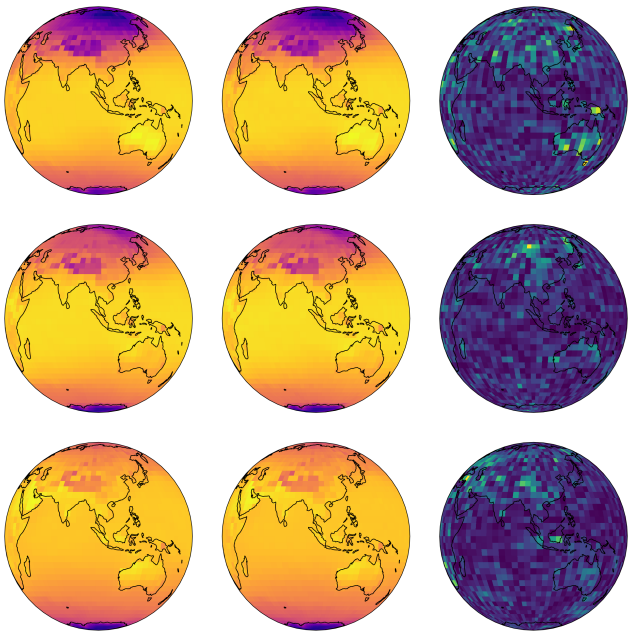

Figure 4.17: Qualitative compression artifacts on ERA5 using the latent dim 8 model with 0.012 bpp (original in first column, COIN++ in second column and residual in third column).

113

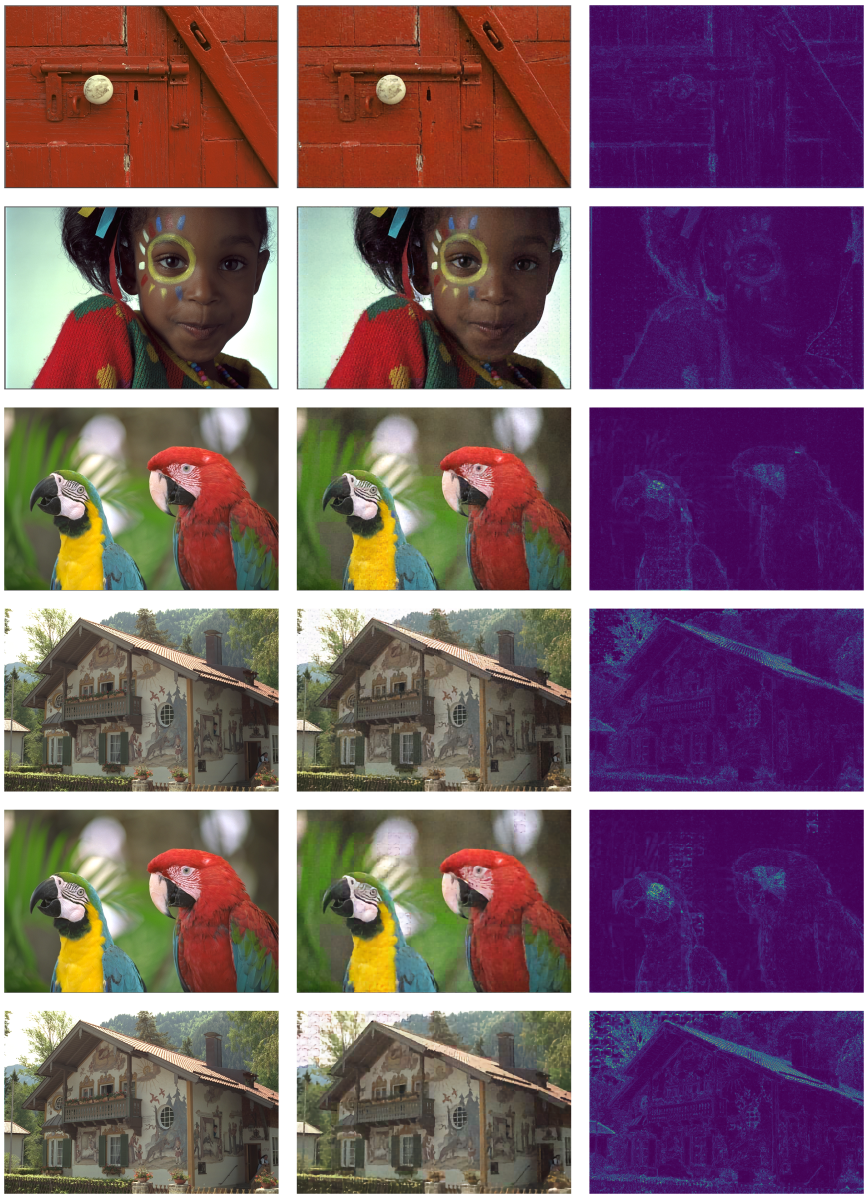

Figure 4.18: Qualitative compression artifacts on Kodak (original in first column, COIN++ in second column and residual in third column). The first 4 rows correspond to the model with latent dim 128 (0.537 bpp), while the bottom two rows correspond to the model with latent dim 64 (0.398 bpp).

114

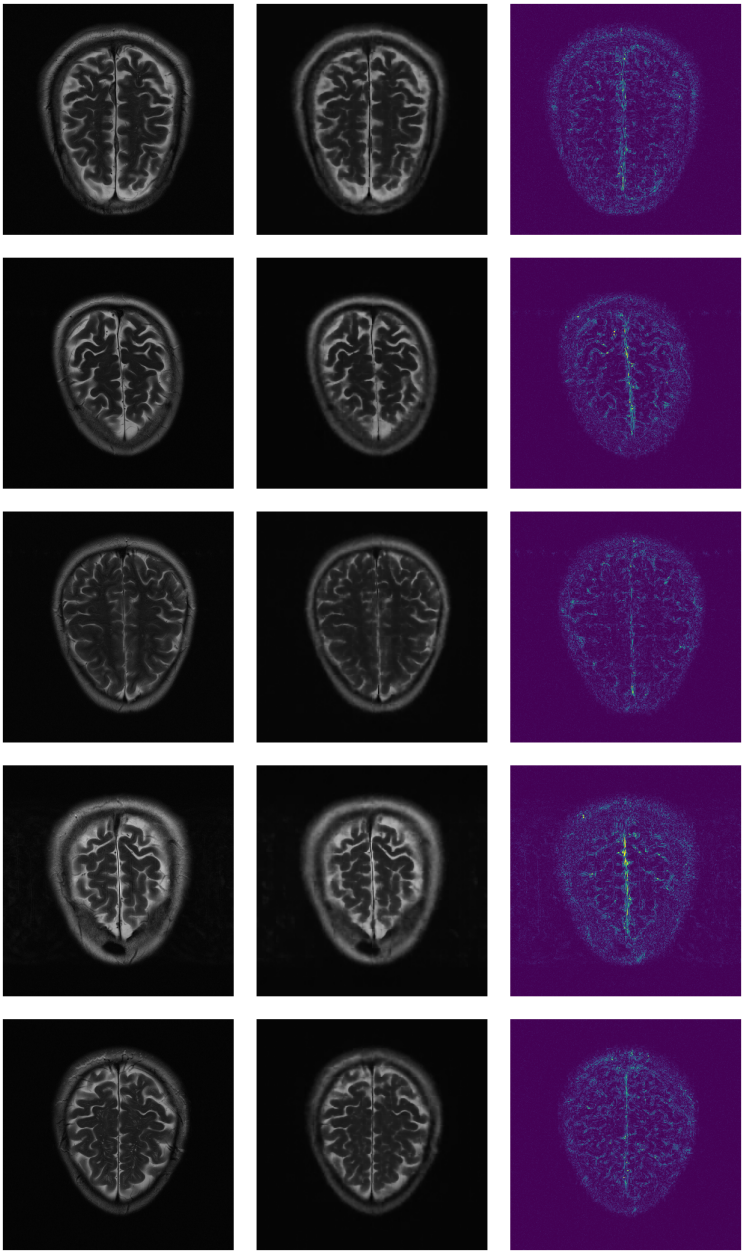

Figure 4.19: Qualitative compression artifacts on FastMRI using the latent dim 128 model with 0.168 bpp (original in first column, COIN++ in second column and residual in third column).

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	COIN++: Neural Compression Across Modalities
Publication Status	□ Published □ Accepted for Publication X Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	*Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, Arnaud Doucet*, Under review at TMLR.

Student Confirmation

Student Name:	Emilien Dupont		
Contribution to the Paper	I had the idea for the project which was refined in discussions with all other authors. Hrushikesh, Milad, Adam and I all wrote code and ran experiments. Hrushikesh performed several ablation studies. I wrote the final codebase, ran meta-learning experiments and ran all quantization and entropy coding experiments. I wrote the paper with all other authors providing feedback on the manuscript.		
Signature 		Date	September 19, 2022

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: *Professor Yee Whye Teh*		
Supervisor comments		
Signature 	Date	23 Sep 2022

5

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

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.

5.1 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.

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 and Zhang, 2019], signed distance functions [Park et al., 2019], videos [Li et al., 2021b], 3D scenes [Mildenhall et al., 2020], audio [Sitzmann et al., 2020b] and data on manifolds [Dupont et al., 2022c]. 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

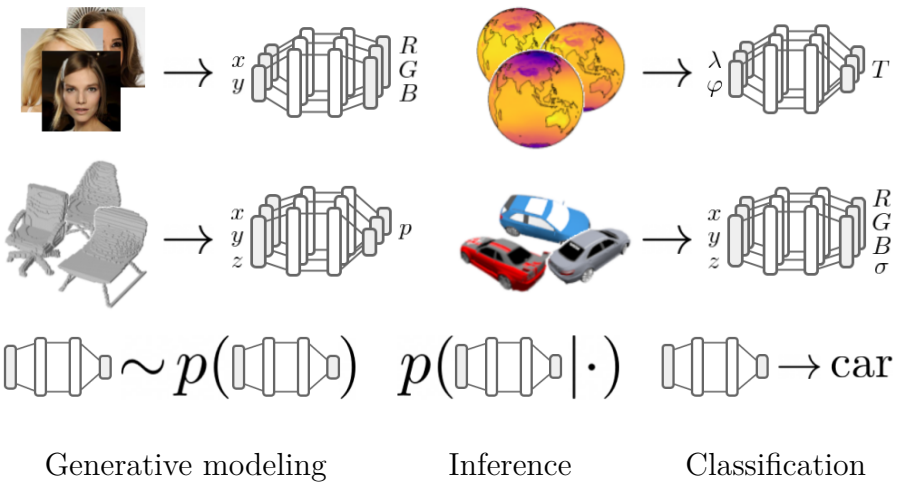

Figure 5.1: We convert array data into functional data parameterized by neural networks, termed *functa*, and treat these as data points for various downstream machine learning tasks.

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 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 5.1). A key difference to prior work on multimodal learning of functions [Dupont et al., 2022c, 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.

5.2 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:

$$\min_\theta \mathcal{L}(f_\theta, \{\mathbf{x}_i, \mathbf{f}_i\}_{i\in\mathcal{I}}) = \min_\theta \sum_{i\in\mathcal{I}} \|f_\theta(\mathbf{x}_i) - \mathbf{f}_i\|_2^2. \tag{5.1}$$

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.

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 5.2), 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.

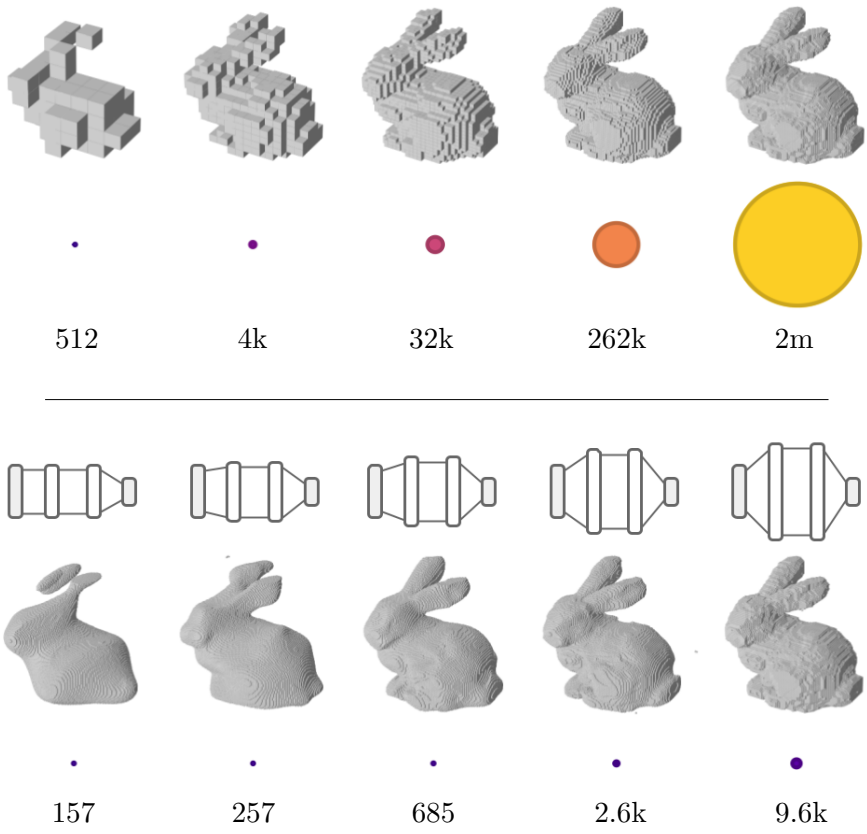

Figure 5.2: Functa scale much more gracefully with resolution than array representations. Circle area reflects the numerical size of the array (top) / function (bottom). See 5.A.9 for details.

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. 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.

5.3 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.

5.3.1 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 (see Section 5.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.,

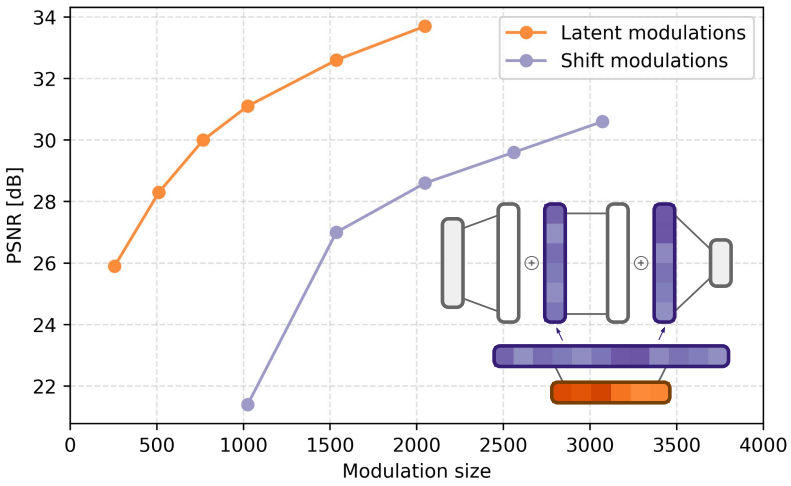

Figure 5.3: Reconstruction accuracy (in PSNR) vs modulation dimensionality on CelebA-HQ 64×64. Reconstruction accuracy is computed from the MSE between the image array and its INR evaluated at each pixel location. The model architecture is shown on the bottom right, with purple vectors corresponding to shift modulations and orange vectors to latent modulations.

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 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 5.A.2, along with intuition for the role of shift modulations via perturbation analysis in 5.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 5.3). 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 5.3 shows that this approach results in the best tradeoff between reconstruction accuracy (Equation 5.1) 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 5.4. We also experimented with other architectures for representing functa but found that they performed worse (see Section 5.B.2).

5.3.2 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 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 (see 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

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 $\in \{1, \ldots, N_{inner}\}$ 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

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 5.7.

5.4 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 5.2.

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 and 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 and 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 (see Section 5.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 Sections 5.A.4 and 5.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:

$$\min_{\phi} -\log p(\phi) + \lambda \sum_{i \in \mathcal{I}} \| f_\phi(\mathbf{x}_i) - \mathbf{f}_i \|_2^2 \tag{5.2}$$

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 (see Figure 5.7). In terms of Equation 5.2, $\mathcal{I}$ 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 5.2. 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).

5.5 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 and 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 and Lipman, 2021]. A series of recent works have built generative models for NeRF

scenes mostly using GANs [Schwarz et al., 2020, Niemeyer and 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., 2022c] 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., 2020b, Alayrac et al., 2020]. However the recently introduced Perceiver [Jaegle et al., 2021b,a] uses a shared architecture 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 and 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 and Kumar [2021], Lu and 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.

5.6 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., 2019b] 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 5.C.

5.6.1 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 5.4, we are able to fit modulations to a high degree of accuracy in only 3 gradient steps across a range of data modalities. Table 5.1 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 (see Section 5.6.4). Qualitative examples of reconstructions for each data modality are shown in Figure 5.18 of Section 5.D.

5.6.2 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 5.5, the diffusion model is able

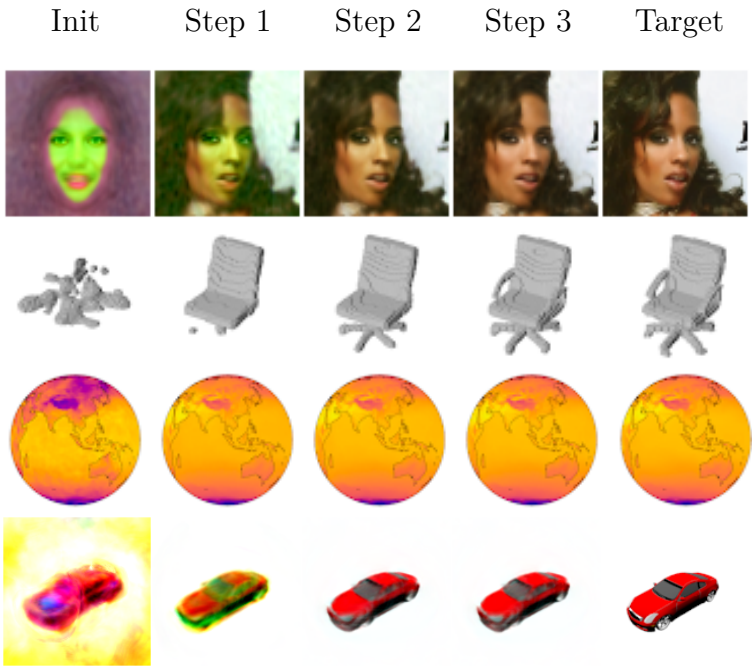

Figure 5.4: Visualization of the meta-learned initialization, each gradient step of the inner loop and target data point. GIF showing scenes at different poses: `bit.ly/3nBjIDO`

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 5.1: Mean reconstruction of modulations across each dataset vs modulation size. Metric is voxel accuracy (%) for ShapeNet and PSNR (dB) for the rest. See Section 5.A.3 for details on metric.

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 5.19 in the appendix. Our model achieves an FID score [Heusel et al., 2017] of 40.4, but

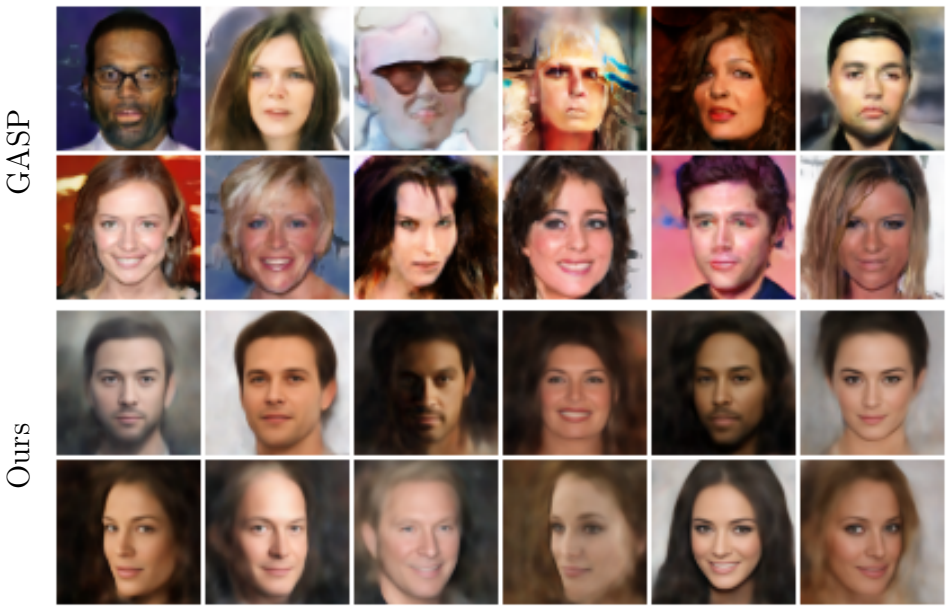

Figure 5.5: Uncurated samples from GASP and DDPM (diffusion) trained on 256-dim CelebA-HQ 64×64 modulations.

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 Section 5.A.8 for discussion).

5.6.3 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 (see Section 5.A.1). We train NSF [Durkan et al., 2019] on a 256 dimensional functaset of the chairs category, with unconditional samples shown in Figure 5.6 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., 2022c]. Our model can also be used to produce semantically meaningful latent interpolations as shown in Figure 5.23 in 5.D. Figure 5.8 shows samples from a class-conditional NSF trained on 10 classes from ShapeNet, where our model

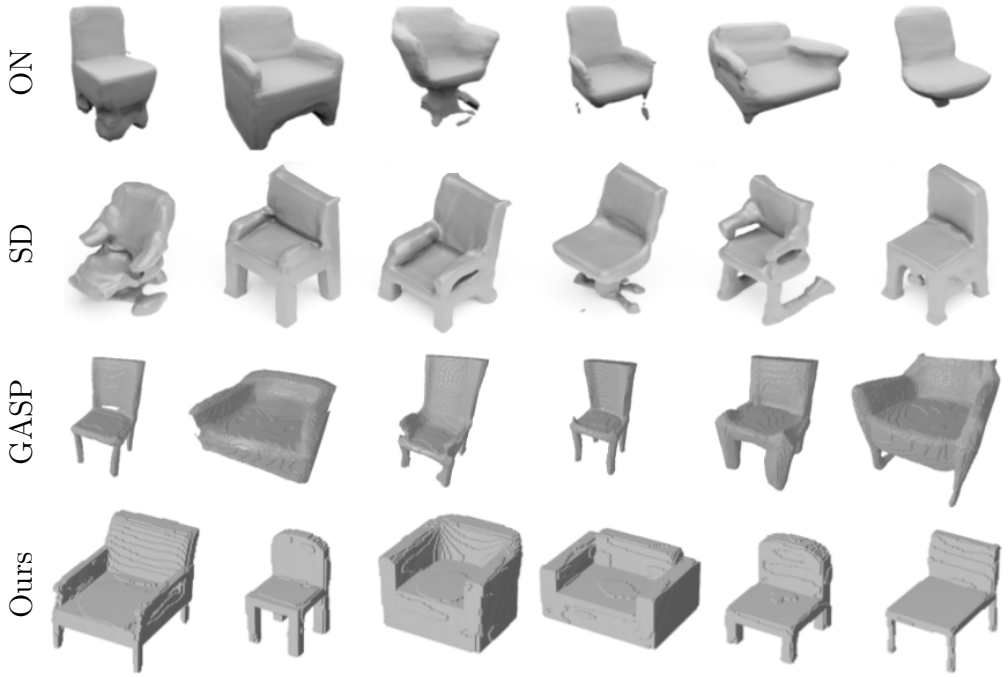

Figure 5.6: Unconditional samples from our model and three baselines: Occupancy Networks trained as a VAE (ON) from Mescheder et al. [2019], a GAN trained on signed distance functions with a set discriminator (SD) from Kleineberg et al. [2020] and GASP from Dupont et al. [2022c].

Figure 5.7: 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 5.27

produces realistic and consistent samples for each class. Finally, we show imputation results in Figure 5.7, where our model is able to infer the shape of the chair from various partial observations, including a simulated lidar scan (last column).

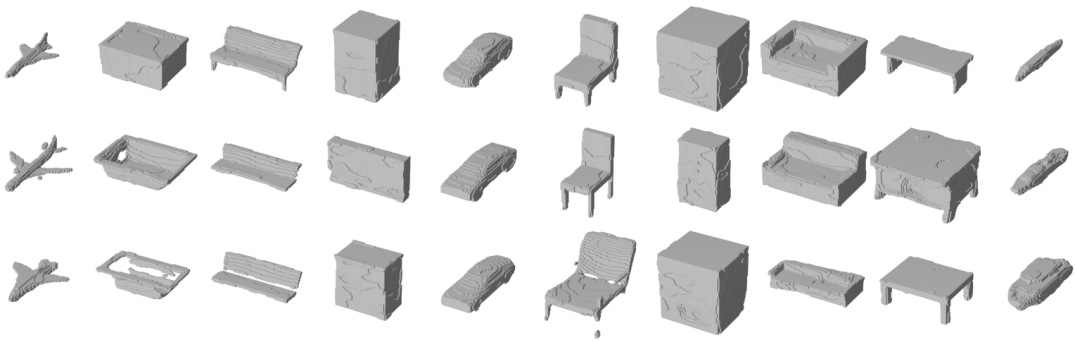

Figure 5.8: Uncurated samples from a class-conditional flow trained on 256-dim modulations of ShapeNet 10 Classes 64^3.

Figure 5.9: Uncurated samples from π-GAN and DDPM trained on 64-dim modulations of SRN Cars. GIF showing different poses: `bit.ly/33Ojhzz`

5.6.4 NeRF scenes

We evaluate our framework on NeRF scenes by using the SRN Cars dataset [Sitzmann et al., 2019b], containing 2458 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 (see Figure 5.1). Views of the scene are then generated via volume rendering, and modulations are fitted by minimizing reconstruction loss on the available set

Figure 5.10: Latent interpolation between two car scenes with moving pose. GIF: `bit.ly/3g41w0Y`

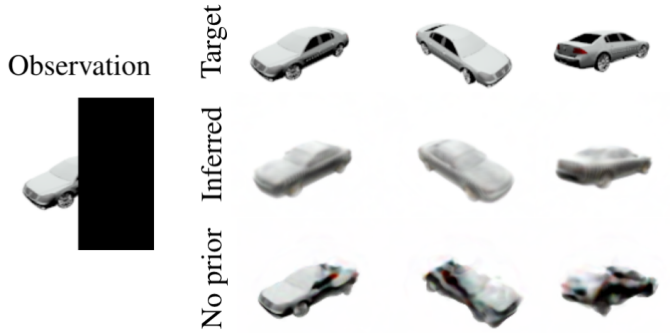

Figure 5.11: Novel view synthesis from occluded test scene. Without the prior, the model is not able to correctly infer the shape of the car. GIF: `bit.ly/3rREHDT`. Additional scenes: Figure 5.28

of posed images on training scenes (see 5.A for background and details). While NeRF uses several tricks to achieve good performance, we use a minimal bare bones model for our experiments (see Section 5.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 5.9, DDPM is able to generate plausible NeRF scenes of cars, giving an FID score of 80.3 (see Section 5.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 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 5.10, NSF gives smooth interpolations both in terms of shape and texture. We also perform novel view synthesis experiments in Figure 5.11. 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 and 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) 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.

5.6.5 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. [2022c]. 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 5.12).

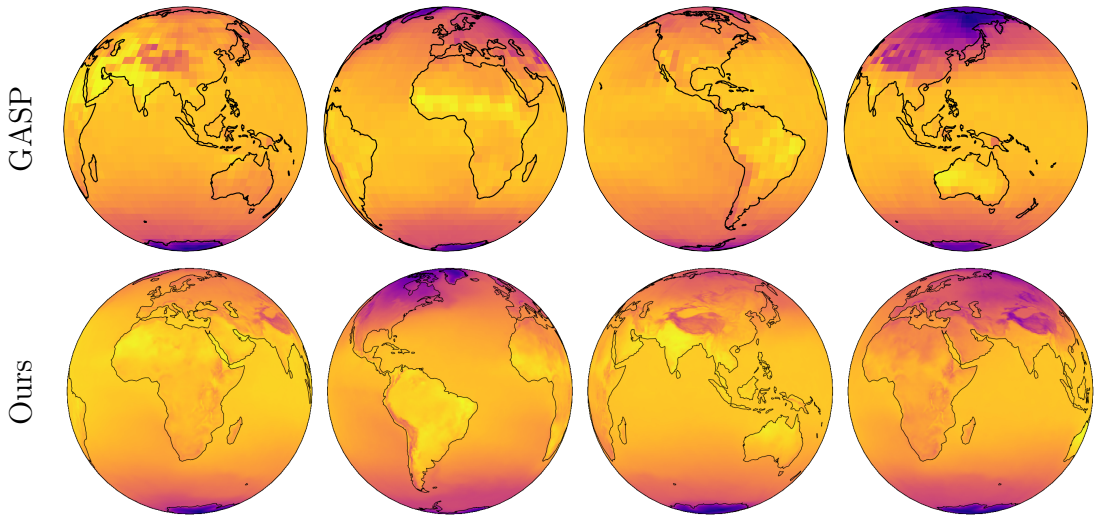

Figure 5.12: Uncurated samples from GASP and a flow trained on 256-dim modulations of ERA5. GIF: `bit.ly/35ya6nr`

5.6.6 Classification

Classifier	Test accuracy	n_{params}
MLP on Functa	$93.6 \pm 0.1\%$	83k
3D CNN	$93.3 \pm 0.3\%$	550k

Table 5.2: 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 and Scherer [2015]. As shown in Table 5.2, 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 5.A.7 for experimental details and 5.D for further results and training curves.

5.7 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 and 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 and 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., 2021, 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., 2021b] exist). In addition, learning distributions of functa is likely to improve entropy coding and hence compression for these frameworks [Ballé et al., 2017].

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.

Appendix

5.A Experimental details

5.A.1 Datasets

CelebA-HQ 64 × 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 5.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.

5.A.2 SIREN and modulations

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

$$\mathbf{x} \mapsto \sin(\omega_0(\mathbf{W}\mathbf{x} + \mathbf{b})) \tag{5.3}$$

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:

$$\mathbf{x} \mapsto \sin(\omega_0(\mathbf{W}^{(i)}\mathbf{x} + \mathbf{b}^{(i)} + \mathbf{s}^{(i)})) \tag{5.4}$$

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 5.4), 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.

5.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 5.3: Selected hyperparameter values for meta-learning functa for each dataset.

For the outer loop, we use Adam [Kingma and 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 5.3 we provide hyperparameter values for each dataset.

For SRN Cars, we provide further details about subsampling views and pixels in 5.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 5.1 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 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.

5.A.4 Neural Spline Flows (NSF)

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

$$\mathbf{x} = f_\theta(\mathbf{z}), \quad \mathbf{z} \sim p(\mathbf{z}) \tag{5.5}$$

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

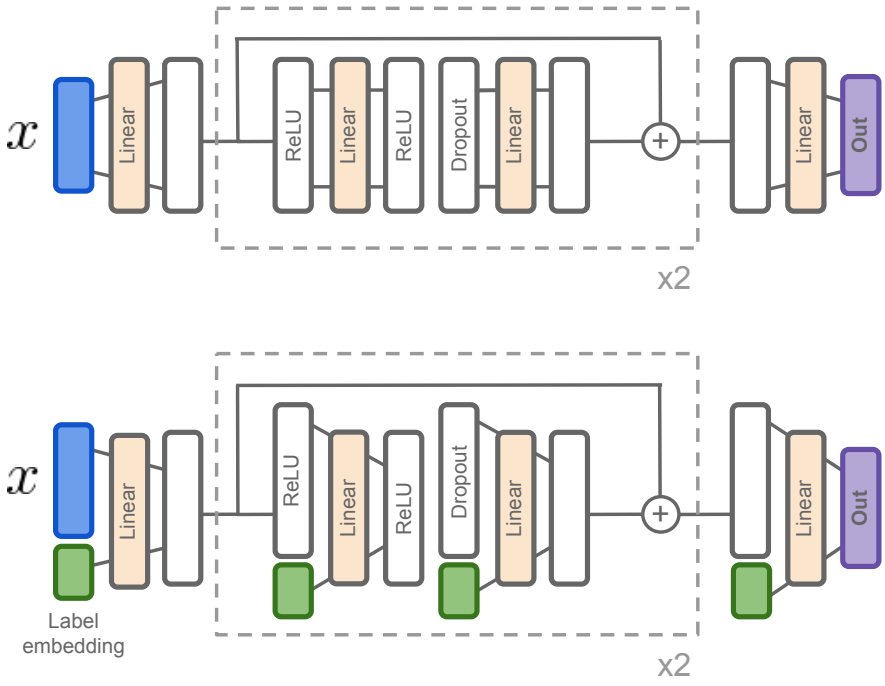

Figure 5.13: Architecture for neural spline flow conditioner g_θ for the unconditional flow (top) and the class-conditional flow (bottom).

maximising the log probability of the data under the flow, that is computed by the change of variable formula:

$$\log p(\mathbf{x}) = \log p(f_\theta^{-1}(\mathbf{x})) + \log \left| \frac{\partial f_\theta^{-1}}{\partial \mathbf{x}} \right|. \tag{5.6}$$

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:

$$[\mathbf{y}_{1:d}, \mathbf{y}_{d+1:2d}] = [\mathbf{x}_{1:d}, b(\mathbf{x}_{d+1:2d}; g_\theta(\mathbf{x}_{1:d}))] \tag{5.7}$$

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 (see Durkan et al. [2019] for a detailed formulation of rational quadratic spline bijectors). See Figure 5.13 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 5.4 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	flow layers	dropout	num iters
ShapeNet Chairs	64	320	0.2	1e6
ShapeNet 10 Classes	64	128	0.3	1e6
ERA5 temperature	256	32	0.2	2e5
SRN Cars	64	8	0.3	1e5

Table 5.4: Selected hyperparameter values for NSF for each dataset. All datasets have modulation dimension 256, except SRN Cars which has modulation dimension 512.

5.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

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) \coloneqq \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \quad q(\mathbf{x}_t|\mathbf{x}_{t-1}) \coloneqq \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{5.8}$$

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

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{5.9}$$

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:

$$p(\mathbf{x}_T) \coloneqq \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}), \quad p_\theta(\mathbf{x}_{0:T}) \coloneqq p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{5.10}$$

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \coloneqq \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{5.11}$$

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

$$\mathbb{E}_q \left[\mathrm{KL}[q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)] \right] \tag{5.12}$$

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):

$$\|\boldsymbol{\epsilon_t} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2 \tag{5.13}$$

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 $\beta_{1:T} =$`np.linspace`$(10^{-4}, 0.02, T)$ following [Ho et al., 2020], and parameterize $\boldsymbol{\epsilon}_\theta$ as in Figure 5.14, 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 (see Section 5.6.2)

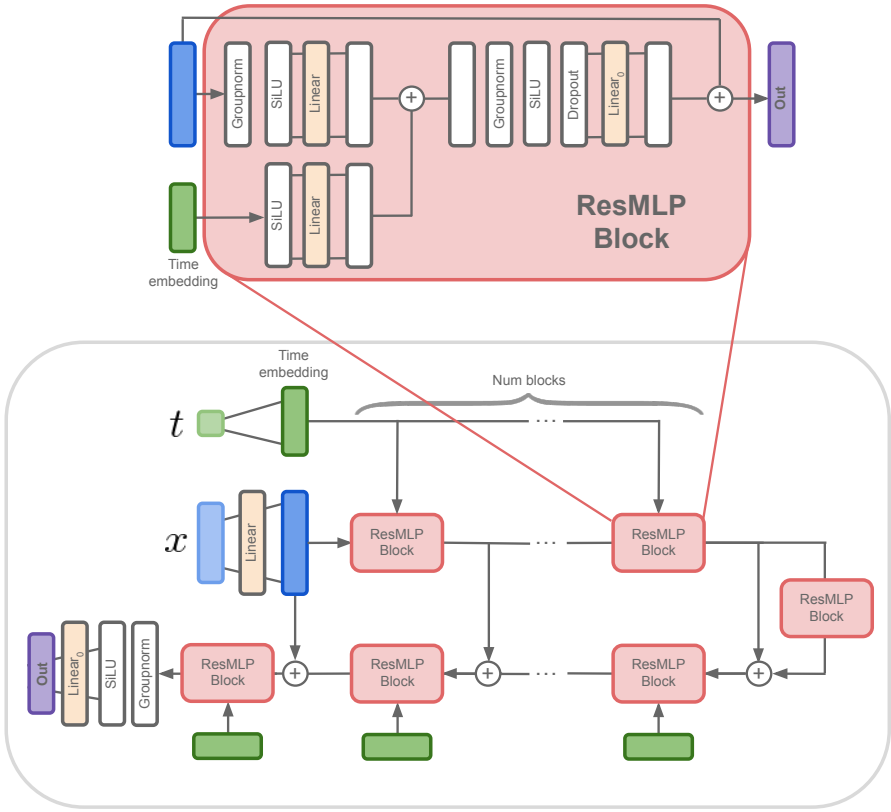

Figure 5.14: Architecture for ϵ_θ in our implementation of DDPM. Linear_0 stands for a linear layer with zero initialization.

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.

5.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 5.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 5.1) 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 128×128×3 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)

```
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 5.1: Minimal volumetric rendering

5.A.7 Classification

We use the default MLP in Haiku [Hennigan et al., 2020] with SiLU/swish activations [Hendrycks and 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 and 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 5.6.6, 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 5.D we show results for larger models.

5.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 (see Figure 5.18). 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].

5.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 (see Section 5.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 5.5.

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 5.5: Hyperparameter values used for Figure 5.2.

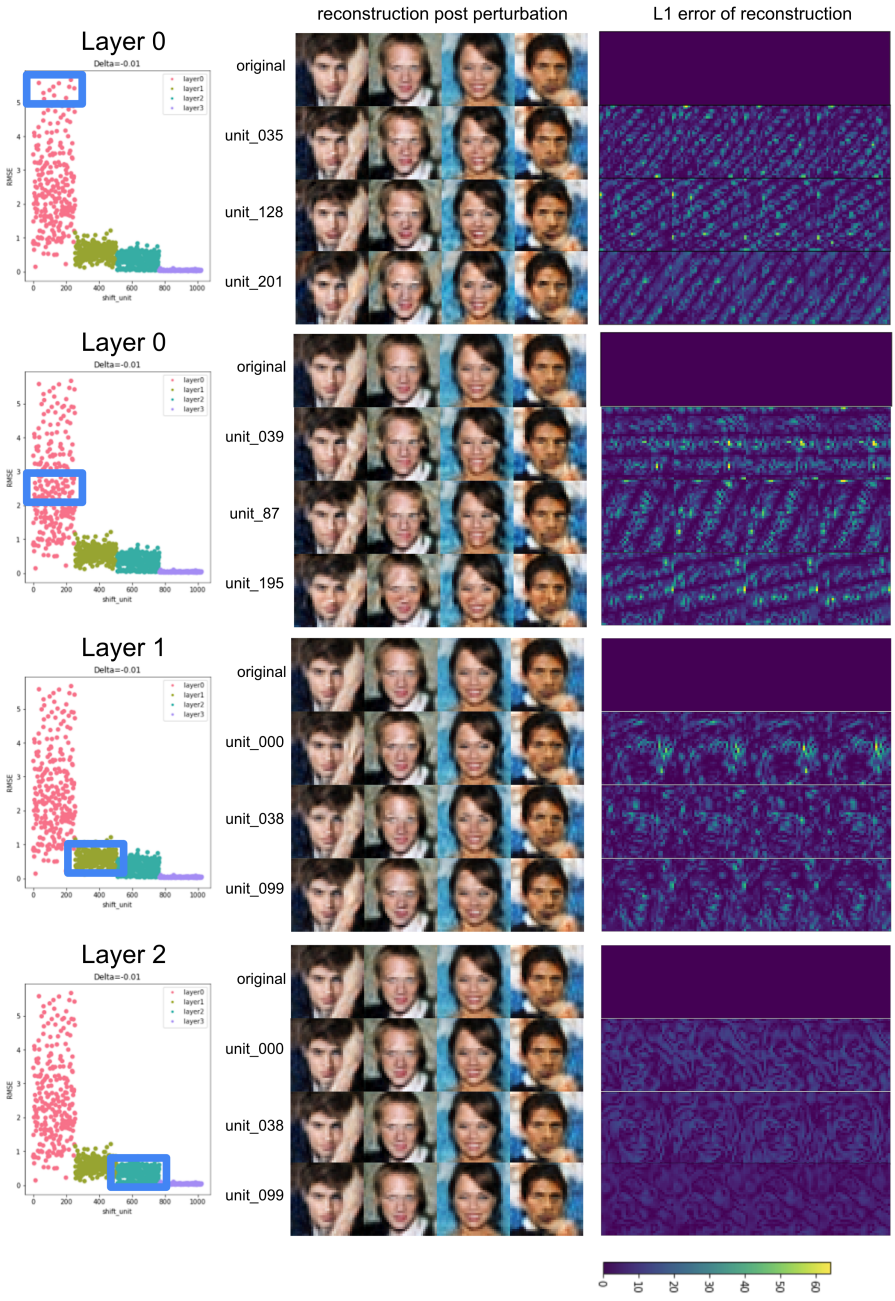

Figure 5.15: Visualization of errors in pixel space when perturbing individual modulation dimensions of each layer by 0.01.

5.B Modulation analysis

5.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 5.15, 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 and Lucey, 2021] or Multiplicative Gabor Networks [Fathony et al., 2020] whose modulations may correspond to more localised information.

5.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 5.3.1:

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 5.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 5.1.

5.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 5.16 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

Figure 5.16: Uncurated samples from autoregressive Transformer trained on 256-dim modulations for CelebA-HQ 64×64.

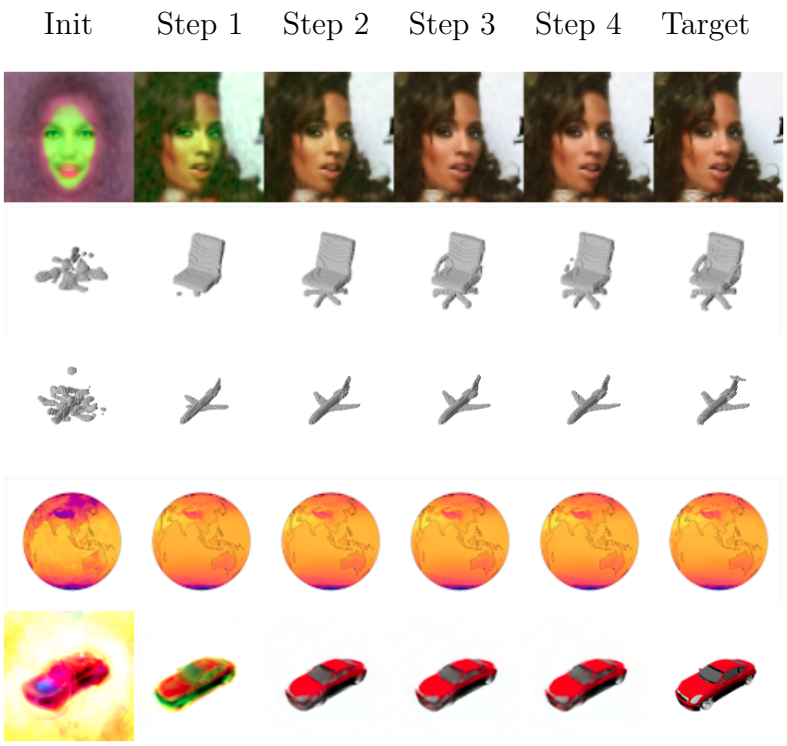

Figure 5.17: Same as Figure 5.4 but with an extra inner loop step, and extra row for ShapeNet 10 classes.

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 5.17 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.

5.D Further Results

Original vs Reconstructions from modulations. In Figure 5.18 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 5.1). 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 5.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 5.19 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

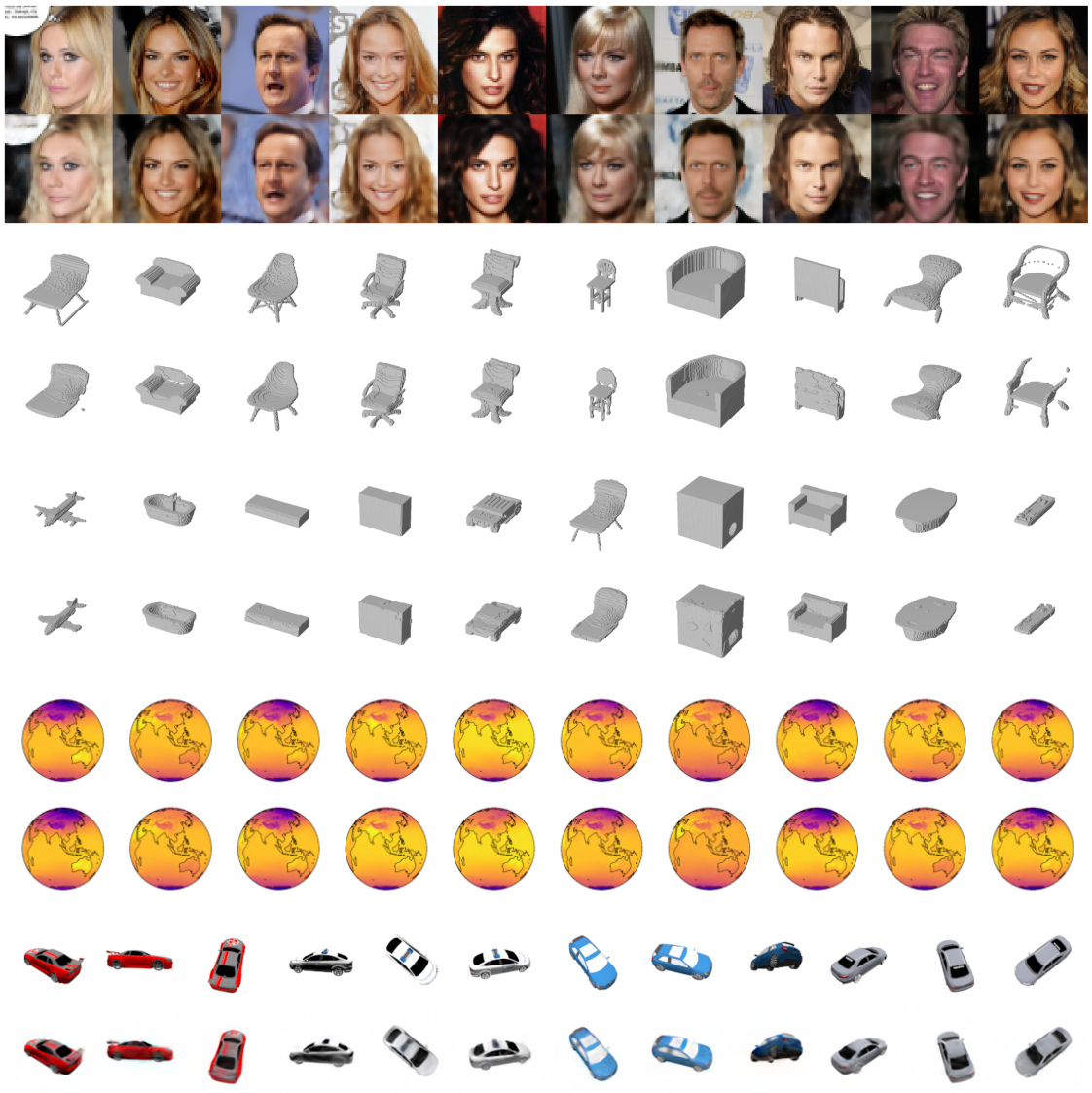

Figure 5.18: Original vs reconstruction for 256-dimensional modulations on the first few data points of each training dataset.

5.20. 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 5.21 we show uncurated samples from the NSF trained on 256-dimensional modulations of

Figure 5.19: 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 5.20: Comparison of rendering sample at 512 resolution vs rendering at 64 resolution then upsampling to 512 by bicubic interpolation

ShapeNet Chairs. We also show samples at various temperatures of the base distribution in Figure 5.22, where samples become more globally coherent with lower temperature but show less diversity. Additionally in Figure 5.25 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 and 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

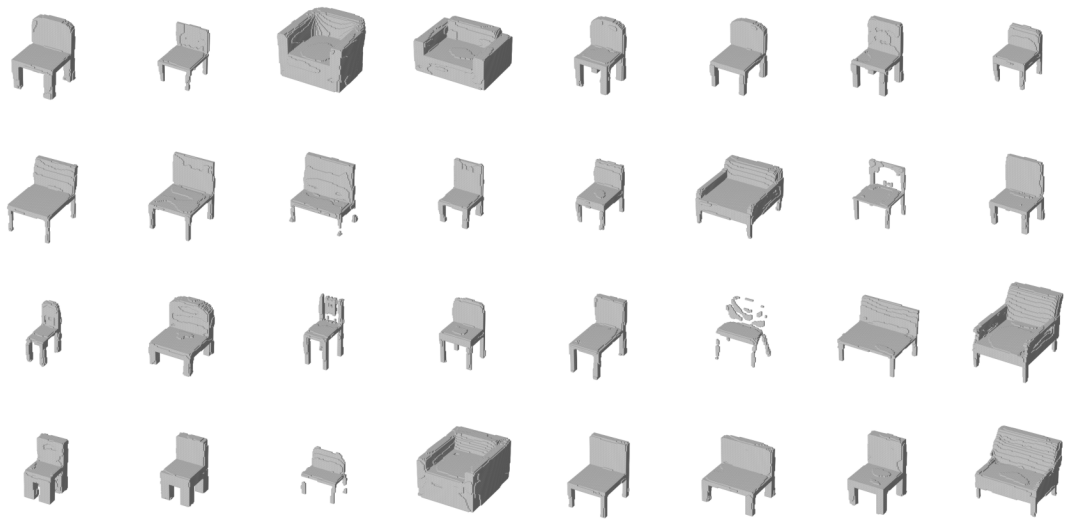

Figure 5.21: Uncurated samples from flow trained on 256-dim ShapeNet Chairs 64^3 modulations, temperature 0.95.

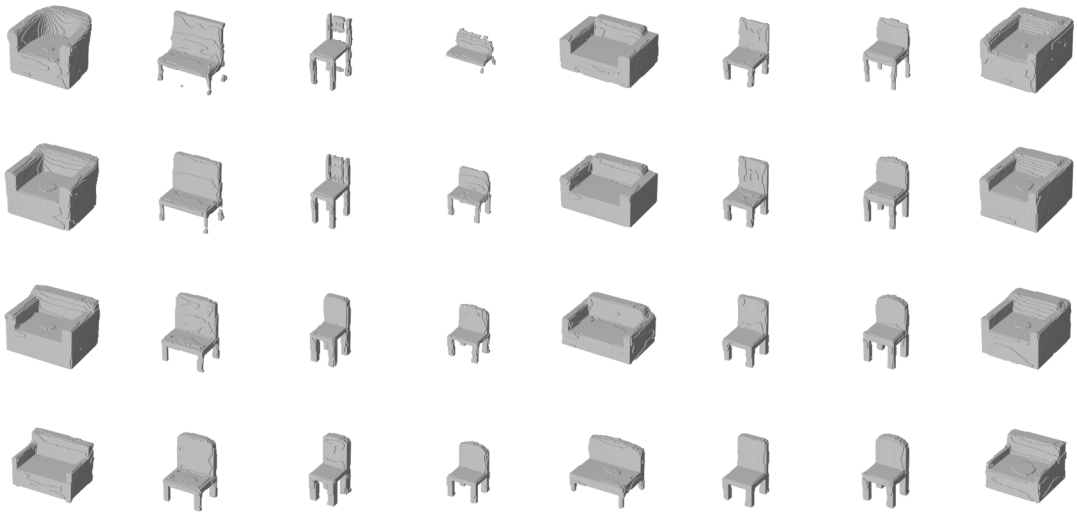

Figure 5.22: Uncurated samples from flow trained on 256-dim modulations of ShapeNet Chairs 64^3, at varying temperatures. Temperatures for each row: 1.0, 0.9, 0.8, 0.7.

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.

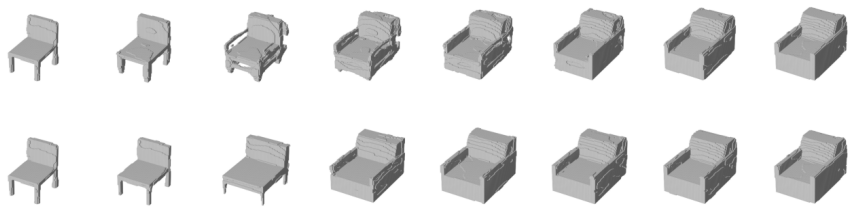

Figure 5.23: Comparison of interpolation in modulation space (top) and flow latent space (bottom) for 256-dim modulations of ShapeNet 10 Classes 64^3.

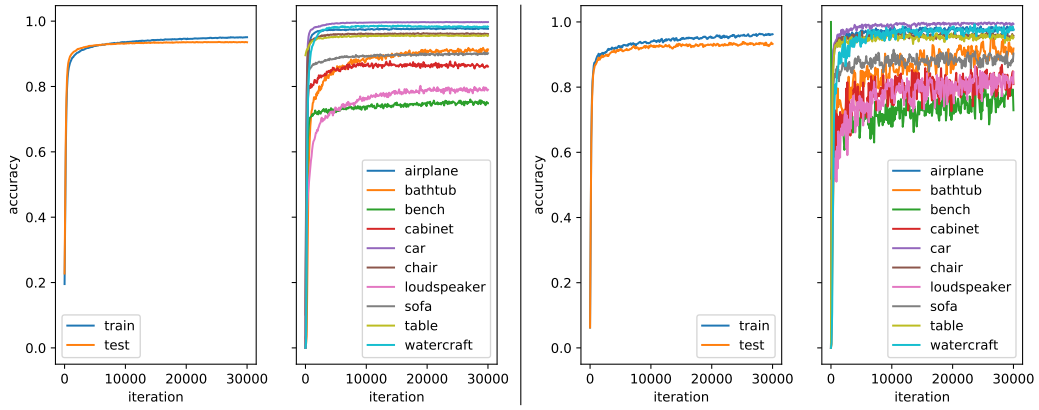

Figure 5.24: Left: Train/test (left) & per-class test (right) classification accuracy throughout training on 256-dim modulations of ShapeNet 10 Classes 64^3, using batch size 1024. Right: Same for 3D CNN, but using batch size 64 (GPU memory only allows up to 8 per device). Both Using exponential moving average with decay=0.99 and bs=1024.

ShapeNet 10 classification on functa for bigger classifiers. In Table 5.6 we show test accuracies and parameter counts for bigger models compared to those shown in 5.6.6. 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 5.24 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.

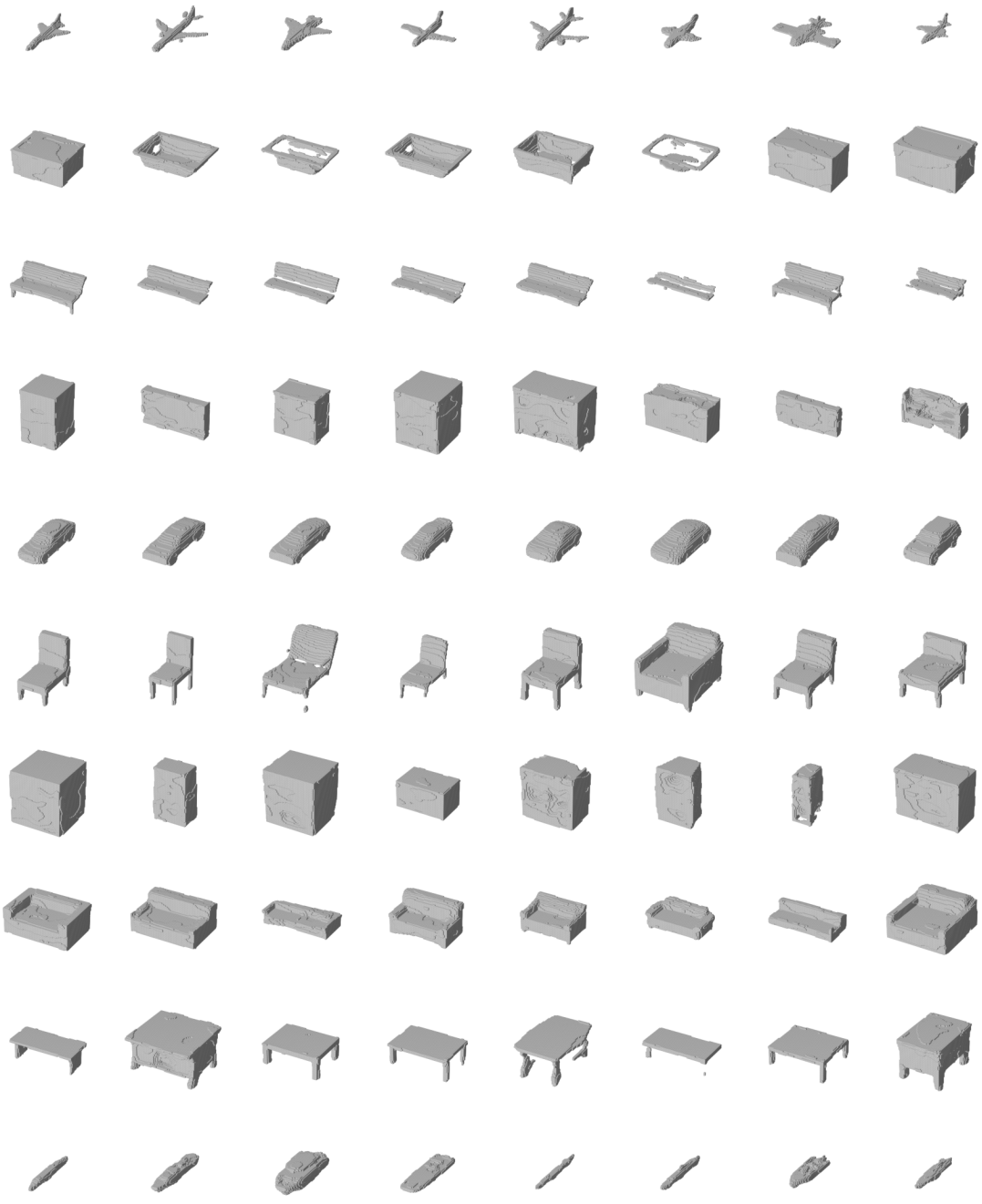

Figure 5.25: Uncurated samples from class-conditional flow trained on 256-dim modulations of ShapeNet 10 classes 64^3.

Figure 5.26: Uncurated samples from DDPM trained on 64-dim modulations of SRN Cars.

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

Table 5.6: Same as Table 5.2 but for a wider range of model sizes

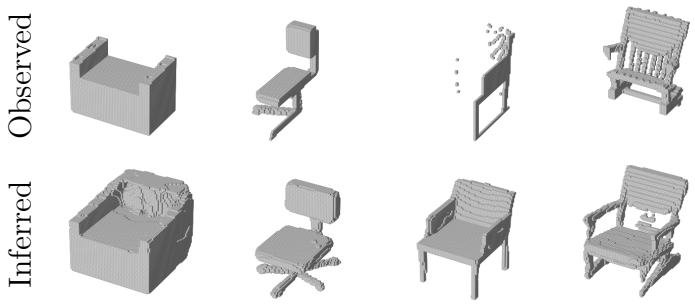

Figure 5.27: Additional results to Figure 5.7 for imputation of different chairs from the test set. GIF showing course of optimization: `bit.ly/3G7KDh8`

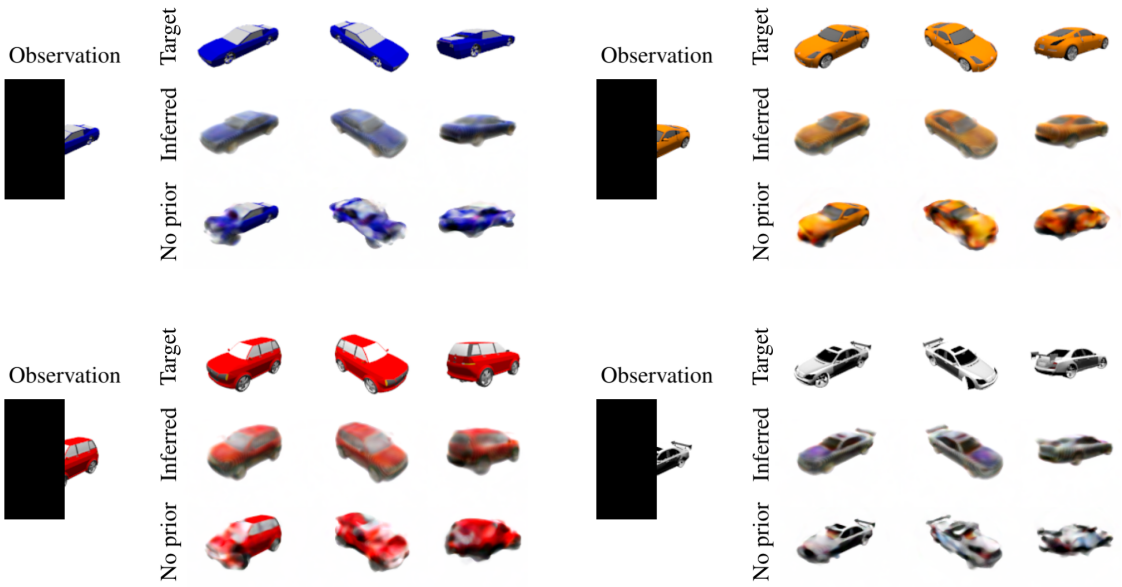

Figure 5.28: Additional results to Figure 5.11 for novel view synthesis from additional occluded test scenes. GIF: `bit.ly/3x842vO`

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	From data to functa: Your data point is a function and you can treat it like one
Publication Status	X Published □ Accepted for Publication □Submitted for Publication □Unpublished and unsubmitted work written in a manuscript style
Publication Details	*Emilien Dupont, Hyunjik Kim, Ali Eslami, Danilo Rezende, Dan Rosenbaum*, From data to functa: Your data point is a function and you can treat it like one, ICML 2022

Student Confirmation

Student Name:	Emilien Dupont		
Contribution to the Paper	I had the initial idea for the project which was refined and expanded through discussions with all other authors. I wrote the initial codebase and ran experiments, particularly on meta-learning. Danilo wrote code for data loading and for initial versions of the generative models. I wrote code and ran experiments for early versions of the inference tasks and generative modeling. Hyunjik ran most experiments in the latter stages of the project and performed several ablations for the generative models and classifiers. Hyunjik and I wrote the NeRF code and ran experiments together. Hyunjik and I wrote the paper, with all other authors providing feedback on the manuscript.		
Signature 		Date	September 19, 2022

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: *Professor Yee Whye Teh*		
Supervisor comments This was Emilien's internship project at DeepMind.		
Signature 	Date	23 Sep 2022

6

Further exploration, future directions and conclusion

There are a range of interesting directions for future work involving the treatment of INRs as data. In this chapter, we provide a few concrete examples (and some preliminary experiments) highlighting promising research directions and important aspects of learning families of INRs.

6.1 Spatial functa

A key problem in scaling up the functa framework in its current parameterization is that there is no obvious inductive bias for models trained on functa. Indeed, most layers in the classifiers and generative models we trained on functa are simply fully connected layers. This typically means we need to train the models on large amounts of data or use various forms of regularization to ensure they generalize well. In contrast, convolutional models trained directly on images typically require much less data to generalize well, because of the (useful) translation invariance that is built into the architecture. It would therefore be interesting to endow functa with spatial structure such that we could take advantage of e.g. convolutions for the downstream models. Another advantage of this approach is that the performance of certain classes of models is heavily dependent on the use of convolutions. For example,

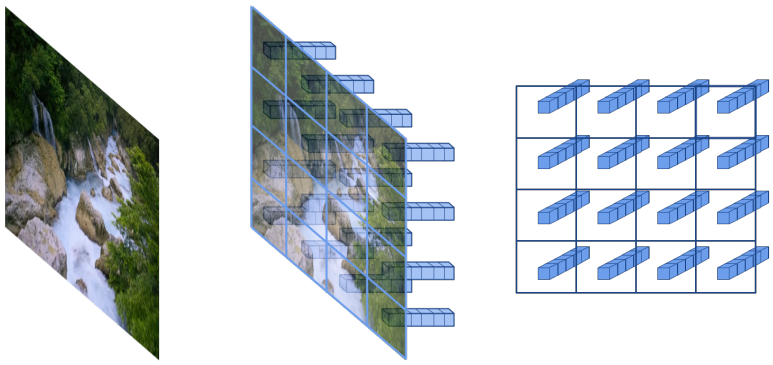

Figure 6.1: We can split an image (left) into patches and fit modulations to each patch (middle). The resulting set of spatially arranged modulations (right) then forms a feature volume we can pass to standard CNNs.

the denoising function in diffusion models is almost always parameterized by a convolutional UNet [Ho et al., 2020] which is crucial for achieving good performance.

While there are several ways of constructing "spatial functa", the simplest method likely consists in splitting the input data into patches and fitting modulations to each patch. On images for example, we could, in a similar way to COIN++ on the large Kodak images, split the image into patches and train a shared base network across patches, while fitting modulations individually to each patch. These spatially arranged modulation vectors then form a convolutional feature volume which could be passed to standard CNNs and diffusion models (see Figure 6.1). This approach not only lets us train convolutional models on functa, it also retains several advantages of using functa in the first place. Indeed, we can typically fit patches to high accuracy using a small number of modulations, which means the resulting models will be trained on highly compressed formats, potentially allowing for training on large scale images or video. Most importantly, such a model would still be applicable to data that does not lie on a grid, including NeRF scenes. Indeed, recent works such as KiloNeRF [Reiser et al., 2021], partition 3D space into patches and fit NeRF functions to each patch leading not only to good reconstructions, but also faster evaluation. We therefore believe spatial functa are a promising approach for the (currently intractable) problem of building large scale generative models of 3D scenes.

6.2 Bilevel optimization

Another important aspect of learning families of functional representations is optimization. For example, for COIN++ and functa, we use a linear map from the latent vector to the shift modulations. However, using an MLP, which is strictly more expressive than a linear layer, leads to significantly worse performance (with the performance counterintuitively degrading as more layers are added to the MLP). This strongly suggests that there is an issue with optimization of the hypernetwork-like layers from the latent vector to the modulations. Resolving this issue could then lead to more compact representations (and therefore also potentially better compression). While devising good optimization algorithms for this setting is likely difficult, we formalize the problem below, link it to current optimization procedures and suggest potential avenues for future work.

To formalize this problem, we assume we are given a dataset $\{\mathbf{d}_i\}_{i=1}^n$ of n datapoints. As in COIN++, we then wish to find shared parameters θ and instance specific parameters ϕ_i such that we minimize some loss $\mathcal{L}(\theta, \phi_i, \mathbf{d}_i)$ per data point. Typically, this loss would be given by the ℓ_2 difference between the data point and an INR approximating the data point, i.e. $\mathcal{L}(\theta, \phi_i, \mathbf{d}_i) = \|\mathbf{d}_i - f(\theta, \phi_i)\|_2^2$ (where we assume f returns the values of the INR evaluated at all coordinates where $\mathbf{d}_i$ is defined). The optimization problem we then seek to solve is

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \min_{\phi_i} \mathcal{L}(\theta, \phi_i, \mathbf{d}_i), \tag{6.1}$$

that is we wish to minimize a sum of minima. This is an instance of a *bilevel optimization* problem, which are notoriously difficult to solve [Colson et al., 2007]. This optimization problem is extremely common when learning families of INRs (a few notable examples include [Park et al., 2019, Sitzmann et al., 2019b, 2020a, Tancik et al., 2021]) and is typically solved one of two ways: using MAML or auto decoding.

For COIN++ and functa, we use MAML/CAVIA. This corresponds to approximating the inner minimum ϕ_i^* over ϕ_i by a single (or a few) gradient steps (with step size α)

$$\phi_i^* \approx \phi_i - \alpha \nabla_{\phi_i} \mathcal{L}(\theta, \mathbf{x}_i, \phi_i). \tag{6.2}$$

The outer minimization problem is then solved by performing gradient descent on Equation 6.1 using the approximation of the inner minimum from Equation 6.2. Similarly, the widely used auto-decoder setting described in 2.B corresponds to performing alternating gradient descent on θ and $\{\phi_i\}_{i=1}^n$. However, it is likely that neither of these are optimal, and to the best of our knowledge, no other optimization procedures have been explored for families of INRs. Adapting algorithms from the bilevel optimization literature to the problem of learning families of INRs could then be promising. Indeed, several recent papers have had success in applying bilevel optimization procedures to deep learning problems [Ji et al., 2021, Arbel and Mairal, 2021], and we believe this is an exciting avenue for future research.

6.3 Parameterizing families of functions

INRs are typically represented by the parameters of an MLP. While this is often a reasonable choice when modeling a single data instance, it can become problematic when we treat a collection of INRs as a dataset for a downstream task. For example, we can consider an INR fit to a given image. Depending on the initialization of the INR and the optimization procedure, the exact same image can be accurately represented by INRs with vastly different sets of parameters. This is a problem when we treat INRs as data: there is not necessarily a one-to-one correspondence between the data and the parameters we use to represent them. Further, it is not obvious how to build a good metric on this space, since two images that are close in pixel space (or in some semantic space, such as the feature space of an image classifier) may be represented by completely different sets of parameters. This is partially mitigated by using modulations, but a unique relationship between data and the functions used to represent them is still not guaranteed. Further, as shown in the perturbation analysis in Section 5.B.1, small perturbations of the modulations can lead to large changes in the output.

An interesting direction for future work is then to think of novel ways to effectively parameterize a space of functions that have this uniqueness property (functions produce the same outputs if and only if they have the same parameters). It is likely

that ideas from functional analysis [Yosida, 2012] could be useful in this regard. For example, we could consider learning an orthogonal function basis and using the basis coefficients to parameterize individual functions. We believe that, ultimately, MLP weights are not the "right" parameterization for INRs and that devising better ways to parameterize families of INRs is an important direction for future work.

6.4 Compression

As described in Section 1.3.6, INRs could in principle match (or improve upon) the reconstruction quality of the autoencoders typically used for image compression. Indeed, images provide a useful benchmark allowing us to compare INR based compression algorithms to highly optimized image codecs. As INR based codecs are applicable across a wide range of modalities, a COIN-like codec that equals or surpasses state of the art image codecs is then likely to be useful for a range of other data modalities where compression has been much less researched than for images.

In this section, we discuss three promising research directions (including some preliminary results) for improving compression with INRs: improved quantization, deep entropy coding and the rate-distortion perspective. However, there are also several other exciting research directions for compression, including sparsity which has been explored in recent follow up works to COIN [Lee et al., 2021, Schwarz and Teh, 2022, Ramirez and Gallego-Posada, 2022].

6.4.1 Improved quantization

In COIN++, we perform quantization by taking the mean and standard deviation across *all modulations*, then normalizing them (to have zero mean and unit variance) and finally clipping them to lie in $[-3, 3]$. The modulation values are then uniformly quantized in the range $[-3, 3]$. While this very simple approach works well, it can still lead to drops in PSNR of several dB, hurting compression performance. It is likely that more sophisticated quantization techniques (particularly quantization aware training) could reduce this drop significantly.

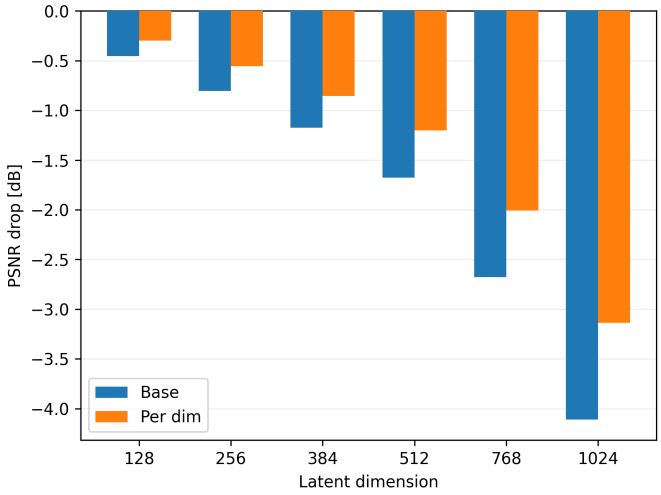

Figure 6.2: Drop in PSNR when quantizing 32-bit modulations to 5-bits using basic quantization (as in COIN++) and per modulation quantization.

To test whether improved quantization is likely to improve performance, we make a very simple change to the quantization used in COIN++. Instead of calculating the mean and variance across all modulations, we normalize the modulations *per dimension* and therefore use different quantization ranges for different modulations. Indeed, we found empirically that different modulation dimensions had different distributions and ranges, suggesting that a per dimension approach could work better.

We compared this approach to the one used in COIN++ on CIFAR10 across various modulation dimensions. As can be seen in Figure 6.2, per dimension quantization significantly reduces the drop in PSNR (nearly 1dB for the latent dimension 1024 model). This further highlights that the simple quantization technique used in COIN++ is far from optimal and that more sophisticated approaches are likely to improve performance.

6.4.2 Deep entropy coding

In order to further improve compression performance, we can losslessly compress the quantized modulations by entropy coding them. In order to do entropy coding, we need a probability model for the modulations, i.e. we need to model the distribution of modulations $\phi \in \{0, 1, ..., 2^b - 1\}^d$ where b is the number of bits we use per

modulation dimension and d is the number of modulations. Typically $b = 5$, so that we are modeling data in $\{0, 1, ..., 31\}^d$. We then assume the modulations come from some unknown distribution $P(\phi)$[1]. To model this discrete distribution, we use a continuous density model q over $\mathbb{R}^d$, since these are typically easier to optimize [Tran et al., 2019]. Given the continuous density model, we can define a PMF over the discrete modulation values by integrating q over hypercubes

$$Q(\phi) := \int_{[-0.5,0.5[^d} q(\phi + \mathbf{u}) d\mathbf{u}. \tag{6.3}$$

We can then add uniform noise $\mathbf{u} \in [-0.5, 0.5)^d$ to the modulations inducing a density p on the noisy modulations $\hat{\phi} = \phi + \mathbf{u}$. It can then be shown that [Theis et al., 2015]

$$\int p(\hat{\phi}) \log_2 q(\hat{\phi}) d\hat{\phi} \leq \sum_{\phi} P(\phi) \log_2 Q(\phi), \tag{6.4}$$

i.e. optimizing the continuous density model q over the noisy (or dequantized [Uria et al., 2013]) data is equivalent to optimizing a lower bound on the cross entropy between P and Q, which is the average number of bits required to compress samples from P using a code optimized for Q. Learning a good density model q could then allow us to more efficiently compress the modulations. We note that this is a standard approach for lossless compression [Yang et al., 2022].

In COIN++, we use a factorized distribution for entropy coding. However, as can be verified empirically, modulation dimensions are correlated, so a factorized distribution cannot model the true modulation distribution and hence will lead to suboptimal entropy coding and compression.

We can instead use normalizing flows to model the continuous (dequantized) distribution of modulations q and use this for entropy coding. To test this, we train a neural spline flow (using the `nflows` package [Durkan et al., 2020]) on the CIFAR10 128, 256 and 384 modulation datasets. Results for the 384-dimensional modulations are shown in Figure 6.3. As can be seen, deep entropy coding significantly improves performance. Indeed, with no entropy coding we require 5 bits per modulation

[1]This distribution is induced by the distribution of the data combined with our parameterization of the base network in a complicated way and is generally intractable.

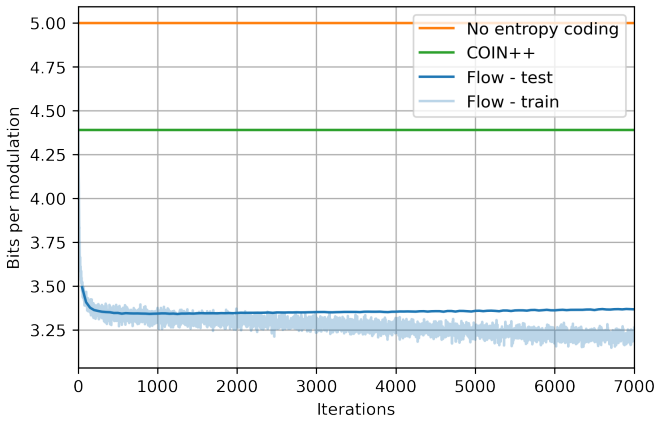

Figure 6.3: Number of bits required per modulation without entropy coding, with the simple entropy coding used in COIN++ and when entropy coding with a normalizing flow trained on modulations.

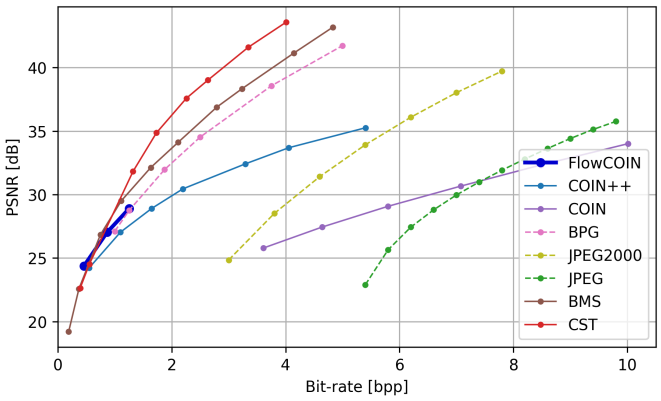

Figure 6.4: Rate distortion plot on CIFAR10.

(since we quantize the modulations to 5 bits), with the entropy coding scheme from COIN++ we require 4.39 bits per modulation, while with a neural spline flow we only require 3.34 bits per modulation.

We can also combine the per dimension quantization approach with the deep entropy coding scheme (calling the resulting model FlowCOIN) to generate a new rate-distortion curve. As can be seen in Figure 6.4, FlowCOIN performs better than COIN++ on CIFAR10 and even outperforms state of the art codecs at the lowest bitrate (we only evaluated FlowCOIN for latent dimensions 128, 256 and 384). These results further highlight the potential gains that can be obtained from improving both quantization and entropy coding for INR-based compression.

6.4.3 Rate-distortion perspective

A key component of nearly all modern neural compression methods is the joint optimization of rate (number of bits required to transmit signal) and distortion (reconstruction error of signal) [Yang et al., 2022]. Indeed, as first argued by Ballé et al. [2017], jointly optimizing rate and distortion is crucial for obtaining strong compression performance with autoencoders. However, in both COIN and COIN++, we use a two stage process: we first optimize for distortion (by fitting either the MLP weights or modulations) and then quantize and entropy code the representation after this (hence optimizing the rate). It would therefore be interesting to consider optimizing COIN-like models using a rate-distortion objective.

We can formulate a rate-distortion objective for COIN by analogy with the procedure for autoencoders. We assume we have data $\mathbf{d} \sim p(\mathbf{d})$, shared INR parameters θ and per instance parameters $\Phi(\mathbf{d})$ obtained through some optimization procedure Φ applied to a given datapoint $\mathbf{d}$ (e.g. for MAML Φ corresponds to a few steps of gradient descent in the inner loop). We further assume the parameters $\Phi(\mathbf{d})$ we transmit are quantized and denote these by $\tilde{\Phi}(\mathbf{d})$. Given some reconstruction error $\mathcal{L}(\theta, \tilde{\Phi}(\mathbf{d}), \mathbf{d})$, the distortion is then given by

$$D \coloneqq \mathbb{E}_{\mathbf{d} \sim p(\mathbf{d})}[\mathcal{L}(\theta, \tilde{\Phi}(\mathbf{d}), \mathbf{d})]. \tag{6.5}$$

The rate is given by the expected number of bits required to store the (discretized) per instance parameters $\tilde{\Phi}(\mathbf{d})$, i.e.

$$R \coloneqq \mathbb{E}_{\mathbf{d} \sim p(\mathbf{d})}[-\log_2 P(\tilde{\Phi}(\mathbf{d}))], \tag{6.6}$$

where P is a (potentially learned) probability model of the discretized per instance parameters $\tilde{\Phi}(\mathbf{x})$). The goal is then to minimize a weighted sum of the rate and distortion

$$\min_{\theta, \Phi} D + \lambda R, \tag{6.7}$$

where $\lambda > 0$[2]. Unlike standard rate distortion objectives, this is a bilevel optimization problem as Φ itself is an optimization procedure. Indeed, as described in Section 6.2, in COIN++ we minimize D only (also a bilevel optimization problem) using MAML. However, it would be interesting explore potential ways to incorporate the rate term R into this procedure.

Finally, we note that there are several other important ingredients for obtaining state of the art performance with autoencoders (including in particular the use of side information or hyperpriors [Ballé et al., 2018]). It is likely that incorporating such ideas into COIN-based models will also boost performance.

6.5 Conclusion and future outlook

In this thesis, we proposed to treat neural networks as data instead of models. We explored the feasibility and properties of such an approach, with a particular focus on compression and generative modeling.

In Chapter 1, we discussed the shortcomings of representing data as arrays in the context of deep learning. We also showed through simple experiments why representing data as neural networks might be compelling for certain tasks and data modalities.

In Chapter 2, we introduced GASP, a generative model of functions which, in addition to being agnostic to discretization, can be seamlessly applied to a range of different data modalities from images and 3D shapes to climate data. By acting only on functions and their input and output pairs, the proposed model can be trained end-to-end without the need for grids or arrays.

In Chapter 3, we introduced COIN, an alternative to traditional neural compression methods. Instead of directly compressing the RGB values for each pixel of an image, we compress the weights of a neural network overfitted to the image. We showed that such an approach has various compelling properties which could make it an interesting alternative to autoencoder based neural compression.

[2]Note that we minimize over Φ as well since the optimization procedure itself can have learnable parameters as in meta-SGD for example [Li et al., 2017]

In Chapter 4, we introduced COIN++, which addresses several key problems with COIN. More specifically, COIN++ uses meta-learning to reduce compression time by several orders of magnitude, a base network with modulations to encode shared features across datasets, as well as an improved quantization and entropy coding scheme. In addition, we demonstrated that COIN++ can be applied to a wide range of data modalities, including images, audio, medical and climate data.

In Chapter 5, we introduced the functa approach. We used a parameterization similar to COIN++ to build datasets of INRs, which were then used as training data for various deep learning tasks, including generative modeling, classification and data imputation. We showed that it was feasible to train models on datasets of INRs, while also demonstrating why this might be a useful approach, for example for generative modeling of NeRF scenes or 3D shape inference.

In the context of computer vision, we hope that the approach we argued for in this thesis will eventually make generative modeling and inference for large scale (possibly dynamic) 3D scenes tractable. Indeed, to the best of our knowledge, ours is the first approach for building generative models of 3D NeRF scenes that does not require backpropagating through an expensive rendering operation.

In the context of scientific applications of machine learning, we hope that the flexibility of our method will ease the application of deep learning models to non-standard data modalities (particularly fields and densities) and can act as a complimentary approach to current physics inspired models.

In the context of compression, we hope that advances in the training of INRs, combined with more sophisticated entropy coding and quantization may allow COIN-like algorithms to equal or even surpass SOTA codecs, while potentially allowing for compression on currently unexplored modalities.

Figure 6.5 summarizes our approach. Instead of feeding data (usually as an array) to neural networks to produce a certain output, we propose to feed data as functions parameterized by neural networks to the neural network to let it produce a certain output. However, we can also imagine neural networks taking as input neural networks and outputting neural networks. For example, if we are interested

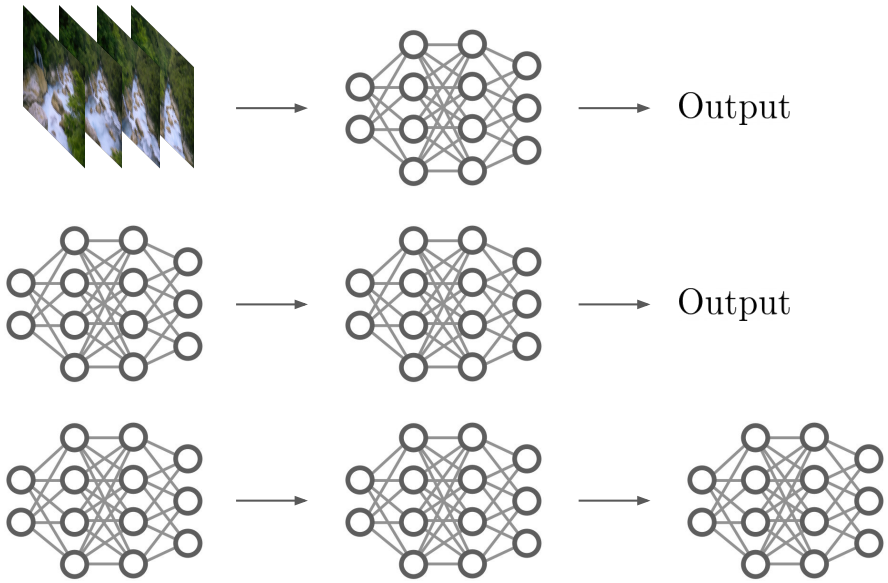

Figure 6.5: 1st row - traditional deep learning, 2nd row - deep learning on functa, 3rd row - neural networks as inputs to neural networks that output neural networks.

in segmenting a dynamic 3D scene, both the input 3D scene and the segmentation map itself could be parameterized by INRs.

However, as described in this chapter and throughout the thesis, there are several open problems and questions around scaling and optimization within this framework. Nonetheless, we are optimistic about the potential of this approach and have outlined a few potential solutions in this chapter. While it remains an open question whether our approach will eventually be applicable to large scale real world problems, we hope the ideas and promising results from this thesis motivate further exploration in this direction.

Bibliography

A. Aaron, Z. Li, M. Manohara, J. De Cock, and D. Ronca. Per-title encode optimization. `https://netflixtechblog.com/per-title-encode-optimization-7e99442b62a2`, 2015. [Online; accessed 26-Feb-2021].

E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. V. Gool. Generative adversarial networks for extreme learned image compression. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 221–231, 2019.

E. Agustsson, D. Minnen, N. Johnston, J. Balle, S. J. Hwang, and G. Toderici. Scale-Space Flow for End-to-End Optimized Video Compression. In *CVPR*, 2020.

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

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

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

M. Arbel and J. Mairal. Amortized implicit differentiation for stochastic bilevel optimization. In *International Conference on Learning Representations*, 2021.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

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

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

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

J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations*, 2017.

J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

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

R. Basri, D. Jacobs, Y. Kasten, and S. Kritchman. The Convergence Rate of Neural Networks for Learned Functions of Different Frequencies. *Advances in Neural Information Processing Systems*, 2019.

J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020.

F. Bellard. Bpg image format. `https://bellard.org/bpg/`, 2014. [Online; accessed 27-Dec-2021].

A. Bietti and J. Mairal. On the inductive bias of neural tangent kernels. *Advances in Neural Information Processing Systems*, 32, 2019.

S. Bond-Taylor and C. G. Willcocks. Gradient origin networks. In *International Conference on Learning Representations*, 2021.

V. Borovitskiy, I. Azangulov, A. Terenin, P. Mostowsky, M. Deisenroth, and N. Durrande. Matérn Gaussian processes on graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 2593–2601, 2021.

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

A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

B. Bross, Y.-K. Wang, Y. Ye, S. Liu, J. Chen, G. J. Sullivan, and J.-R. Ohm. Overview of the versatile video coding (vvc) standard and its applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021.

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

J. Campos, S. Meierhans, A. Djelouah, and C. Schroers. Content Adaptive Optimization for Neural Image Compression. *CVPR Workshop on Learned Image Compression*, 2019.

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

E. R. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. 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*, pages 5799–5809, 2021b.

E. R. Chan, M. Monteiro, P. Kellnhofer, J. Wu, and G. Wetzstein. 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*, pages 5799–5809, 2021c.

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

A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su. Tensorf: Tensorial radiance fields. *arXiv preprint arXiv:2203.09517*, 2022.

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

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

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

Y. Chen, S. Liu, and X. Wang. Learning continuous image representation with local implicit image function. *arXiv preprint arXiv:2012.09161*, 2020a.

Y.-C. Chen, L. Li, L. Yu, A. El Kholy, F. Ahmed, Z. Gan, Y. Cheng, and J. Liu. Uniter: Universal image-text representation learning. In *European conference on computer vision*, pages 104–120. Springer, 2020b.

Z. Chen and H. Zhang. Learning Implicit Fields for Generative Shape Modeling. *CVPR*, 2019.

Y. Cheng, D. Wang, P. Zhou, and T. Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282v9*, 2020a.

Z. Cheng, H. Sun, M. Takeuchi, and J. Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *CVPR*, 2020b.

C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision*, pages 628–644. Springer, 2016.

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

B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.

C. Cremer, X. Li, and D. Duvenaud. Inference Suboptimality in Variational Autoencoders. *ICML*, 2018.

T. Davies, D. Nowrouzezahrai, and A. Jacobson. On the effectiveness of weight-encoded neural implicit 3d shapes. *arXiv preprint arXiv:2009.09808*, 2020.

F. Dellaert and L. Yen-Chen. Neural volume rendering: Nerf and beyond. *arXiv preprint arXiv:2101.05204*, 2020.

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

Domo. Data never sleeps 5.0. `https://www.domo.com/learn/infographic/data-never-sleeps-5`, 2018. [Online; accessed 14-Sep-2021].

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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

E. Dupont. Learning disentangled joint continuous and discrete representations. In *Advances in Neural Information Processing Systems*, pages 710–720, 2018.

E. Dupont and S. Suresha. Probabilistic semantic inpainting with pixel constrained cnns. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2261–2270. PMLR, 2019.

E. Dupont, T. Zhang, P. Tilke, L. Liang, and W. Bailey. Generating realistic geology conditioned on physical measurements with generative adversarial networks. *arXiv preprint arXiv:1802.03065*, 2018.

E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural odes. *Advances in Neural Information Processing Systems*, 32, 2019.

E. Dupont, M. B. Martin, A. Colburn, A. Sankar, J. Susskind, and Q. Shan. Equivariant neural rendering. In *International Conference on Machine Learning*, pages 2761–2770. PMLR, 2020.

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

E. Dupont, H. Kim, S. A. Eslami, D. J. Rezende, and D. Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning*, pages 5694–5725. PMLR, 2022a.

E. Dupont, H. Loya, M. Alizadeh, A. Goliński, Y. W. Teh, and A. Doucet. Coin++: Data agnostic neural compression. *arXiv preprint arXiv:2201.12904*, 2022b.

E. Dupont, Y. W. Teh, and A. Doucet. Generative models as distributions of functions. In *International Conference on Artificial Intelligence and Statistics*, pages 2989–3015. PMLR, 2022c.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. Neural Spline Flows. *arXiv:1906.04032 [cs, stat]*, 2019.

C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios. nflows: normalizing flows in PyTorch, Nov. 2020. URL `https://doi.org/10.5281/zenodo.4296287`.

M. Ehrlich and L. S. Davis. Deep residual learning in the jpeg transform domain. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

T. Elsken, J. H. Metzen, and F. Hutter. Neural Architecture Search: A Survey. *JMLR*, 2019.

S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020.

C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis. Learning SO(3) equivariant representations with spherical CNNs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018.

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

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

S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

M. Garnelo, D. Rosenbaum, C. J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami. Conditional Neural Processes. *arXiv:1807.01613 [cs, stat]*, 2018a.

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

A. Goliński, R. Pourreza, Y. Yang, G. Sautiére, and T. S. Cohen. Feedback recurrent autoencoder for video compression. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

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

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

A. Gropp, L. Yariv, N. Haim, M. Atzmon, and Y. Lipman. Implicit geometric regularization for learning shapes. *arXiv preprint arXiv:2002.10099*, 2020.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

T. Guo, J. Wang, Z. Cui, Y. Feng, Y. Ge, and B. Bai. Variable Rate Image Compression with Content Adaptive Optimization. *CVPR Workshops*, 2020.

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

D. Ha, A. Dai, and Q. V. Le. HyperNetworks. *International Conference on Learning Representations*, 2017.

M. Havasi, R. Peharz, and J. M. Hernández-Lobato. Minimal Random Code Learning: Getting Bits Back from Compressed Model Parameters. *International Conference on Learning Representations*, 2019.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

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

H. Hersbach, B. Bell, P. Berrisford, G. Biavati, A. Horányi, J. Muñoz Sabater, J. Nicolas, C. Peubey, R. Radu, I. Rozum, D. Schepers, A. Simmons, C. Soci, D. Dee, and J.-N. Thépaut. 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.

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

R. D. Hjelm, K. Cho, J. Chung, R. Salakhutdinov, V. Calhoun, and N. Jojic. Iterative Refinement of the Approximate Posterior for Directed Belief Networks. *NeurIPS*, 2016.

J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. *arXiv:2006.11239 [cs, stat]*, 2020.

J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet. Video diffusion models. *arXiv preprint arXiv:2204.03458*, 2022.

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

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

M. J. Hutchinson, C. Le Lan, S. Zaidi, E. Dupont, Y. W. Teh, and H. Kim. Lietransformer: Equivariant self-attention for lie groups. In *International Conference on Machine Learning*, pages 4533–4543. PMLR, 2021.

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

B. Isik, P. A. Chou, S. J. Hwang, N. Johnston, and G. Toderici. Lvac: Learned volumetric attribute compression for point clouds using coordinate based networks. *arXiv preprint arXiv:2111.08988*, 2021.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *CVPR*, 2018.

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

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

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

N. Jean, M. Burke, M. Xie, W. M. Davis, D. B. Lobell, and S. Ermon. Combining satellite imagery and machine learning to predict poverty. *Science*, 353(6301): 790–794, 2016.

K. Jensen, T.-C. Kao, M. Tripodi, and G. Hennequin. Manifold GPLVMs for discovering non-euclidean latent structure in neural data. *Advances in Neural Information Processing Systems*, 2020.

S. Jha, D. Gong, X. Wang, R. E. Turner, and L. Yao. The neural process family: Survey, applications and perspectives. *arXiv preprint arXiv:2209.00517*, 2022.

K. Ji, J. Yang, and Y. Liang. Bilevel optimization: Convergence analysis and enhanced design. In *International Conference on Machine Learning*, pages 4882–4892. PMLR, 2021.

C. Jiang, A. Sud, A. Makadia, J. Huang, M. Nießner, T. Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

I. T. Jolliffe. *Principal component analysis for special types of data*. Springer, 2002.

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

A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

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

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

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

P. Kidger. On neural differential equations. *arXiv preprint arXiv:2202.02435*, 2022.

P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33: 6696–6707, 2020.

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

Y. Kim, S. Wiseman, A. C. Miller, D. Sontag, and A. M. Rush. Semi-Amortized Variational Autoencoders. *ICML*, 2018.

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

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

W. B. Kleijn, F. S. Lim, A. Luebs, J. Skoglund, F. Stimberg, Q. Wang, and T. C. Walters. Wavenet based low rate speech coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 676–680. IEEE, 2018.

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

F. Knoll, J. Zbontar, A. Sriram, M. J. Muckley, M. Bruno, A. Defazio, M. Parente, K. J. Geras, J. Katsnelson, H. Chandarana, et al. fastmri: A publicly available raw k-space and dicom dataset of knee images for accelerated mr image reconstruction using machine learning. *Radiology: Artificial Intelligence*, 2(1):e190007, 2020.

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

Kodak. Kodak Dataset. `http://r0k.us/graphics/kodak/`, 1991.

A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, and D. J. Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *International Conference on Machine Learning*, pages 5742–5752. PMLR, 2021.

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arxiv preprint arxiv:1806.08342*, 2018a.

R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018b.

R. G. Krishnan, D. Liang, and M. Hoffman. On the challenges of learning with inference networks on sparse, high-dimensional data. *AISTATS*, 2018.

J. Lee, S. Cho, and S.-K. Beack. Context-adaptive entropy model for end-to-end optimized image compression. In *International Conference on Learning Representations*, 2019a.

J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pages 3744–3753. PMLR, 2019b.

J. Lee, J. Tack, N. Lee, and J. Shin. Meta-learning sparse implicit neural representations. *Advances in Neural Information Processing Systems*, 34, 2021.

Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. PointCNN: Convolution on x-transformed points. *Advances in Neural Information Processing Systems*, 31: 820–830, 2018.

Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. BRECQ: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2021a.

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

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Z. Li, S. Niklaus, N. Snavely, and O. Wang. 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*, pages 6498–6508, 2021b.

R. Liang, H. Sun, and N. Vijaykumar. Coordx: Accelerating implicit neural representation with a split mlp architecture. In *International Conference on Learning Representations*, 2021a.

T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *arXiv preprint arXiv:2101.09671*, 2021b.

C. H. Lin, C.-C. Chang, Y.-S. Chen, D.-C. Juan, W. Wei, and H.-T. Chen. Cocogan: Generation by parts via conditional coordinating. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4512–4521, 2019.

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

C. Louizos, K. Ullrich, and M. Welling. Bayesian Compression for Deep Learning. *Advances in Neural Information Processing Systems*, 2017.

G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

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

L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021a.

Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive neural representations of volumetric scalar fields. In *Computer Graphics Forum*, volume 40, pages 135–146. Wiley Online Library, 2021b.

S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2019.

J. Marino, Y. Yue, and S. Mandt. Iterative Amortized Inference. *ICML*, 2018.

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

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

I. Mehta, M. Gharbi, C. Barnes, E. Shechtman, R. Ramamoorthi, and M. Chandraker. Modulated periodic activations for generalizable local functional representations. *arXiv preprint arXiv:2104.03960*, 2021.

F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4394–4402, 2018.

F. Mentzer, G. Toderici, M. Tschannen, and E. Agustsson. High-fidelity generative image compression. *arXiv preprint arXiv:2006.09965*, 2020.

L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning*, pages 3481–3490, 2018.

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

L. M. Mescheder. *Stability and Expressiveness of Deep Generative Models*. PhD thesis, Universität Tübingen, 2020.

B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*, 2020.

D. Minnen, J. Ballé, and G. Toderici. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In *Advances in Neural Information Processing Systems*, 2018.

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

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

MP3. MP3 codec. `https://www.iso.org/standard/22412.html`, 1993.

M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. *arXiv preprint arXiv:1906.04721*, 2019.

C. Nash, J. Menick, S. Dieleman, and P. Battaglia. Generating images with sparse representations. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7958–7968. PMLR, 18–24 Jul 2021a.

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

Netflix. Netflix data consumption. `https://help.netflix.com/en/node/87`, 2022. [Online; accessed 10-Aug-2022].

T. Nguyen-Phuoc, C. Li, S. Balaban, and Y.-L. Yang. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. *Advances in Neural Information Processing Systems*, 2018.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

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

M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Occupancy Flow: 4D Reconstruction by Learning Particle Dynamics. *ICCV*, 2019.

M. Niemeyer, L. Mescheder, M. Oechsle, and A. Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, 2019.

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

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

A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10318–10327, 2021.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on Computer Vision and Pattern Eecognition*, pages 652–660, 2017.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

A. Rajeswaran, C. Finn, S. Kakade, and S. Levine. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 2019.

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

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

J. Ramirez and J. Gallego-Posada. Loonie: Compressing coins with l0-constraints. *arXiv preprint arXiv:2207.04144*, 2022.

J. O. Ramsay and C. Dalzell. Some tools for functional data analysis. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):539–561, 1991.

N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

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

C. Reiser, S. Peng, Y. Liao, and A. Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

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

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*, 2014.

J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162, 1979.

D. W. Romero, R.-J. Bruintjes, J. M. Tomczak, E. J. Bekkers, M. Hoogendoorn, and J. C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*, 2021a.

D. W. Romero, A. Kuzina, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021b.

B. Ronen, D. Jacobs, Y. Kasten, and S. Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019.

K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann. Stabilizing training of generative adversarial networks through regularization. In *Advances in Neural Information Processing Systems*, pages 2018–2028, 2017.

Y. Rubanova, R. T. Chen, and D. K. Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. *Advances in neural information processing systems*, 32, 2019.

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

J. R. Schwarz and Y. W. Teh. Meta-learning sparse compression networks. *arXiv preprint arXiv:2205.08957*, 2022.

K. Schwarz, Y. Liao, M. Niemeyer, and A. Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020.

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

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

V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein, and M. Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019a.

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

V. Sitzmann, E. R. Chan, R. Tucker, N. Snavely, and G. Wetzstein. MetaSDF: Meta-learning Signed Distance Functions. In *Advances in Neural Information Processing Systems*, 2020a.

V. Sitzmann, J. N. P. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit Neural Representations with Periodic Activation Functions. In *Advances in Neural Information Processing Systems*, 2020b.

A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, 2001.

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

I. Skorokhodov, S. Tulyakov, and M. Elhoseiny. Stylegan-v: A continuous video generator with the price, image quality and perks of stylegan2. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3626–3636, 2022.

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

C. K. Sønderby, L. Espeholt, J. Heek, M. Dehghani, A. Oliver, T. Salimans, S. Agrawal, J. Hickey, and N. Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *arXiv preprint arXiv:2003.12140*, 2020.

P. Spurek, S. Winczowski, J. Tabor, M. Zamorski, M. Zięba, and T. Trzciński. Hypernetwork approach to generating point clouds. *arXiv preprint arXiv:2003.00802*, 2020.

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

K. Stelzner, K. Kersting, and A. R. Kosiorek. Generative adversarial set transformers. In *Workshop on Object-Oriented Learning at ICML*, volume 2020, 2020.

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

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

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

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

L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6411–6420, 2019.

D. Tran, K. Vafa, K. Agrawal, L. Dinh, and B. Poole. Discrete flows: Invertible generative models of discrete data. *Advances in Neural Information Processing Systems*, 32, 2019.

A. Trevithick and B. Yang. GRF: Learning a General Radiance Field for 3D Scene Representation and Rendering. *arXiv:2010.04595 [cs]*, 2020.

K. Ullrich, E. Meeds, and M. Welling. Soft Weight-Sharing for Neural Network Compression. *International Conference on Learning Representations*, 2017.

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

B. Uria, I. Murray, and H. Larochelle. Rnade: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems*, 26, 2013.

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

J.-M. Valin and J. Skoglund. LPCNet: Improving neural speech synthesis through linear prediction. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5891–5895. IEEE, 2019.

M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling. Bayesian Bits: Unifying Quantization and Pruning. *NeurIPS*, 2020.

A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

T. van Rozendaal, I. A. M. Huijben, and T. S. Cohen. Overfitting for Fun and Profit: Instance-Adaptive Data Compression. *International Conference on Learning Representations*, 2021.

F. Vasconcelos, B. He, N. Singh, and Y. W. Teh. Uncertainr: Uncertainty quantification of end-to-end implicit neural representations for computed tomography. *arXiv preprint arXiv:2202.10847*, 2022.

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

G. K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992.

J.-L. Wang, J.-M. Chiou, and H.-G. Müller. Functional data analysis. *Annual Review of Statistics and its application*, 3:257–295, 2016.

X. Wang, L. Gao, P. Wang, X. Sun, and X. Liu. Two-stream 3-d convnet fusion for action recognition in videos with arbitrary size and length. *IEEE Transactions on Multimedia*, 20(3):634–644, 2017.

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

W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical Science*, 9(2):513–530, 2018.

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

Y. Xie, K. L. Cheng, and Q. Chen. Enhanced invertible encoding for learned image compression. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 162–170, 2021.

Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022.

T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman. Video enhancement with task-oriented flow. *International Journal of Computer Vision*, 127(8):1106–1125, 2019.

Y. Yang, G. Sautiére, J. J. Ryu, and T. S. Cohen. Feedback Recurrent AutoEncoder. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2019.

Y. Yang, R. Bamler, and S. Mandt. Improving Inference for Neural Image Compression. *NeurIPS*, 2020.

Y. Yang, S. Mandt, and L. Theis. An introduction to neural data compression. *arXiv preprint arXiv:2202.06533*, 2022.

K. Yosida. *Functional analysis*. Springer Science & Business Media, 2012.

A. Yu, V. Ye, M. Tancik, and A. Kanazawa. pixelNeRF: Neural Radiance Fields from One or Few Images. *arXiv:2012.02190 [cs]*, 2020.

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

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3391–3401, 2017.

J. Zbontar, F. Knoll, A. Sriram, T. Murrell, Z. Huang, M. J. Muckley, A. Defazio, R. Stern, P. Johnson, M. Bruno, et al. fastmri: An open dataset and benchmarks for accelerated mri. *arXiv preprint arXiv:1811.08839*, 2018.

N. Zeghidour, A. Luebs, A. Omran, J. Skoglund, and M. Tagliasacchi. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2021.

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

Y. Zhou, X. Sun, C. Luo, Z.-J. Zha, and W. Zeng. Spatiotemporal fusion in 3d cnns: A probabilistic view. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9829–9838, 2020a.

Y. Zhou, C. Wu, Z. Li, C. Cao, Y. Ye, J. Saragih, H. Li, and Y. Sheikh. Fully convolutional mesh autoencoder using efficient spatially varying kernels. *Advances in Neural Information Processing Systems*, 33:9251–9262, 2020b.

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