Self–Similarity Priors: Neural Collages as Differentiable Fractal Representations

Michael Poli[*]
Stanford University
DiffeqML

Winnie Xu[*]
University of Toronto

Stefano Massaroli[*]
Mila
DiffeqML

Chenlin Meng
Stanford University

Kuno Kim
Stanford University

Stefano Ermon
Stanford University
CZ Biohub

Abstract

Many patterns in nature exhibit *self–similarity*: they can be compactly described via self–referential transformations. Said patterns commonly appear in natural and artificial objects, such as molecules, shorelines, galaxies and even images. In this work, we investigate the role of learning in the automated discovery of self-similarity and in its utilization for downstream tasks. To this end, we design a novel class of implicit neural operators, Neural `Collages`, which (1) represent data as the parameters of a self–referential, structured transformation, and (2) employ hypernetworks to amortize the cost of finding these parameters to a single forward pass. We detail how to leverage the representations produced by Neural `Collages` in various tasks, including data compression and generation. Neural `Collage` image compressors are orders of magnitude faster than other self–similarity–based algorithms during encoding and offer compression rates competitive with implicit methods. Finally, we showcase applications of Neural `Collages` in fractal art and as deep generative models.

1 Introduction

Given a specified image, can one come up with a dynamical system with it as its attractor? (Welstead, 1999)

Scientific fields are underpinned by a search for structure. Geometry, sparsity and invariances, when appropriately introduced in a mechanistic model, allow us to concisely describe phenomena. To this end, machine learning has been introduced as a means to fix partial priors in a model, and discover the rest through data (Rackauckas et al., 2020; Dao et al., 2020; Bronstein et al., 2021). In general, the notion of structure is also essential for compression: through a suitable choice of language, one can explain phenomena in fewer symbols, yielding shorter representations of the observables (Tishby et al., 2000; Lee et al., 2007). Objects exhibiting *self–similarity* structure are composed of patterns that appear similar to themselves at multiple scales, as shown in Figure 1. This type of structure frequently appears in nature, at different degrees: shorelines, molecules, plants, turbulent flows and basins of attraction of dynamical systems all display elements of self–similarity (Mandelbrot and Mandelbrot, 1982; Song et al., 2005; Vulpiani et al., 2009; Barnsley, 2014). In this work, we explore the role of learning in the automatic discovery of *self–similarity* structure in data, and how it can serve as an inductive bias in machine learning models.

The mathematical embodiment of this idea is found in fractal patterns, which often arise by characterizing limit sets of nonlinear maps e.g. iterations of complex numbers for Julia and Mandelbrot

[*]Equal contribution authors. Contact email: `poli@stanford.edu`.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

sets (Julia, 1918; Mandelbrot, 1980). Fractals are scale–invariant: they can be "zoomed in" by increasing the resolution of the limit sets, and manifest arbitrarily similar patterns at different scales. Despite their apparent infinite complexity, a fractal can be uniquely and compactly described by its generating nonlinear map.

A method to discover self–similar structure in data (not necessarily of fractal nature) can then be formalized as an optimization problem: after choosing an appropriate class of contractive, parameterized maps, one searches for parameters such that a given data point can be (approximately) recovered as the (unique) fixed–point of the chosen map. This approach, pioneered in (Barnsley et al., 1986), paved the way for one of the most successful algorithmic applications of self–similarity, *fractal image compression* (Jacquin et al., 1992; Jacquin, 1993; Barnsley et al., 1996; Welstead, 1999; Fisher, 2012). First, an encoding step carries out a search to solve the inverse problem of data to operator parameters via extensive search, or by restricting the class of operators such that a closed–form solution may be found. Then, given the parameters, the decoding step solves for the fixed–point of the operator, corresponding to a corrupted version of the original data. The quality of decoded images i.e. the *loss* of the fractal compression method is directly tied with the expressivity of the class of operators considered, which are often designed to seek self–similarities in pixel space. Yet, larger classes induce more challenging optimization problems, leading to long encoding times.

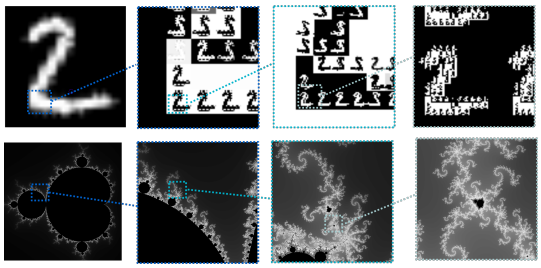


Figure 1: **[Top]** MNIST digit *fractalized* via a `Collage`. The image is represented as the coefficients of a `Collage`, and decoded as its attractor. Magnification is done by decoding at higher resolutions. **[Bottom]** The Mandelbrot set (Mandelbrot, 1980), an example of a fractal displaying self–similarity.

Here, we propose a novel learning–based technique to extract and utilize self–similar representations of data. We develop Neural `Collages`, a family of differentiable, parametrized operators structured to capture self–similarity between partitions of data. The inverse problem of Neural `Collages` is solved with a single forward pass of a hypernetwork (Ha et al., 2016) trained to generate a set of parameters as the fractal code. This amortized approach is orders of magnitude faster than traditional search based methods. `Collage` operators are composable with neural network architectures and have wide applicability beyond compression, as they can be optimized end–to–end for a variety of tasks including generative modeling.

In data compression, Neural `Collages` preserve advantages of fractal compression methods, with up to $10\times$ (accounting for training time) and $100\times$ (at test time) speedups during encoding. Further, we investigate deep generative models based on Neural `Collages`, where `Collage` parameters assume the role of latent variables of a hierarchical *variational autoencoder* (VAE) (Kingma and Welling, 2013). `Collage` VAEs can sample at resolutions unseen during training by decoding at higher resolutions through their `Collage` operator, revealing additional detail over upsampling via interpolation. Finally, we showcase applications for fractal art, where an image can be "fractalized" i.e. reconstructed as a collage of smaller copies of itself at different scales (see Figure 1, top).

2 Problem Setting

We start with an example related to a core idea behind fractal compression and Neural Collages: representing data as the parameters of a function that converges to it.

2.1 Representing Data as a Fixed–Point of an Iterated Function

Consider a value x that happens to be of interest for a downstream task, for example regression. Rather than performing computation on it directly, we can use the parameters ω of a contractive map with x as its fixed-point, for example a simple affine function with parameters $\omega := (a, b)$, $a \in \mathbb{R}, b \in \mathbb{R}$:

$$x_{k+1} = ax_k + b, \quad |a| < 1$$

Given ω, the value x can be recovered by repeatedly applying the map starting from any other initial condition, that is

$$x^* : x^* = ax^* + b.$$

Parameters ω can then replace x as an alternative representation, using two values (a, b) instead of x.

Other classes of functions, if chosen appropriately, can represent data with *less* parameters than its dimensionality. In general, encoding through a function is a lossy process: the original data is recovered up to a degree of accuracy. The insight behind fractal compression and Neural `Collages` is to view data as *sets*, and define iterated functions as ensembles of simple scalar linear functions acting on different subsets. Below, we introduce background required to make these notions precise.

2.2 Fractal Compression Background

We proceed by formalizing *fractal data encoding* as an optimization problem (Barnsley and Demko, 1985; Fisher, 2012; Barnsley, 2014).

Let $(\mathbb{X}, d)$ be a complete metric space and $(\mathcal{H}(\mathbb{X}), d_{\mathcal{H}})$ its corresponding Haussdorff metric space, i.e. $\mathcal{H}(\mathbb{X}) = \{\mathbb{A} \subset \mathbb{X} : \mathbb{A} \text{ is compact}\}$. We represent a data point as a set $\mathbb{S} \in \mathcal{H}(\mathbb{X})$. This choice of space supports an data modality–agnostic treatment of fractal data encoding. A concrete realization is discussed for image domains in Section 3. Results on metric spaces are provided as reference in Appendix A.

Example 1. *Consider binary images on a square domain. Then, $\mathbb{X}$ is a finite compact subset of $\mathbb{R}^2$. An image $\mathbb{S}$ is the finite set of coordinates of either black or white pixels. Further, each image corresponds to a point in $\mathcal{H}(\mathbb{X})$.*

Let $\{f_1, f_2, \ldots, f_K\}$ be a collection of maps on $\mathbb{X}$, $f_k : \mathbb{X} \to \mathbb{X}$. This is colloquially referred to as *iterated function system* (IFS) (Barnsley, 2014). We can then define a map $F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$ by

$$F(\mathbb{A}) = \bigcup_{k=1}^{K} f_k(\mathbb{A}) \quad \forall \mathbb{A} \in \mathcal{H}(\mathbb{X})$$

where $f_k(\mathbb{A})$ is intended as $f_k(\mathbb{A}) = \{f_k(a) : a \in \mathbb{A}\}$. An interpretation is the following: F produces as output a composition, or *collage*, of transformations applied to a subset $\mathbb{A}$ of $\mathbb{X}$.

2.3 The Inverse Problem: Data to IFS

Given data $\mathbb{S}$, can we find a map F with $\mathbb{S}$ as its fixed point? This can be achieved by identifying a collection of maps $f_k : \mathbb{X} \to \mathbb{X}$ such that the following conditions hold

$$i.\ \ F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X}); \mathbb{A} \mapsto \bigcup_{k=1}^{K} f_k(\mathbb{A}) \text{ is contractive;}$$

$$ii.\ \ \mathbb{S} \text{ is } \underline{\text{the}} \text{ fixed point of } F,\ \mathbb{S} = F(\mathbb{S}) = \bigcup_{k=1}^{K} f_k(\mathbb{S});$$

Note that, F is contractive w.r.t the Hausdorff metric with Lipsichitz constant $L < 1$ iff all the maps f_k are contractive w.r.t d with constant $\ell_k < 1$. In such a case it holds $L = \max_k\{\ell_k\}$ and F admits a unique fixed point. A classical result provides an indirect method to find such F.

Theorem 1 (Collage Theorem (CT) (Barnsley and Demko, 1985))**.** *Let $(\mathbb{X}, d)$ be a complete metric space and let $f : \mathbb{X} \to \mathbb{X}$ be a ℓ–Lipschitz contractive map with fixed point $x^* \in \mathbb{X}$. Then,*

$$d(x, x^*) \leq \frac{1}{1-\ell} d(x, f(x)) \tag{2.1}$$

By applying the CT directly to F using the Hausdorff metric $d_{\mathcal{H}}$, we have $d_{\mathcal{H}}(\mathbb{S}, \mathbb{A}^*) \leq \frac{1}{1-L} d_{\mathcal{H}}\left(\mathbb{S}, \bigcup_{k=1}^{K} f_k(\mathbb{S})\right)$. This means that we can upper bound the distance $d_{\mathcal{H}}(\mathbb{S}, \mathbb{A}^*)$ between data $\mathbb{S}$ and attractor $\mathbb{A}^*$ of F via $d(\mathbb{S}, F(\mathbb{S}))$, which requires a single application of F and is thus cheaper to evaluate. Even if it is not possible to stitch together transformed copies $f_k(\mathbb{S})$ to perfectly reconstruct the data $\mathbb{S}$, i.e. $d_{\mathcal{H}}\left(\mathbb{S}, \bigcup_{k=1}^{K} f_k(\mathbb{S})\right) \neq 0 \quad (\Leftrightarrow \mathbb{S} \neq F(\mathbb{S}))$, a smaller IFS Lipschitz constant L of the IFS implies a lower distance between the data $\mathbb{S}$ and the attractor $\mathbb{A}^*$ of F, given a mismatch $d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S}))$. This, in turn, implicitly promotes the use of "very contractive" maps f_k (i.e. with low ℓ_k). We refer to the procedure of searching for an F that minimizes the r.h.s of the CT bound as the *fractal data encoding* problem.

A learning perspective of fractal data encoding Fractal data encoding problem can be translated into finding a parametric representation $f_k(\,\cdot\,; w_k)$, $w \in \mathbb{R}^{n_w}$ for functions $f_k(\cdot)$ (e.g. neural networks with parameters w_k) where $w = (w_1, \ldots, w_K) \in \mathbb{W}$ are optimized to minimize a Hausdorff metric loss function $d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S}; w))$ naturally induced by the CT,

$$\min_{w \in \mathbb{W}} \; d_{\mathcal{H}}(\mathbb{S}, \textstyle\bigcup_{k=1}^{K} f_k(\mathbb{S}; w_k)) \tag{2.2}$$

To decode the data encoded in parameters w of F encodes in one samples any initial condition $\mathbb{A}_0$, and iterates $\mathbb{A}_{t+1} = F(\mathbb{A}_t)$ until convergence to $\mathbb{A}^* \approx \mathbb{S}$.

IFS, PIFS and Beyond *Partitioned iterated function systems* (PIFS) (Jacquin et al., 1992) are a generalization of IFSs that can capture localized self–similarity by allowing each domain of a contraction map f_k to be a different subset $\mathbb{A}_k \subset \mathbb{S}$. his introduces a significant challenge in 2.2: the optimization problem need now determine optimal (as measured by $d_{\mathcal{H}}$) domains $\mathbb{A}_k$ for each f_k by searching across all possible subsets of $\mathbb{S}$, yielding an exploding combinatorial problem.

Solving for affine IFS As with traditional approximation problems, there is a tension in the objective of fractal data encoding between the "expressiveness" of the class of functions, and the tractability of the optimization problem. The solution w of (2.2) is an equivalent representation for $\mathbb{S}$ (up to $d_{\mathcal{H}}(\mathbb{S}, \mathbb{A}^*)$).

Existing methods based on the idea of (Barnsley and Demko, 1985) resolve this tension by considering (a) compression as a task, such that w should be encodeable in the least number of bits possible and (b) affine functions $f_k(x; w_k) = a_k x + b_k$. With these choices, a solution to (2.2) can be found in closed–form (Fisher, 2012), and the parametrization w results compact enough to be a valid compression code – only two floats for each f_k in F, i.e. $w_k = (a_k, b_k) \in \mathbb{R}^2$.

There are a number of limitations we aim to address:

- Fractal data encoding is only considered as an intermediate step towards compression. We develop a learning–based approach to the solution of (2.2) for tasks beyond compression.
- Solving (2.2) on a collection of data as per (Fisher, 2012) is computationally expensive. We directly solve a collection of fractal data encoding problems in parallel via hypernetworks (Ha et al., 2016), effectively amortizing the cost.
- As noted by (Welstead, 1999; Fisher, 2012), for an (affine) IFS to provide a satisfactory solution to (2.2), the self-similarity property has to be global across the set $\mathbb{S}$. That is, the entire set $\mathbb{S}$ is made up of smaller copies of itself, or a part of itself, property that is rather rare in natural data: indeed, most images are only self–similar to a degree. To alleviate these restrictions, we develop `Collage` operators, a generalization of IFSs which can be broadly categorized as a soft–*partitioned iterated function system* (PIFS) (Jacquin et al., 1992).

3 Neural `Collages`

Moving forward, we treat Neural `Collages` algebraically. This allows us to discuss in detail the properties of a Neural `Collage`, including algorithmic differences with other iterated function systems. To do so we consider, instead of generic sets, data that can be expressed as simple ordered sets: in other words, as vectors. Images will be our recurring example, with the understanding that the entire discussion can readily be adapted to other modalities e.g. sequences.

From generic sets to vectors Following the PIFS treatment of Øien and Lepsøy (1995), we focus our analysis on `Collage` composed of *affine* maps, operating on the space of discrete images of a given resolution with a total number m of pixels each taking values in $\mathbb{R}$. Pixels of different channels are treated without loss of generality as different elements. This allows us to collect all m pixel values in an *ordered*[2] vector $z \in \mathbb{R}^m$.

A type of subsets on images involves the formation of square patches. Let us assume that each image is partitioned into (1) K non–overlapping *range cells* and (2) N possibly–overlapping *domain cells*.

[2] with a specific predefined criterion, e.g. row–major ordering.

A range cell R_k is then of size $n_{r,k} \times n_{r,k}$, such that $m = \sum_{k=1}^{K} n_{r,k}^2$, and domain cells D_n are of size $n_{d,n} \times n_{d,n}$. With such coordinatization, the (affine) fixed–point map F reduces to a structured affine contraction on $\mathbb{R}^m$.

Definition 1 (Neural `Collage` Operator)**.** *Consider a m-pixel image represented by the ordered vector $z \in \mathbb{R}^m$. Then, a Collage Operator is defined as the parametric linear map:*

$$F(z; w) = \sum_{k,n} \gamma_{k,n} a_{k,n} T_k P_{k,n} S_n z + \sum_{k,n} \gamma_{k,n} b_{k,n} T_k \mathbb{1} \tag{3.1}$$

- $S_n \in \mathbb{R}^{n_{d,n}^2 \times m}$ ***selects*** *a domain cell D_n of $n_{d,n} \times n_{d,n}$ pixels.*
- $P_{k,n} \in \mathbb{R}^{n_{r,k}^2 \times n_{d,n}^2}$ *is a* ***pooling*** *operator that shrinks the domain cell D_k into the size of the corresponding range cell R_k, i.e. from $n_{d,n} \times n_{d,n}$ to $n_{r,k} \times n_{r,k}$ pixels;*
- $T_k \in \mathbb{R}^{m \times n_{r,k}^2}$ ***positions*** *the pooled domain cell in the correct range cell location and zeroes out the rest;*
- $a_{k,n}, b_{k,n} \in \mathbb{R}$ ***scales*** *and* ***translates*** *the value in each pixel of the pooled domain cell, respectively.*
- $\gamma_{k,n} \in \mathbb{R}$*; convex* ***combination*** *of affine outputs produced from all N domains.*

The parameters w is the collection of all $a_{k,n}, b_{k,n}$ and the mixing weights $\gamma_{k,n}$.

`Collage` operators represent each range cell R_k as a convex combination of pooled and scaled versions of all domain cells D_n translated block–wise by b_n. On individual range cells comprising the output, a symbolic representation can be given as

$$\mathsf{R}_k = \sum_n \gamma_{k,n} a_{k,n} \mathsf{D}_n + \sum_n \gamma_{k,n} b_{k,n}. \tag{3.2}$$

In the generic set formulation, these maps correspond to functions f_k of F. Differently from a PIFS, each f_k <u>does not</u> act on a different subset (domain cell) to produce R_k. Instead, all f_k aggregate affine transformations – parametrized by $a_{k,n}, b_{k,n}$ – on domains via $\gamma_{k,n}$. However, each f_k is equipped with different mixing weights and different affine maps. Hence, a `Collage` in this form can be seen as a soft–PIFS.

Introducing auxiliary domains A `Collage` step maps mixtures of all domains to each range, and assembles the ranges into its output. As Neural `Collages` are often optimized on datasets, rather than single data points, we posit that improvements in the expressiveness can be readily achieved by mixing additional dataset–level information through *auxiliary domains* U_v.

Definition 2 (`Collage` operator with auxiliary domains)**.**

$$\hat{\mathsf{R}}_k = \mathsf{R}_k + \sum_{v=1}^{V} \gamma_{k,v} a_{k,v} \mathsf{U}_v$$

In coordinates this translates to

$$\hat{F}(z, u; w) = F_w(z; w) + \sum_{k=1}^{K} \sum_{v=1}^{V} \gamma_{k,v} a_{k,v} T_k P_{k,v} S_v u$$

where $F_w(z)$ is (3.1) and R_k is (3.2), and the parameters w include the coefficients $\gamma_{k,v}, a_{k,v}$ of the auxiliary domains U_v, as well as $\gamma_{k,n}, a_{k,n}, b_{k,n}$.

We consider different variants of U_v, including: deterministic transformations of domain cells D_n e.g. rotations as per (Jacquin et al., 1992), learned cells directly parametrized and optimized for an objective, similar to feature maps in (Jaegle et al., 2021), and U_v produced by a neural network encoder. Specifics are provided in Section 4. A schematic of a single step of `Collage` is given in Fig. 3.

3.1 The Forward Problem: Collage to Data

Given a parametrization w for the `Collage` operator F_w and an initial image z_0, the attractor $z^* \,:\, z^* = F_w(z^*)$ can be recovered by iterating the fixed–point map

$$z_{t+1} = F(z; w) = A(w)z_t + b(w)$$

assuming F to be a contraction w.r.t. the standard Euclidean metric on $\mathbb{R}^m$. This can be ensured by an appropriate choice of the coefficients $a_{k,n}, \gamma_{k,n}$. In particular, if all the mixing weights are such that $\sum_{n=1}^{N} \gamma_{k,n} = 1$ and $|a_{k,n}| < 1$, then contractivity of the collage operator follows as in standard PIFS (see e.g. Fisher (2012)).

Note that the attractor of a `Collage` can be also computed in closed–form as $z^* = [\mathbb{I} - A(w)]^{-1} b(w)$. A similar discussion follows for a general `Collage` with auxiliary domains. Note that auxiliary domains U across iterations t are to be chosen such that the sequence $\{\mathsf{U}_t\}_{t=0}^{\infty}$ converges e.g. constant functions.

Pseudocode for a single step of a `Collage` is shown below. Figure 2 provides a visualization of the convergence of a `Collage` to its fixed–point (after repeated application of the operator).

```
def collage_operator(self, z, collage_weight, collage_bias):
    """Collage Operator (decoding). Performs the steps described in  Def. 3.1, Figure 2."""
    # Split the current iterate `z` into source patches according to the partitioning scheme.
    domains = img_to_patches(z)
    # Pool domains (pre augmentation) to range patch sizes.
    pooled_domains = pool(domains)
    # If needed, produce additional candidate source patches as augmentations of existing
    # domains, or concatenate auxiliary patches parametrized and optimized directly.
    if self.n_aug_transforms > 1:
        pooled_domains = self.generate_candidates(pooled_domains)
    pooled_domains = repeat(pooled_domains, 'b c d h w -> b c d r h w', r=self.num_ranges)
    # Apply the affine maps to source patches
    range_domains = einsum('bcdrhw, bcdr -> bcrhw', pooled_domains, collage_weight)
    range_domains = range_domains + collage_bias[..., None, None]
    # Reconstruct data by composing the output patches back together.
    z = patches_to_img(range_domains)
    return z
```

Decoding at higher resolutions A `Collage` can be applied, without change, to images of different resolutions. Consider scaling the resolution by a factor $s > 1,\ (s \in \mathbb{N})$. Then the *magnified* image representation is made up of s^2 images $z^i \in \mathbb{R}^m$. The `Collage` operator can then be thought to act singularly on each z^i obtaining a forward fixed–point iteration

$$z^i_{t+1} = \sum_{k,n} \gamma^i_{k,n} a^i_{k,n} T_k P_{k,n} S_n z^i_t + \sum_{k,n} \gamma^i_{k,n} b^i_{k,n} T_k \mathbb{1}.$$

When solving by unrolling the fixed–point iteration, the second term can be precomputed as it does not depend on z_t.

3.2 Amortized Solution of the Inverse Problem

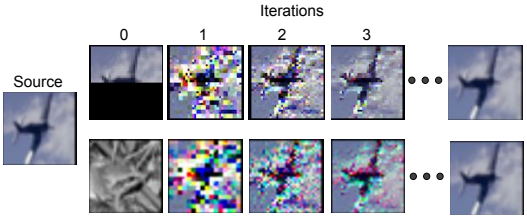


Figure 2: `Collage` forward problem: given w, different initial conditions decoded to the same z^*.

Although the CT suggests a constructive procedure via (2.2) to find a valid fractal representation w, there are no guidelines in case other objectives are of interest. Further, the class of PIFS – without modifications – does not lend itself well to numerical optimization as it involves the combinatorial problem of matching domains to ranges. With their soft aggregation, Neural `Collages` can instead be used for task–based optimization. Given an input image $x \in \mathbb{R}^m$, we can optimize the parameters w (and pixel values u of the auxiliary domains) of the Neural `Collage` operator to minimize an objective $J_\omega(x, z^*(w, u))$ by solving a nonlinear program $\min_{w,u} J_w(x, z^*(w, u))$ with $z^*(w, u)$ obtained by the fixed point iteration on $\hat{F}(z, u; w)$. Choosing J_w to be a reconstruction objective yields a problem similar to fractal data encoding (2.2).

Suppose instead to be given an image dataset whose distribution p is known only through i.i.d. samples $x \in \mathbb{R}^m$, $x \sim p(x)$. In this context, the fractal data encoding problem in standard form needs to be solved for each sample x. Instead, we introduce an *hypernetwork* (Ha et al., 2016) $\mathcal{E}$ with weights θ, generating a fractal code w given an input image x, i.e. parametrizing a map $x \mapsto w_\theta(x)$,

$$\forall x \sim p(x) \quad w_\theta(x) = \mathcal{E}(x;\theta)$$

The hypernetwork is trained to solve the following empirical risk minimization problem, effectively amortizing the cost over the full dataset:

$$\begin{aligned} \min_{\theta,u} \quad & \mathbb{E}_{x\sim p(x)}\left[J(x, z^*(\theta,u))\right] \\ \text{subject to} \quad & w_\theta(x) = \mathcal{E}(x;\theta) \\ & z^*(\theta,u) = \hat{F}(z^*(\theta,u), u; w_\theta(x)) \\ & u \in \mathbb{U}. \end{aligned} \tag{3.3}$$

4 `Collages` in Learning Tasks

We showcase applications of Neural `Collages` as decoders for deep generative models, as neural compressors for images and as a method to generate fractal art. All variants of the model share a common structure, outlined in Figure 3. The code is available at `github.com/ermongroup/self-similarity-prior`.

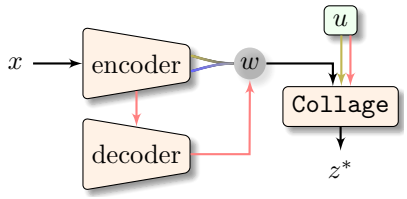


Figure 3: Schematic of a Neural `Collage`. In yellow, computation unique to the compressor variant; x is deterministically encoded into `Collage` parameters w, then used to decode. Similarly, blue indicates computation done by the fractalizer, and red by the generative variant, where w takes the role of a latent variable.

4.1 Generative Neural `Collages`

We investigate application of Neural `Collages` as deep generative models for distributions $p(x)$ of images. In this context, w assume the role of latent variables. In particular, we consider a hierarchical *variational autoencoder* (VDVAE) (Kingma and Welling, 2013; Child, 2020) model based on Neural `Collages`.

Magnifying samples via `Collage` VAEs VAE models seek to data x into a latent representation w such that the following lower bound (`ELBO`) on data log–likelihood be maximized

$$\mathbb{E}_{q_\phi}\underbrace{[\log p_\theta(x|w)]}_{-\text{distortion}} - \underbrace{D_{KL}[q_\phi(w|x)||p_\phi(w)]}_{\textit{rate}} \leq \log p(x;\theta,\phi)$$

where approximate posterior $q_\phi(w|x)$, prior $p_\phi(w)$, and generator $p_\theta(x|w)$ are implemented as neural networks. A multiplicative hyperparameter β is often introduce to control the relative weight between

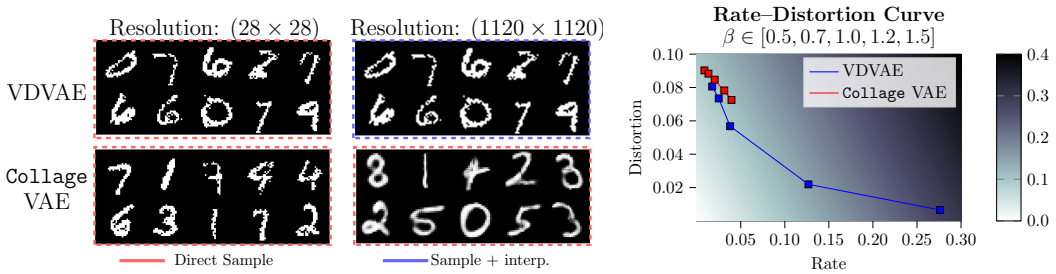


Figure 4: **[Left]** Samples obtained from VDVAEs and `Collage` VAEs on dynamically binarized MNIST. `Collage` VAEs can generate samples at resolutions unseen during training, adding detail that cannot be obtained by (1) sampling first (2) upscaling via bicubic interpolation, as done in the first row. **[Right]** Rate–distortion curve on dynamically binarized MNIST, with a curve obtained by sweeping β in a range. `Collage` VAEs are less sensitive to the tuning of β, and pay a marginal price in Pareto suboptimality to gain the ability to sample at different resolutions.

rate and distortion (Higgins et al., 2016). In a `Collage` VAE, $q_\phi(w|x)$ and $p_\phi(w)$ are parameterized as per (Child, 2020), except the approximate posterior $q_\phi(w|x)$ need not produce large feature maps but scalar `Collage` codes w. Moreover, $p_\theta(x|w)$ is a `Collage` such that by leveraging the consideration of Section 3.1, samples can be decoded at any resolution.

Results To investigate the properties of `Collage` VAEs, including quality of magnified samples, we compare VDVAEs and `Collage` VAEs on dynamically binarized MNIST. We report both test rate and distortion, following the analysis of (Alemi et al., 2018). This supports a more detailed evaluation of model behavior at different β weights for the rate. We train several `Collage` VAEs and VDVAEs with β ranging in $[0.5, 0.7, 1.0, 1.2, 1.5]$, and report the rate–distortion curve in Figure 4 (right). `Collage` VAEs are only marginally Pareto suboptimal, but are shown to be less sensitive to training under different β, which is a common strategy employed to stabilize VAE training ("KL warmup"). Furthermore, `Collage` VAEs can generate samples at resolution unseen during training, as showcased in Figure 4 (left). Direct samples at a magnification factor of $40\times$ reveal additional details over VDVAE samples that are magnified via bicubic interpolation. We note that as discussed in Section 5.1, the detail introduced by magnification is entirely dependent on the `Collage` class; future designs may be developed to introduce types of detail expected in a given dataset.

4.2 Neural `Collage` Compressors

Next, we apply Neural `Collage` to image compression. We store images as the parameters w of an affine `Collage` produced by a neural encoder. After training, the encoder can be used to compress additional images in parallel with a single forward pass by producing the corresponding w. Figure 3 provides an overview of the computation done by a Neural `Collage` compressor. `Collage` compressors employ learned feature maps as auxiliary domains.

For compression, the main desideratum is visual fidelity: regular domains D_n allow Neural `Collages` to capture intra–image self–similarity, whereas auxiliary domains U optimized for image quality complement them by focusing on inter–image patterns.

Method	↑ **PSNR**	**bpp** ↓		
Fractal (no aug.)	31.06	0.14	31.68	0.36
Fractal (augment.)	30.51	0.15	30.80	0.39
COIN	25.74	0.17	27.34	0.33
Neural `Collage` **(ours)**	31.30	0.13	32.12	0.31
block–DCT	33.22	0.13	34.65	0.32

Table 1: Average *peak signal-to-noise ratio* (PSNR) at low (≈ 0.15) and medium (≈ 0.30) *bits–per–pixel* (bpp) budgets of baselines (baseline fractal, implicit, spectral) and Neural `Collages` compressors. Neural `Collages` introduces less visible artifacts than other self–similarity or implicit compression schemes, narrowing the gap with spectral compression methods such as JPEG.

Compression of high–resolution aerial images We consider compressing images obtained from the DOTA large-scale aerial images dataset (Xia et al., 2018). From the DOTA training set, we produce 80000 random 40×40 crops as our training dataset. We optimize convolutional encoder parameters θ and auxiliary sources on the

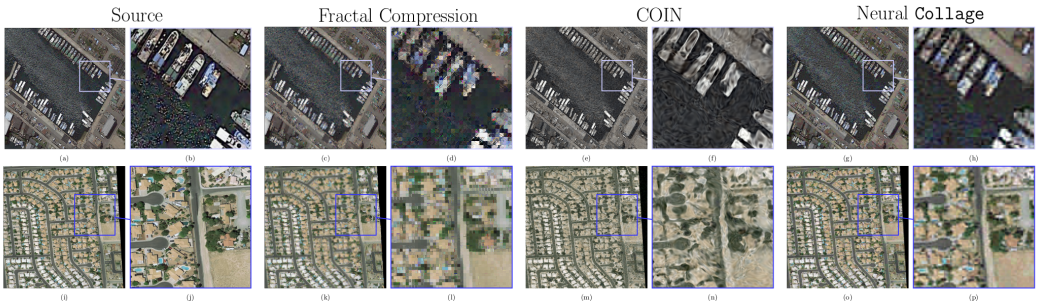


Figure 5: Visual comparison of decoded images from 1200×1200 crops of the DOTA dataset, obtained from different compression methods (≈ 0.30) bpp. Images decoded from Neural `Collage` codes exhibit less noticeable artifacts, with improved color preservation over COIN and more detail than fractal compression.

reconstruction objective $J(x, z^*(\theta, u)) = \sum_{i=1}^{m}(x_i - z_i^*(\theta, u))^2 + \|w\|_2^2$ The model is trained on the 40×40 crops and evaluated on 10 held-out 1200×1200 images. This is possible as an image of any resolution can be first broken up into blocks of appropriate size, in this case the training resolution, 40×40, passed through the encoder to obtain the corresponding parameters, then concatenated to construct a valid code for the entire image. This operation can be performed in parallel by treating each block as an element of a pseudo–batch. Neural `Collage` compressor can thus be used to compress images of any resolution at test time, without retraining.

Results We compare *peak signal-to-noise ratio* (PSNR), in addition encoding and decoding measurements for a variety of compression baselines. We contextualize our results with comparisons to both non–neural as well as another implicit neural compressor. In particular, we evaluate the performance of a standard PIFS–based fractal compression as per (Jacquin, 1993), implemented to exploit parallelization on GPU, and COIN (Dupont et al., 2021). Fractal compression baselines and `Collage` both use non–adaptive tiling partitioning schemes. We evaluate two variants of fractal compression, one where domain cells are augmented via rotations and color flips (Welstead, 1999), and one without augmentations. We further compare with block–DCT, the spectral lossy compression backbone of most JPEG codecs. All compression methods are evaluated at low and medium bpps, with metrics provided in Table 1. Figure 5 provides a visual comparison of images obtained by decoding the lossy code of each method. Neural `Collages` show less noticeable artifacts and improved color retention.

Finally, Table 1 provides wall–clock time measurements of all methods during the respective per image encoding and decoding procedures. Results for COIN and Neural `Collage` measure encoding times (including the training procedure), and encoding times after training. Neural `Collages` are orders of magnitude faster than fractal compression and, at test–time, of COIN. Although spectral lossy compressors common in state–of–the–art codecs perform with best PSNR in medium and high bpp settings, `Collages` narrows the gap in terms of reconstruction quality as well as encoding speed.

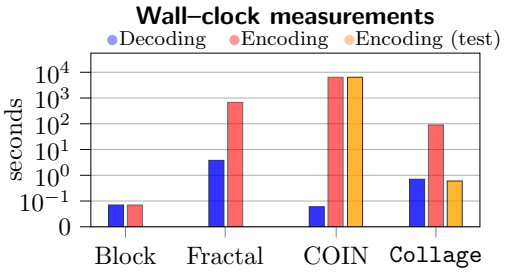


Figure 6: Wall–clock time (s) measurements of compression methods applied to the held–out set of DOTA. For neural compressors, we report encoding times including training, as well as encoding times on unseen images (after training). At test time, encoding for Neural `Collage` compressors is $100\times$ faster than fractal compression.

Code length of a `Collage` The total *bits–per–pixel* (bpp) cost of the code w corresponding to the parameters of a `Collage` depends on the coding scheme used to store each numeric entry. We exploit bounds on a, b, γ, enforced via `tanh` and softmax, as well as regularization to reduce the total cost. The L_2 regularization term on w is introduced to shrink the range of values assumed by elements of w, ensuring that less bits can be used for storage. We do not use lossless coding schemes to store parameters of `Collages` and other baselines. Further details are provided in the Appendix.

4.3 Neural `Collages` for Fractal Art

When data is represented through the parameters w of a `Collage`, it can be arbitrarily magnified by decoding at any resolution. The type of patterns revealed through magnification need not be corresponding to real detail missing from the image. In particular, the patterns found depend on how one generates domains, auxiliary domains and class of operator F_w. Similar phenomena have been observed in the literature of fractal compression (Mitra et al., 2000). As an example, consider Figure 7, where the fractal pattern of snowflakes within snowflakes does not correspond to reality. We call this type of globally self–referential magnification as fractalization of an image. Here, we use Neural `Collage` fractalizers to generate fractal art.

Experimental details We solve the inverse problem of a `Collage`, namely image x to w, via a convolutional architecture parametrized by θ. The objective of the inverse problem (3.3) is a reconstruction objective $J(x, z^*(\theta, u)) = \sum_{i=1}^{m}(x_i - z_i^*(\theta, u))^2$ where z^* is the fixed–point of the Neural `Collage` with parameters w computed through the encoder $w = \mathcal{E}_\theta(x)$. We choose the single domain D to be the entire image. Before applying F_w, we augment the domain via rotations of itself,

utilizing those as auxiliary domains. Figure 1 and 7 show example fractalizations possible with Neural `Collages`, on greyscale and RGB images. The images can be magnified to any resolution (up to memory limits), revealing multiple fractal levels.

5 Related Work and Conclusion

Implicit representations and models Representation of data implicitly through functions is extensively used in simulation (Osher et al., 2004). (Sitzmann et al., 2020; Mildenhall et al., 2020; Dupont et al., 2021) parametrize the implicit functions via neural networks for use in downstream tasks such as compression. Implicit models, on the other hand, solve optimization problems within their forward pass (Poli et al., 2020). Neural `Collages` belong to both classes of methods: a fixed–point iteration with parameters defining data implicitly. In particular, Neural `Collages` can be framed as a compactly–parametrized operator variant of *deep equilibrium networks* (DEQs) (Bai et al., 2019), with parameters produced by a hypernetwork (Ha et al., 2016).

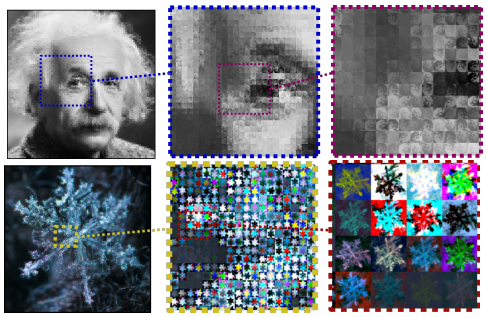


Figure 7: Fractalization of greyscale and RGB images via `Collage`. The images are compressed as the coefficients of a `Collage`, and decoded as its attractor. Note that the above is a showcase of *global* fractalization: no partitioning is performed, and the entire image itself is taken as the source.

Fractal compression The idea of representing images through *iterated function systems* (IFS) dates back to (Barnsley and Demko, 1985; Barnsley, 1986). Jacquin et al. (1992) introduces more flexible fractal compression schemes for images based on *partitioned iterated function systems* (PIFSs). Since then, alternative partitioning schemes i.e. adaptive quadtrees have been proposed. We refer to (Fisher, 2012) for an overview of the main variants. Additional references are provided in the Appendix.

Conclusion This work builds a framework for a learning–based approach to automated discovery of self–similarity. We introduce Neural `Collages`, implicit models designed to represent and manipulate data as the parameters of a structured fixed–point iteration, and showcase their application to compression and deep generative modeling of images. We envisage future use of Neural `Collage` with other data modalities with naturally occurring self–similarity, such as audio, sequences, or turbulent flows.

Acknowledgments

This work is supported by NSF(1651565), AFOSR (FA95501910024), ARO (W911NF-21-1-0125), ONR, DOE, CZ Biohub, and Sloan Fellowship.

References

A. Alemi, B. Poole, I. Fischer, J. Dillon, R. A. Saurous, and K. Murphy. Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168. PMLR, 2018.

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

M. F. Barnsley. Fractal functions and interpolation. *Constructive approximation*, 2(1):303–329, 1986.

M. F. Barnsley. *Fractals everywhere*. Academic press, 2014.

M. F. Barnsley and S. Demko. Iterated function systems and the global construction of fractals. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 399(1817): 243–275, 1985.

M. F. Barnsley and L. P. Hurd. *Fractal image compression*, volume 366. AK peters Wellesley, 1993.

M. F. Barnsley, V. Ervin, D. Hardin, and J. Lancaster. Solution of an inverse problem for fractals and other sets. *Proceedings of the National Academy of Sciences of the United States of America*, 83 (7):1975, 1986.

M. F. Barnsley et al. Fractal image compression. *Notices of the AMS*, 43(6):657–662, 1996.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

R. Child. Very deep vaes generalize autoregressive models and can outperform them on images. *arXiv preprint arXiv:2011.10650*, 2020.

T. Dao, N. S. Sohoni, A. Gu, M. Eichhorn, A. Blonder, M. Leszczynski, A. Rudra, and C. Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. *arXiv preprint arXiv:2012.14966*, 2020.

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.

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

Y. Fisher. *Fractal image compression: theory and application*. Springer Science & Business Media, 2012.

R. C. Guido, L. S. Vieira, S. B. Junior, F. L. Sanchez, M. B. A. Guilherme, K. I. C. Sergio, T. L. Scarpa, E. S. Fonseca, J. C. Pereira, and M. Monteiro. A fractal and wavelet-based approach for audio coding. In *Eighth IEEE International Symposium on Multimedia (ISM'06)*, pages 253–256. IEEE, 2006.

D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

A. E. Jacquin. Fractal image coding: A review. *Proceedings of the IEEE*, 81(10):1451–1465, 1993.

A. E. Jacquin et al. Image coding based on a fractal theory of iterated contractive image transformations. *IEEE Transactions on image processing*, 1(1):18–30, 1992.

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*, 2021.

G. Julia. Mémoire sur l'itération des fonctions rationnelles. *J. Math. Pures Appl.*, 8:47–245, 1918.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

H. Lee, A. Battle, R. Raina, and A. Y. Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pages 801–808, 2007.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

B. B. Mandelbrot. Fractal aspects of the iteration of z$\rightarrow$ λz (1-z) for complex λ and z. *Annals of the New York Academy of Sciences*, 357(1):249–259, 1980.

B. B. Mandelbrot and B. B. Mandelbrot. *The fractal geometry of nature*, volume 1. WH freeman New York, 1982.

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 *European conference on computer vision*, pages 405–421. Springer, 2020.

S. K. Mitra, C. Murthy, and M. K. Kundu. A technique for image magnification using partitioned iterativdupont2022coine function system. *Pattern recognition*, 33(7):1119–1133, 2000.

G. E. Øien and S. Lepsøy. *A Class of Fractal Image Coders with Fast Decoder Convergence*, page 153–175. Springer-Verlag, Berlin, Heidelberg, 1995. ISBN 0387942114.

S. Osher, R. Fedkiw, and K. Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004.

M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Torchdyn: A neural differential equations library. *arXiv preprint arXiv:2009.09346*, 2020.

C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan, and A. Edelman. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.

V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.

C. Song, S. Havlin, and H. A. Makse. Self-similarity of complex networks. *Nature*, 433(7024): 392–395, 2005.

K. Sun, S. Lee, and P. Wu. Neural network approaches to fractal image compression and decompression. *Neurocomputing*, 41(1-4):91–107, 2001.

N. Tishby, F. C. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

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.

A. Vulpiani, F. Cecconi, and M. Cencini. *Chaos: from simple models to complex systems*, volume 17. World Scientific, 2009.

S. T. Welstead. *Fractal and wavelet image compression techniques*, volume 40. Spie Press, 1999.

G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang. Dota: A large-scale dataset for object detection in aerial images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3974–3983, 2018.

Z. Zha, X. Yuan, J. Zhou, C. Zhu, and B. Wen. Image restoration via simultaneous nonlocal self-similarity priors. *IEEE Transactions on Image Processing*, 29:8561–8576, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] **See Appendix B.2**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Self–Similarity Priors
Supplementary Material

A Background and Extended Formulation

A.1 Metric Spaces

Lemma 1 (Useful results on bounded and closed sets)**.** *The following hold:*

i. *Let* $\mathbb{X} = \bigcup_{i=1}^{n} \mathbb{A}_i$ *such that* $\mathbb{A}_i$ *is a* ***closed*** *subset of* $\mathbb{R}^n$ *for all* $i = 1, \ldots, n$*. Then,* $\mathbb{X}$ *is* ***closed****.*

ii. *Let* $\mathbb{X} = \bigcup_{i=1}^{n} \mathbb{A}_i$ *such that* $\mathbb{A}_i$ *is a* ***bounded*** *subset of* $\mathbb{R}^n$ *for all* $i = 1, \ldots, n$*. Then,* $\mathbb{X}$ *is* ***bounded****.*

iii. *Let* $f : \mathbb{R}^n \to \mathbb{R}^n$ *to be a continuous function and let* $\mathbb{A}$ *to be a closed, bounded subset of* $\mathbb{R}^n$*. Then* $f(\mathbb{A}) = \{f(x) : x \in \mathbb{A}\}$ *is also closed and bounded.*

Definition 3 (Metric space)**.** *A metric space is a pair* $(\mathbb{X}, d)$ *where* $\mathbb{X}$ *is a set and* $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ *is a map such that for all* $x, y, z \in \mathbb{X}$ *the following conditions hold:*

$$\begin{aligned} &i.\ \ d(x,y) \geq 0; && iii.\ \ d(x,y) = d(y,x); \\ &ii.\ \ d(x,y) = 0;\ \Leftrightarrow x = y && iv.\ \ d(x,y) \leq d(x,z) + d(z,y). \end{aligned}$$

The function d *is called "metric". Note that a metric space is called* ***compact*** *if* $\mathbb{X}$ *is closed and bounded.*

Definition 4 (Convergent sequence)**.** *Given a metric space* $(\mathbb{X}, d)$*, a sequence* $\{x_t\}_{t=0}^{\infty}$ *is said to converge to some* $x^* \in \mathbb{X}$

$$\forall \epsilon > 0 \ \ \exists t^* \ \ : \ \ \forall t > t^* \ \ d(x^*, x_t) < \epsilon.$$

Definition 5 (Cauchy sequence)**.** *A sequence* $\{x_t\}_{t=0}^{\infty}$ *is* $\mathbb{X}$ *is a Cauchy sequence if*

$$\forall \epsilon > 0 \ \ \exists t^* \ \ : \ \ \forall s, t > t^* \ \ d(x_s, x_t) < \epsilon.$$

Definition 6 (Complete metric space)**.** *A metric space* $(\mathbb{X}, d)$ *is complete if every Cauchy sequence in* $\mathbb{X}$ *is convergent in* $\mathbb{X}$*.*

Definition 7 (Hausdorff space)**.** *Let* $(\mathbb{X}, d)$ *be a complete metric space, and define* $\mathcal{H}(\mathbb{X})$ *as the set of all compact subsets of* $\mathbb{X}$*:*

$$\mathcal{H}(\mathbb{X}) = \{\mathbb{A} \subset \mathbb{X} : \mathbb{A} \textit{ is compact}\}.$$

Definition 8 (Hausdorff metric)**.** *Let* $(\mathbb{X}, d)$ *be a metric space and let* $\mathbb{Y}, \mathbb{Z} \subset \mathbb{X}$*. The* ***Hausdorff metric*** $d_{\mathcal{H}} : \mathcal{H}(\mathbb{X}) \times \mathcal{H}(\mathbb{X}) \to \mathbb{R}$ *is then defined as*

$$d_{\mathcal{H}}(\mathbb{Y}, \mathbb{Z}) = \max \left\{ \sup_{y \in \mathbb{Y}} d(y, \mathbb{Z}), \sup_{z \in \mathbb{Z}} d(\mathbb{Y}, z), \right\}$$

where $d(y, \mathbb{Z}) := \inf_{z \in \mathbb{Z}} d(y, z)$*.*

Theorem 2 (Completeness of Hausdorff metric space (Barnsley and Hurd, 1993))**.** *Let* $(\mathbb{X}, d)$ *be a complete metric space. Then* $(\mathcal{H}(\mathbb{X}), d_{\mathcal{H}})$ *is a complete metric space.*

(Barnsley and Hurd, 1993) calls $(H(\mathbb{X}), d_{\mathcal{H}})$ *the space where fractals live*. It is the space where the mathematical foundations necessary to generate fractals via iterated function systems are developed.

A.2 Contraction mappings

Definition 9 (Lipschitz function)**.** *Let $(\mathbb{X}, d)$ be a metric space. A map $f : \mathbb{X} \to \mathbb{X}$ is Lipschitz with constant ℓ if there exists $\ell > 0$ such that*

$$\forall x, y \in \mathbb{X} \quad d(f(x), f(y)) \leq \ell d(x, y).$$

A Lipschitz function is then called *contractive* iff $\ell < 1$. Moreover, if $f : \mathbb{X} \to \mathbb{X}$ is Lipschitz, then f is continuous.

Note: A map f is contractive if brings any elements of $\mathbb{X}$ *close together*. The Lipschitz constant ℓ measures (bounds) how closer the points are brought together by one application of f. Thus, intuitively, if we iterate a discrete dynamical system

$$x_{t+1} = f(x_t)$$

starting from any point of x_0 within $\mathbb{X}$, the sequence $\{x_t\}_{t=0}^{\infty}$ will converge to a **unique** fixed point x^* such that $f(x^*) = x^*$.
It is worth to notice that linear affine maps $f : \mathbb{R}^n \to \mathbb{R}^n$ (where we consider the standard Euclidian distance to form a complete metric space on $\mathbb{R}^n$)

$$f(x) = Ax + b$$

are contractive if and only if

$$\|A\|_2 = \sup_{x \in \mathbb{R}^n} \frac{\|Ax\|_2}{\|x\|_2} < 1$$

If this is the case any sequence $\{x_t\}_{t=0}^{\infty}$ would converge to a fixed point

$$x^* = (I - A)^{-1} b.$$

The above intuitions can be formalized in the following classic result

Theorem 3 (Banach fixed–point theorem)**.** *Let $(\mathbb{X}, d)$ be a (non–empty) complete metric space and let $f : \mathbb{X} \to \mathbb{X}$ be a contractive map. Then f admits a **unique** fixed point $x^* \in \mathbb{X}$ such that any sequence $\{x_t\}_{t=0}^{\infty}$ defined by the iteration*

$$x_{t+1} = f(x_t)$$

converges to x^* for any starting point $x_0 \in \mathbb{X}$, i.e.

$$\exists! x^* \in \mathbb{X} \quad \lim_{t \to \infty} x_t = \lim_{t \to \infty} f(x_t) = x^* \quad \forall x_0 \in \mathbb{X}.$$

Corollary 2 (Collage theorem)**.** *Under the assumptions of Theorem 3, it holds*

$$\forall x \in \mathbb{X} \quad d(x, x^*) \leq \frac{1}{1 - \ell} d(x, f(x)).$$

A.3 Iterated Function Systems

With the aim of deriving fractal compression algorithms it is necessary to define functions on the Hausdorff metric space $(\mathcal{H}(\mathbb{X}), d_{\mathcal{H}})$. In particular, let $\{f_1, f_2, \ldots, f_K\}$ be a collection of maps on $\mathbb{X}$, $f_k : \mathbb{X} \to \mathbb{X}$. Then, we can define maps $F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$ by

$$F(\mathbb{A}) = \bigcup_{k=1}^{K} f_k(\mathbb{A}) \quad \forall \mathbb{A} \in \mathcal{H}(\mathbb{X})$$

where $f_k(\mathbb{A})$ is intended as $f_k(\mathbb{A}) = \{f_k(a) : a \in \mathbb{A}\}$. Moreover, note that $\mathbb{A} \in \mathcal{H}(\mathbb{X})$ is also a compact subset of $\mathbb{X}$. The following results shows that if all f_k are contractive, then also F is.

Theorem 4 (Contractivity of maps on the Hausdorff metric space)**.** *If for all $k = 1, \ldots, K$, the maps $f_k : \mathbb{X} \to \mathbb{X}$ are contractive with Lipschitz constant $\ell_k < 1$, then $F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X}); \mathbb{A} \mapsto \bigcup_{k=1}^{K} f_k(\mathbb{A})$ is contractive in the Hausdorff metric with Lipschitz constant $L = \max_k \{\ell_k\}_k$.*

Definition 10 (Iterated function system (IFS))**.** *An iterated function system is a collection* $\{f_1, \dots, f_K\}$ *of contractive maps* $f_k : \mathbb{X} \to \mathbb{X}$*, represented by* $F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$*.*

Thanks to its contractivess (as established by Theorem 4), F define a unique fixed point (*attractor*) by the *Banach fixed–point theorem* (Theorem 3), i.e. the discrete iteration defined by

$$\mathbb{A}_{t+1} = F(\mathbb{A}_t) = \bigcup_{k=1}^{K} f_k(\mathbb{A}_t)$$

converges to $\mathbb{A}^* \in \mathcal{H}(\mathbb{X})$ for $t \to \infty$. Since the attractor $\mathbb{A}^*$ is unique, it is completely defined by the map F. The *data encoding problem* can be then formulated as follows.

Problem: *Fractal Data Encoding* (Section 2.2)

If we are given some set $\mathbb{S} \in \mathcal{H}(\mathbb{X})$ (our *data*), can we find map F whose attractor is $\mathbb{S}$? In other words, given data $\mathbb{S} \in \mathcal{H}(\mathbb{X})$, find the collection of maps $f_k : \mathbb{X} \to \mathbb{S}$ such that the following conditions hold

$$i.\ \ F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X}); \mathbb{A} \mapsto \bigcup_{k=1}^{K} f_k(\mathbb{A}) \text{ is contractive;}$$

$$ii.\ \ \mathbb{S} \text{ is } \underline{\text{the}} \text{ fixed point of } F,\ \mathbb{S} = F(\mathbb{S}) = \bigcup_{k=1}^{K} f_k(\mathbb{S});$$

Properties of data encoding Note that condition $ii.$ suggests that, in order for the problem to admit a solution, data should be made up of transformed copies of itself. Specifically, we are assuming that it is possible to take the data $\mathbb{S}$, copy it K-times, apply to the copies some contractive transformations and finally stitch them together to reconstruct the initial data $\mathbb{S}$. The uniqueness of an attractor induced by the contractivity of F is fundamental to practically solve the fractal encoding problem because if we can find an F such that $\mathbb{S} = F(\mathbb{S})$, then we will be sure that F is the unique solution of the encoding problem.

On the Collage Representation By applying the *Collage Theorem* (Corollary 2) to F using the Hausdorff metric $d_{\mathcal{H}}$ we have

$$\begin{aligned} & d_{\mathcal{H}}(\mathbb{S}, \mathbb{A}^*) \leq \frac{1}{1-L} d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S})) \\ \Leftrightarrow\ \ & d_{\mathcal{H}}(\mathbb{S}, \mathbb{A}^*) \leq \frac{1}{1-\max_k\{\ell_k\}} d_{\mathcal{H}}\left(\mathbb{S}, \bigcup_{k=1}^{K} f_k(\mathbb{S})\right) \end{aligned}$$

This means that if we can't stitch the transformed copies $f_k(\mathbb{S})$ together to perfectly reconstruct the data $\mathbb{S}$, i.e.

$$d_{\mathcal{H}}\left(\mathbb{S}, \bigcup_{k=1}^{K} f_k(\mathbb{S})\right) \neq 0 \quad (\Leftrightarrow \mathbb{S} \neq F(\mathbb{S})),$$

then the lower the Lipschitz constant L of the IFS is, the lower the distance between the data $\mathbb{S}$ and the attractor $\mathbb{A}^*$ of $\mathbb{S}$ will be given a mismatch $d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S}))$. As mentioned in the main text, this implicitly promotes the use of "very contractive" maps f_k (i.e. with low ℓ_k).

A learning perspective to the *fractal data encoding* In the language of machine learning practice, the *fractal data encoding* problem can be translated into finding a parametric representation $f_k(\,\cdot\,; w_k)$, $w \in \mathbb{R}^{n_w}$ for the functions $f_k(\cdot)$ (e.g. Neural Networks with parameters w_k) where the parameters $w = (w_1, \dots, w_K) \in \mathbb{W}$ are trained to minimize the Hausdorff metric loss function $d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S}; w))$

naturally induced by the *Collage Theorem*, i.e.

$$\begin{aligned} \min_{w} \quad & d_{\mathcal{H}}(\mathbb{S}, F(\mathbb{S}; w)) \\ \text{subject to} \quad & F(\mathbb{S}; w) = \bigcup_{k=1}^{K} f_k(\mathbb{S}; w_k) \\ & w \in \mathbb{W} \end{aligned}$$

Once optimal $f_k(\ \cdot\ ; w_k)$ are computed, it is easy to find the data that F encodes (i.e. the *decoding* process): after sampling any initial condition $\mathbb{A}_0$, the encoded data can be obtained by iterating

$$\mathbb{A}_{t+1} = F(\mathbb{A}_t),$$

until convergence to $\mathbb{A}^* \approx \mathbb{S}$.

A.4 Partitioned Iterated Function Systems

Existance of solutions of the fractal encoding (inverse) problem requires data to be perfectly representable by an IFS. Conversely, we can define *self-similar* data if it is the attractor of an IFS.

Definition 11 (Self–similar sets)**.** *A set $\mathbb{S}$ is called self–similar if and only if there exists a contractive map $F : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$ whose attractor is $\mathbb{S}$, i.e. $\mathbb{S} = F(\mathbb{S})$.*

While verifying the self–similarity of a specific data point is undoubtedly a NP–hard problem, natural data (e.g. in an image dataset) is unlikely to satisfy this strict property. The challenge is that the self-similarity property has to be global across the set $\mathbb{S}$. That is, the entire set $\mathbb{S}$ has to be made up of smaller copies of itself, or parts of itself. If one zooms in on it, it would display the same level of detail, regardless of the resolution scale (Welstead, 1999).

For this reason, it is necessary to extend the fractal encoding to more general, non globally self–similar sets. This can be achieved by introducing the technology of *partitioned function systems* (PFS) where the domains of the contraction maps f_k are restricted.

Definition 12 (Partitioned Function System)**.** *Let $(\mathbb{X}, d)$ be a complete metric space and let $\mathbb{D}_k \subset \mathbb{X}$ for $k = 1, \ldots, K$. A partitioned function system is a collection of contraction maps $f_k : \mathbb{D}_k \to \mathbb{X}$.*

Note that, according to Fisher (2012), it is not possible to extend Theorem 3 to PFSs in the general case to effectively ensure existance and uniqueness of fixed points. Intuitively this is due to the fact that the domains of f_k are restricted and the convergence of the decoding dynamics (i.e. the fixed-point iteration)

$$\mathbb{A}_{t+1} = \bigcup_{k=1}^{K} f_k(\mathbb{A}_t)$$

becomes dependent on the choice of the initialization $\mathbb{A}_0$. In fact, even though if choose $\mathbb{A}_0 \subset \bigcap_{k=1}^{K} \mathbb{D}_k$, after one step we may end up with an empty set. Note that this is generally not a problem in practice when applying PFSs to or `Collage` working with common type of data such as images or audio signals.

A.5 Functional Representation of Data

In order to derive an implementation–oriented formulation of data encoding with partitioned functions systems in the general case, it can be convenient to rely on a *functional* description of data (see e.g. (Welstead, 1999; Fisher, 2012)). In example, images (of infinite resolution) can be represented as functions from the unit square to $\mathbb{R}$. Time series can also be thought as real continuous functions over a compact time domain.

Specifically we restrict our analysis to the space $\mathcal{F} = \{\phi : \mathsf{dom}(\phi) \to \mathbb{R}\}$ of *data* defined the *graphs* $(z, \phi(z))$, $z \in \mathsf{dom}(\phi)$ of (measurable) functions over the compact domain $\mathsf{dom}(\phi)$ and values in $\mathbb{R}$. $\mathsf{dom}(\phi)$ is assumed to be a compact subset of $\mathbb{R}^n$.

Partitioned fractal encoding *a la* Welstead (1999) We choose $\mathcal{F} = L^p(\mathcal{X}; \mathbb{R})$ with $\mathcal{X}$ a compact subset of $\mathbb{R}^n$ and we equip it with a metric $d_{\mathcal{F}}$ induced by the Lebesgue measure

$$d_{\mathcal{F}}(\phi, \psi) = \left(\int_{\mathcal{X}} |\phi(x) - \psi(x)|^p \mathrm{d}x \right)^{1/p}.$$

Then $(\mathcal{F}, d_{\mathcal{F}})$ is a complete metric space and the *Banach fixed–point* (Theorem 3) holds. Then, we specialize the partitioned function system on $(\mathcal{F}, d_{\mathcal{F}})$ as comprised of the following collections of K elements:

a. *sub–domains* $\mathsf{D}_k \subset \mathcal{X}$;
b. *invertible contractive maps* $v_k : \mathsf{D}_k \to \mathsf{R}_k \subset \mathcal{X}$;
c. *maps* $f_k : \mathcal{F} \to \mathcal{F}$ *defined as*

$$f_k(\phi)(x) = c_k \phi(v_k^{-1}(x)) + d_k \quad \forall \phi \in \mathcal{F}, x \in \mathcal{X}.$$

Note that we can define subsets R_k as the range of v_k operating on D_k, i.e. $\mathsf{R}_k = v_k(\mathsf{D}_k)$. The constants c_k, d_k realize an affine trasformation on ϕ by expanding/contracting and shifting the range of ϕ. The contractive maps v_k are the "spatial part" of the PFS and map the domains D_k to their respective ranges R_k. v_k are often chosen to be affine maps

$$v_k(x) = A_k x + b_k, \quad A_k \in \mathbb{R}^{n \times n}, b_k \in \mathbb{R}^n.$$

Note that it is possible to choose A_k and c_k so that f_k is contractive. In particular, it is sufficient to require $|c_k| |\det A_k|^{1/p} < 1$

Definition 13 (Tiling partition)**.** *A collection of ranges* $\mathsf{R}_k \subset \mathcal{X}$ *is said to tile* $\mathcal{X}$ *iff* $\mathcal{X} = \bigcup_{k=1}^K \mathsf{R}_k$ *and* $\forall i \neq j \;\; \mathsf{R}_i \cap \mathsf{R}_j = \emptyset$.

If the ranges R_k *tile* $\mathcal{X}$, we can define the operator $F : \mathcal{F} \to \mathcal{F}$ by

$$F(\phi)(x) = f_k(\phi)(x) \;\text{ for }\; x \in \mathsf{R}_k,$$

i.e.

$$F(\phi)(\mathcal{X}) = \bigcup_{k=1}^K f_k(\phi)(\mathsf{R}_k) = \bigcup_{k=1}^K c_k \phi(\mathsf{R}_k) + d_k = \bigcup_{k=1}^K c_k \phi(v_k^{-1}(\mathsf{D}_k)) + d_k$$

Since the ranges R_k tile $\mathcal{X}$, F is defined for all $x \in \mathcal{X}$, so $F(\phi)$ is a function of the same class of ϕ.

If ϕ is an image on the unit square tiled by the ranges R_k, then $F(\phi)$ will also be an image on the unit square.

Assuming all the maps f_k to be contractions on $\mathcal{F}$, F satisfies the Banach fixed point theorem and has unique fixed point $\phi^* \in \mathcal{F}$ such that

$$\phi^* = F(\phi^*).$$

Collage and PIFS for digital images Similarly to the main text, we can define PIFS by restricting our analysis to *affine* maps, operating on the space of discrete images of a given resolution with a total number m of pixels. Note that pixels across channels can be treated effectively as different elements.

We assume the value of each pixel to range in $\mathbb{R}$ and to collect all the pixel values in an *ordered*[3] in a vector $z \in \mathbb{R}^m$. Then, a *partitioned function system* (Jacquin, 1993; Welstead, 1999; Fisher, 2012) can be represented as the *structured* map

Definition 14 (Discrete PIFS)**.** *Consider a* m*-pixel image represented by the ordered vector* $z \in \mathbb{R}^m$*. Then, a Discrete PIFS is defined as the parametric linear map* $F : \mathbb{R}^m \to \mathbb{R}^m$*:*

$$F(z; w) = \sum_{k=1}^K a_k T_k P_k S_k z + \sum_{k=1}^k b_k T_k \mathbb{1}, \quad w = (a_1, \ldots, a_K, b_1, \ldots b_K) \tag{A.1}$$

where $T_k,\; P_k, S_k$ *are defined similarly to Definition 1.*

[3] with a specific predefined criterion.

Algorithm 1 Fractal Compression with PIFS Jacquin et al. (1992)
● Encoding ● Decoding

Input: Image I, parametrized PIFS $F(\ \cdot\ ; w)$.
Partition I into two collections of N domains D_n and K ranges R_k.
for k **from** 1 **to** K **do**
  $\mathsf{D}_n^*, w_k^* = \arg\min_{\mathsf{D}_n, w_k} d(f_k(\mathsf{D}_n; w_k), \mathsf{R}_k)$ *matching*
end for
Store `code` $:= \{w_k^*, (n,k)\}_{k=1}^K$ *fractal code and domain index*
Initialize $\tilde{\mathsf{I}}_0$ as any image with same size as I
repeat
  $\tilde{I}_{t+1} = \bigcup_{k=1}^{K} f_k(\mathsf{D}_{n,t}^*; w_k^*)$
until convergence
return $\tilde{I}^*$ *decoded data*

Figure 8: Algorithmic summary of fractal compression with PIFS.

The output of the collage operator for R_k is thus a pooled and scaled version of D_k translated block–wise by b_k. A symbolic formulation of the collage operator can be also given by

$$\mathsf{R}_k = f_k(\mathsf{D}_k; w_k), \quad w_k = (a_k, b_k)$$

Since the collection of range cells R_k tiles the whole image I, we can write (*a la* IFS)

$$F(\mathsf{I}; w) = \bigcup_{k=1}^{K} f_k(\mathsf{D}_k; w_k)$$

Remark 1 (Extensive search)**.** *Note that, in the classic setting of Jacquin (1993); Welstead (1999), for each range cell R_k, the corresponding domain cell D_k has to be found by extensive search through the set of all possible pooled domain cells.*

We provide a compact algorithmic summary of the core steps in fractal compression as per Jacquin et al. (1992); Jacquin (1993) in Figure 8. Other variants of fractal compressions have historically been attempted, including ones with adaptive partitions and different algorithms to solve the combinatorial search during encoding.

B Additional Details

Code–length flexibility: `Collage` and PIFS A `Collage` operator is a generalization of PIFS operators of fractal compression algorithms (Jacquin et al., 1992; Jacquin, 1993; Fisher, 2012). In particular, the `Collage` introduces additional flexbility in the choice of *code length* i.e. how many bits to allocate to the compression code. For example, given non–adaptive (square) tiling domain and range cells[4], the *bits–per–dimension* (bpp) cost of PIFS–based fractal compression of Jacquin et al. (1992) is $\frac{\texttt{cost}_{w_k}}{n_{r,k}^2}$, where $\texttt{cost}_{w_k}$ is the cost of saving the parameters of single element in the f_k of F_w.

Here, other than seeking further compression and reduction of $\texttt{cost}_{w_k}$ by modifying the class of operator, the only degree–of–freedom is to reduce or increase the dimensions of tiling partitions. Note that modifying the partition scheme has not only effect on the bpp cost but also on the type of self–similarity that can be captured. Instead, `Collage` operators offer an additional design axis; indeed, the bpp budget – given a fixed partition – can be modified by increasing or decreasing the number of auxiliary domains introduced as learnable feature maps. This number is independent on the number of original domains, whereas the number of additional domains generated as affine augmentations of fractal compression (see Welstead (1999); Fisher (2012) for details) is not.

On adaptive partitions Elaborated partitioning schemes have been developed for PIFS–based fractal compression methods (Fisher, 2012). While the analysis and empirical comparisons of this work have been centered around the operators, rather than partition schemes, we remark that `Collage` are compatible with alternative and potentially adaptive schemes. Much like for PIFS, this is a likely direction for further improvement of Neural `Collage`.

B.1 `Collage` Schematic

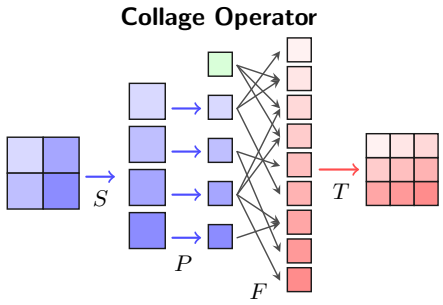


Figure 9: Conceptual schematic of a `Collage`. In blue, domain cells S_n; in red, range cells R_k of ranges. Green highlights auxiliary domains. A step can be broken down into (1) S partitions into domains (2) P reduces dimensions to ensure dimensions match (3) F produces all range cells, each following (3.2) (4) T rearranges the output.

B.2 Extended related work

Attention operators and patches There exist superficial similarities between `Collages` and attention operators (Vaswani et al., 2017). In particular, recent variants of vision transformers (Dosovitskiy et al., 2020) where attention acts on square patches, can be seen as a single step of a `Collage`, where source and target partition match and aggregation weights are found via similarity scores. `Collages` differ from attention in that they are structured fixed–point iterations, are built to accommodate non–overlapping partitions, are resolution–invariant, and have a compact parametrization that can be used as a compression code. It remains to be seen whether investigating attention operators through the lenses of `Collages` can yield improvements in theoretical understanding or performance.

Fractal compression Sun et al. (2001) parametrize elements of the iterative map with small neural networks. The proposed method still requires training on each image, with marginal improvements

[4] Although different choices are possible, "tiling" partitions are most convenient; when applying F, a domain partition D can be identified via single $\lceil \log_2 N \rceil$–bit integer address.

over standard variants. (Guido et al., 2006) provide a preliminary exploration of fractal coding for audio. Despite the extensive body of work, fractal methods for image compression are rarely used in place of other codecs due to slow encoding. As discussed in the main text sections, Neural `Collages` address this limitation via neural network amortization. We note that Neural `Collage` remain compatible with adaptive partitioning schemes, which provides a likely avenue of further improvement. We highlight a line of work on different probabilistic models of self–similarity (Zha et al., 2020) for tasks such as image restoration.

C Additional Experiment Details

Hardware and software The experiments have been performed on a workstation with 2 NVIDIA GEFORCE RTX 3090 GPUs. We use JAX[5] for model implementation and distributed training.

C.1 Neural `Collages` for Fractal Art

We construct the encoder $\mathcal{E}$ for w by stacking 4 blocks composed of interleaved depthwise and pointwise convolutions. We train for 5000 iterations on each image displayed in Figure 7 with AdamW (Loshchilov and Hutter, 2017). We produce U by augmenting at each step of the `Collage` with rotations of $[90, 180, 270]$ degrees and flips, produced by multiplying all pixels values of a domain cell by -1. We do not use any additional learned auxiliary domain, so that the patterns can be kept globally fractal.

Neural `Collage` Texturizers We report additional results in D.3, where a Neural `Collage` is used to texturize images by optimizing transformations of a fixed U, provided as external "texture source". To promote utilization of texture sources we introduce a coefficient to weigh U relative to D in the `Collage` iteration. We note that in this case the Neural `Collage` is not leveraging any self–similarity; rather, this should be intended as a display of the capability of a Neural `Collage` to aggregate both external as well as self–referential information to achieve a given task.

C.2 Neural `Collages` for Generation

We design the architecture of a `Collage` following (Child, 2020). Table 3 describes the model structure. We introduce 30 learned auxiliary domains, parametrized to be pixel patches of same size of range cells. As the tiling partition, we choose a single domain cell of size 28×28 and size 4 range cells of 14×14. All models use a Bernoulli likelihood. Note that in this case, the fixed–point of the generator $p(x|w)$, chosen as a `Collage`, is given by the collection of all Bernoulli parameters, one for each pixel. This is thus an example of a `Collage` that is does not decode pixel–values of an image as its fixed–point, but rather parameters of their distributions. We optimize the `ELBO` by sweeping the KL weight β as discussed in Figure 4 for 2000 epochs. Additional training details are provided in 4.

C.3 Neural `Collages` for Compression

We construct the encoder $\mathcal{E}$ for w by stacking 4 blocks composed of interleaved depthwise and pointwise convolutions. We train for 5 epochs with AdamW Loshchilov and Hutter (2017) on a dataset of 8000 crops of size 40×40 obtained from the DOTA Xia et al. (2018) aereal image training dataset. The dataset is generated (statically) randomly by applying a random rotation, followed by a random crop. We produce the 10 held–out images of size 1200×1200 with a similar procedure, applied to the test dataset. We note that DOTA images are all of different resolutions, motivating the above procedure. Speedup results are provided in Figure 12.

As baselines, we use the official COIN Dupont et al. (2021) implementation. We develop a GPU–parallel version of fractal compression with PIFS Jacquin et al. (1992); Jacquin (1993); Welstead (1999); Fisher (2012) as a baseline. We use the same partition strategy as for Neural `Collages`, namely tiling into domain and range cells. Our evaluation includes a fractal compression variant which incorporate U by augmenting domains, D at each step, with rotations of $[90, 180, 270]$ degrees and flips, produced by multiplying all pixels values of a domain cell by -1. The matching problem of domains to ranges is solved via least–squares as per Welstead (1999). We parallelize the least–square solving across domains.

C.4 Computation of bits–per–pixel

We report the per–image *bits–per–pixel* (bpp) cost of compression baselines and Neural `Collage` compressors.

[5] https://github.com/google/jax

	K	N	V	ϵ	$\texttt{bpp}_u$	$\texttt{bpp}_a$	$\texttt{bpp}_b$	total
low–bpp	4	1	3	3	$9 \cdot 10^{-4}$	$6.6 \cdot 10^{-3}$	$6.3 \cdot 10^{-3}$	0.134
medium–bpp	4	1	10	4	$8.9 \cdot 10^{-4}$	$6.5 \cdot 10^{-3}$	$5.9 \cdot 10^{-3}$	0.319

Table 2: Example *bits–per–pixel* (bpp) code length computation for Neural `Collage`. To determine $\texttt{bpp}_a$ and $\texttt{bpp}_b$ we consider respective maximum values and represent their quantized integer range as discussed in C.4. The cost of auxiliary inputs is amortized on the 10 held–out 1200×1200 crops of DOTA, as decoding uses the same learned auxiliary domains for all images. We consider the auxiliary cost $\texttt{bpp}_u$ as part of the fractal code for a worst–case comparison, noting that reutilization of the same compressor eventually amortizes the cost to 0. For M images, $\lim_{m\to\infty} \texttt{bpp}_u = 0$.

Neural `Collage` Computation of the bpp of fractal codes generated by a Neural `Collage` compressor requires the following considerations. First, mixing weights $\gamma_{k,n}$ are premultiplied to both $a_{k,n}$ and $b_{k,n}$. The same holds for mixing weights of auxiliary domains. We store a single b_k for each R_k, noting that the corresponding term can be precomputed as $b_k = \sum_{n=1}^{N} \gamma_{k,n} b_{k,n}$. Further, we exploit the a priori knowledge that $a, b \in [-1, 1]$, enforced via `tanh`, in combination with significant digit clipping. Quantizing by clipping to a threshold ϵ of significant digits, in combination with the bounds enforced by `tanh`, allows bit–packing each into less than $\lceil \log_2 10^\epsilon \rceil + 2$. One of the 2 additional bits is for sign information. This can be verified by noticing that by quantizing the range of values, after multiplying by 10^ϵ, is contained by the integer range $[-10^\epsilon, 10^\epsilon - 1]$. In practice, the number of bits is less than $\lceil \log_2 10^\epsilon \rceil + 2$ since not all values in entire integer interval defined by ϵ–quantization are utilized by $a_{n,k}$ and b_k for a given image. In particular, we consider maximum absolute values of the quantized values, and restrict the interval accordingly.

Neural `Collage` do not require storing of domain cell addresses, since each map of the `Collage` corresponding to a range cell (see 3.2) always transforms all domains. The specification of patch–sizes, especially when they are the same across all domains, and across all ranges, as well as type of pooling operators can be considered part of the codec, adding a negligible amount of bits. Further considerations are necessary in one wishes to employ more elaborate partition schemes (Fisher, 2012).

The overall cost is given by

$$\texttt{bpp} = K(N + V)\texttt{bpp}_a + K\texttt{bpp}_b + V\texttt{bpp}_u \tag{C.1}$$

where N is the number of domains, V is the number of learned auxiliary cells, and K the number of ranges. We indicate with $\texttt{bpp}_u$ the cost of saving auxiliary learned patches. This cost is amortized across each image of the held–out set, as the patches are the same for a given Neural `Collage`. For V auxiliary patches of size $h \times w \times c$, the (non–amortized) bit cost is $32 \cdot V \cdot h \cdot w \cdot c$ bits. We provide some example calculations in 2.

COIN We use the official implementation of (Dupont et al., 2021), where the bpp is computed by serializing the weights of the network into a bytes.

Fractal compression baseline We quantize fractal compression affine maps f_k into half–floats, 16 bits for each a_k and b_k. As the address of the domain cell associated to a given range R_k, we store the index with cost $\lceil \log_2 (N + V) \rceil$–bit, where N is the number of source domains D and V the number of auxiliary domains U.

block–DCT We apply a forward, two–dimensional *discrete cosine transform* (DCT) to patches of sizes 12 (high bpp) and 16 (medium bpp) and filter all but the lowest coefficient. The total cost is thus $32 \cdot n_{\text{patches}}$. The image is decoded by applying an inverse DCT.

	Architecture			Learning Parameters		
Model	Encoder	Decoder	Channels	β	Decoder Latent	Num. Auxiliary
`Collage` VAE	"28x1,28d4,7x1,7d7,1x1"	"1x4"	128	$[0.5, 0.7, 1.0, 1.2, 1.5]$	64	[30]
VDVAE	"28x1,28d4,7x1,7d7,1x1"	"1x1,7m1,7x2,28m7"	64	$[0.5, 0.7, 1.0, 1.2, 1.5]$	16	–

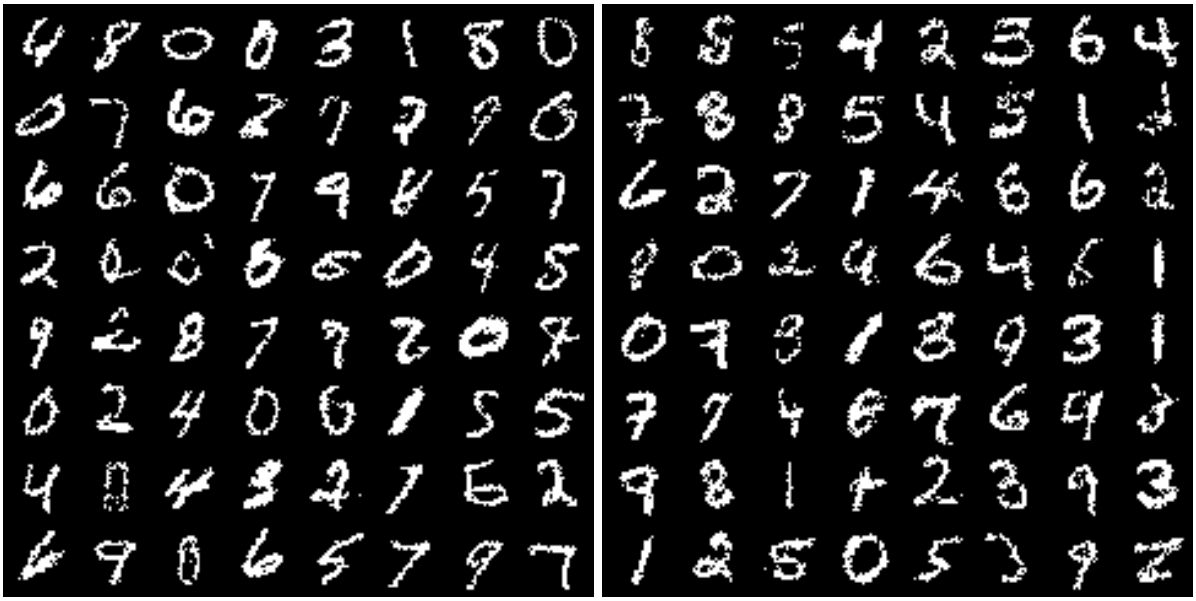
Table 3: Autoencoder model setup for VDVAE and VDCVAE (ours). Both were trained on BMNIST with Bernoulli likelihood. The encoder and decoder architectures strings can be interpreted as follows: "28x1" indicates 1 block of 28 residual layers and "28m7" is the mixing a 7 by 7 activation into an upsampled 28 by 28 embedding. A block consists of standard autoencoder parameterization with a learned prior, posterior and latent projection output layer. The channel dimensions apply to both the widths of encoder and decoder blocks with the collage decoder variants having learned separate maps per channel. β describes the KL-coefficient weighting in the ELBO objective.

	Training				Optimizer		
Model	Batch Size	Epochs	EMA	Per Step (secs)	Learning Rate	Weight Decay	Optimizer
`Collage` VAE (ours)	32	2000	0.	1.7	$1e-4$	0.	AdamW(0.9, 0.9)
VDVAE	32	2000	0.	2.2	$1e-4$	0.	AdamW(0.9, 0.9)

Table 4: Optimization settings for training baseline VDVAE and `Collage` VAE (ours) on dynamically binarized MNIST.

D Additional Results

D.1 Super–Resolution of `Collage` VAE Samples



(a) VDVAE (b) VDCVAE

Figure 10: Super-resolution factor of $1\times$ the original data resolution (28×28 MNIST). **[Left]:** Samples from the VDVAE baseline. **[Right]:** Samples from a `Collage` VAE.

Figure 11: **[Left:]** `Collage` VAE samples with $10\times$ magnification (280×280 resolution). **[Right:]** `Collage` VAE samples with $40\times$ magnification (1120×1120 resolution).

D.2 Compression

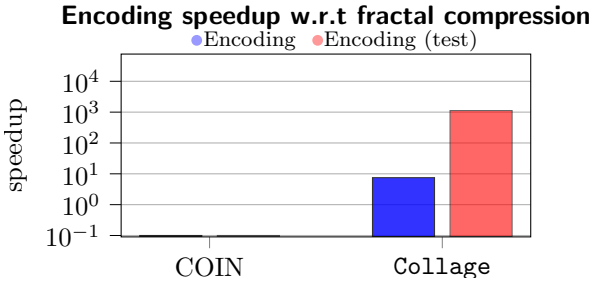
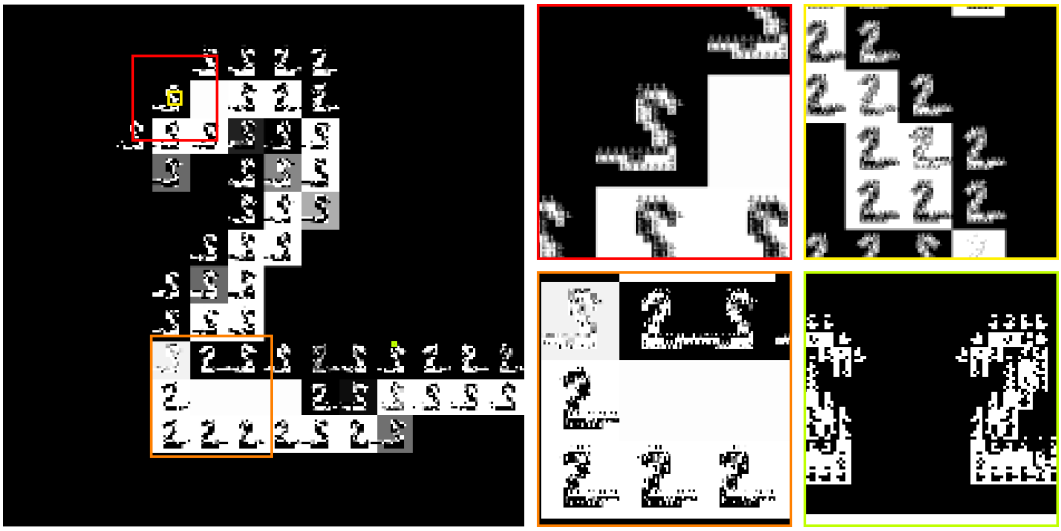


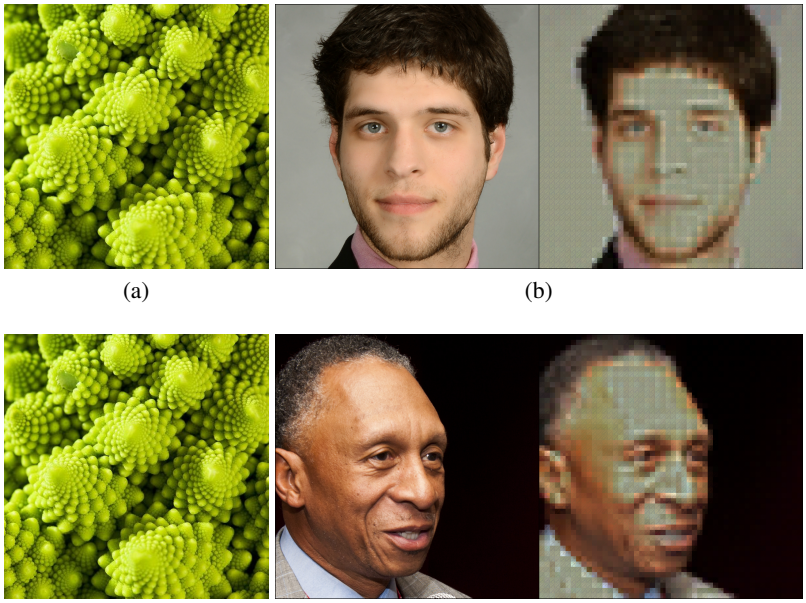
Figure 12: Wall–clock encoding speedups of Neural `Collage` compressors and COIN over a PIFS–based fractal compression implementation on GPU.

D.3 Fractal Stylization



(a) (b)

Figure 13: Fractalized MNIST digit via a `Collage` at $500\times$ the original data resolution. Red box is a $3\times$ magnification, yellow is a $20\times$ magnification, orange is $2\times$ magnification, and lime is $80\times$ magnification. Note that the image is compressed for easier displaying within the pdf. Refer to the code repository for the full quality version.



(a) (b)

(c) (d)

Figure 14: Neural `Collage` texturizer, with relative weights of 0.5 for U (texture source) and 1.0 for D (domain cells) in the `Collage` iteration.