Learning Fractals by Gradient Descent

Cheng-Hao Tu*, Hong-You Chen*, David Carlyn, Wei-Lun Chao

Department of Computer Science and Engineering, The Ohio State University
{tu.343, chen.9301, carlyn.1, chao.209}@osu.edu

Abstract

Fractals are geometric shapes that can display complex and self-similar patterns found in nature (e.g., clouds and plants). Recent works in visual recognition have leveraged this property to create random fractal images for model pre-training. In this paper, we study the inverse problem — *given a target image (not necessarily a fractal), we aim to generate a fractal image that looks like it.* We propose a novel approach that learns the parameters underlying a fractal image via gradient descent. We show that our approach can find fractal parameters of high visual quality and be compatible with different loss functions, opening up several potentials, e.g., learning fractals for downstream tasks, scientific understanding, etc.

Introduction

A fractal is an infinitely complex shape that is self-similar across different scales. It displays a pattern that repeats forever, and every part of a fractal looks very similar to the whole fractal. Fractals have been shown to capture geometric properties of elements found in nature (Mandelbrot and Mandelbrot 1982). In essence, our nature is full of fractals, e.g., trees, rivers, coastlines, mountains, clouds, seashells, etc.

Despite its complex shape, a fractal can be generated via well-defined mathematical systems like iterated function systems (IFS) (Barnsley 2014). An IFS generates a fractal image by "drawing" points iteratively on a canvas, in which the point transition is governed by a small set of 2×2 affine transformations (each with a probability). Concretely, given the current point $\boldsymbol{v}^{(t)} \in \mathbb{R}^2$, the IFS randomly samples one affine transformation with replacement from the set, and uses it to transform $\boldsymbol{v}^{(t)}$ into the next point $\boldsymbol{v}^{(t+1)}$. This stochastic step is repeated until a pre-defined number of iterations T is reached. The collection of $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$ can then be used to draw the fractal image (see Figure 1 for an illustration). *The set of affine transformations thus can be seen as the parameters (or ID) of a fractal.*

Motivated by these nice properties, several recent works in visual recognition have investigated generating a large number of fractal images to facilitate model (pre-)training, in which the parameters of each fractal are randomly sampled (Kataoka et al. 2020; Anderson and Farrell 2022). For instance, Kataoka et al. (2020) sampled multiple sets of parameters and treated each set as a fractal class. Within each class, they leveraged the stochastic nature of the IFS to generate fractal images with intra-class variations. They then used these images to pre-train a neural network in a supervised way without natural images and achieved promising results.

In this paper, we study a complementary and inverse problem — *given a target image that is not necessarily a fractal, can we find the parameters such that the generated IFS fractal image looks like it?* We consider this problem important from at least two aspects. First, it has the potential to find fractal parameters suitable for downstream visual tasks, e.g., for different kinds of image domains. Second, it can help us understand the limitations of IFS fractals, e.g., what kinds of image patterns the IFS cannot generate.

We propose a novel gradient-descent-based approach for this problem (see section 6). Given a target image and the (initial) fractal parameters, our approach compares the image generated by the IFS to the target image, back-propagates the gradients w.r.t. the generated image *through the IFS*, and uses the resulting gradients w.r.t. the parameters to update the fractal parameters. The key insights are three-folded. First, we view an IFS as a special recurrent neural network (RNN) with identity activation functions and stochastic weights. The weights in each recurrent step (i.e., one affine transformation) are not the same, but are sampled with replacement from shared parameters. This analogy allows us to implement the forward (i.e., image generation) and backward (i.e., back-propagation) passes of IFS using deep learning frameworks like PyTorch (Paszke et al. 2019) and TensorFlow (Abadi et al. 2016). Second, the IFS generates a sequence of points, not an image directly. To compare this sequence to the target image and, in turn, calculate the gradients w.r.t. the generated points, we introduce a differentiable rendering module inspired by (Qian et al. 2020). Third, we identify several challenges in optimization and propose corresponding solutions to make the overall learning process stable, effective, and efficient. The learned parameters can thus lead to high-quality fractal images that are visually similar to the target image. We note that besides reconstructing a target image, our approach can easily be repurposed by plugging in other loss functions that return gradients w.r.t. the generated image.

We conduct two experiments to validate our approach. First, given a target image (that is not necessarily a fractal),

*These authors contributed equally.

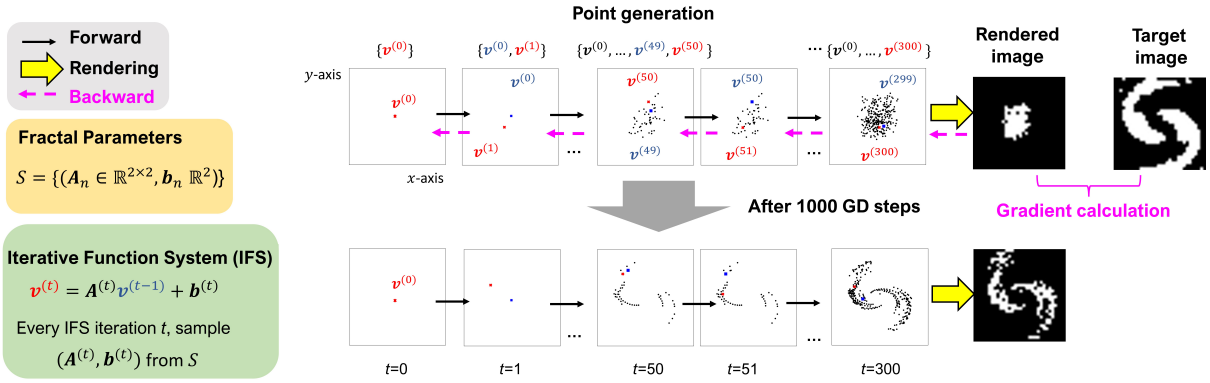


Figure 1: Overview of our approach, which aims to learn the fractal parameters $\mathcal{S}$ so that the generated point sequence by IFS (i.e., $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(300)}\}$), after being rendered onto an image, is close to the target image. We learn $\mathcal{S}$ by gradient descent (GD). As shown, after 1000 GD steps, we can learn high-quality $\mathcal{S}$ to generate images that are visually close to the target one.

we find the fractal parameters to reconstruct it and evaluate our approach using the pixel-wise mean square error (MSE). Our approach notably outperforms the baselines. Second, we extend our approach to finding a set of parameters to approximate a set of target images. This can be achieved by replacing the image-to-image MSE loss with set-to-set losses like the GAN loss (Goodfellow et al. 2014). In this experiment, we demonstrate the potential of finding more suitable fractal images for downstream tasks than the randomly sampled ones (Kataoka et al. 2020; Anderson and Farrell 2022).

Contributions. We propose a gradient-descent-based approach to find the fractal parameters for a given image. Our approach is stable, effective, and efficient. It can be easily implemented using popular deep learning frameworks. Moreover, it can be easily extended from finding fractal parameters for image reconstruction to other purposes, by plugging the corresponding loss functions. While we mainly demonstrate the effectiveness of our approach using image reconstruction, we respect think it is a critical and meaningful step towards unlocking other potential usages and applicability of fractal images in visual recognition and deep learning. The code is provided at https://github.com/andytu28/LearningFractals.

Related Work

Fractal inversion. Finding the parameters of a given fractal image is a long-standing open problem (Vrscay 1991; Iacus and La Torre 2005; Kya and Yang 2001; Guérin and Tosan 2005; Graham and Demers 2019). Many approaches are based on genetic algorithms (Lankhorst 1995; Gutiérrez, Cofiño, and Ivanissevich 2000; Nettleton and Garigliano 1994) or moment matching (Vrscay and Roehrig 1989; Forte and Vrscay 1995). Our approach is novel by tackling the problem via gradient descent and can be easily implemented using deep learning frameworks. In this paper, we mainly focus on image reconstruction in the image/pixel space.

Fractal applications. Recent works leveraged fractals to generate geometrically-meaningful synthetic data to facilitate neural network training (Hendrycks et al. 2021; Baradad Jurjo et al. 2021; Ma et al. 2021; Anderson and Farrell 2022; Kataoka et al. 2020; Gupta, O'Connell, and Egger 2021; Nakashima et al. 2021). Some other works leveraged self-similarity for image compression, for example, by partitioned IFS (Fisher 1994), which resembles an image with other parts of it (Pandey and Singh 2015; AL-Bundi and Abd 2020; Jacquin et al. 1992; Al-Bundi, Al-Saidi, and Al-Jawari 2016; Venkatasekhar, Aruna, and Parthiban 2013; Menassel, Nini, and Mekhaznia 2018; Poli et al. 2022). Fractals have also been used in scientific fields, like diagnostic imaging (Karperien et al. 2008), 3D modeling (Ullah et al. 2021), optical systems (Sweet, Ott, and Yorke 1999), biology (Otaki 2021; Enright and Leitner 2005), etc. In this paper, we study the inverse problem, which potentially benefits these applications.

Fractals and neural networks. Fractals also inspire neural network designs (Larsson, Maire, and Shakhnarovich 2016; Deng, Yu, and Long 2021) and help the understanding of their underlying mechanisms (Camuto et al. 2021; Dym, Sober, and Daubechies 2020). Specifically, some earlier works (Kolen 1994; Tino and Dorffner 1998; Tino, Cernansky, and Benuskova 2004) used IFS formulations to understand the properties of recurrent neural networks (RNNs). These works are starkly different from ours. First, we focus on fractal inversion. Second, we formulate an IFS as a special RNN to facilitate solving the inverse problem via gradient descent. We note that Stark (1991) formulated an IFS as sparse feed-forward networks not for solving the inverse problem.

Inversion of generative models. Our problem is related to the inversion of generative models, such as GANs (Zhu et al. 2016; Xia et al. 2021). The goal is to find the latent representation of GANs for any given image. Our problem has some unique challenges since the fractal generation process is stochastic and not directly end-to-end differentiable.

Background

We first provide the background about fractal generation via Iterated Function Systems (IFS) (Barnsley 2014). As briefly mentioned in section , an IFS can be thought of as "drawing"

Algorithm 1: IFS generation process.
See section 6 for details.

Input : Fractal parameter $\mathcal{S}$, # of iterations T, an initial point $\boldsymbol{v}^{(0)} \in \mathbb{R}^2$

for $t \leftarrow 1$ **to** T **do**
  Sample $(\boldsymbol{A}^{(t)}, \boldsymbol{b}^{(t)})$ from $\mathcal{S}$
  $\boldsymbol{v}^{(t)} = \boldsymbol{A}^{(t)}\boldsymbol{v}^{(t-1)} + \boldsymbol{b}^{(t)}$
end

Output : $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$

Algorithm 2: IFS generation process via the FE layer.
See subsection 6 for details.

Input : Fractal embedding $\{\boldsymbol{S}_{\mathrm{A}}, \boldsymbol{S}_{\mathrm{b}}\}$, # of iterations T, an initial point $\boldsymbol{v}^{(0)} \in \mathbb{R}^2$

Sample index sequence $\boldsymbol{z} = [z^{(1)}, \cdots, z^{(t)}]$;
for $t \leftarrow 1$ **to** T **do**
  $\boldsymbol{v}^{(t)} = \mathtt{mat}(\boldsymbol{S}_{\mathrm{A}}[:, z^{(t)}])\boldsymbol{v}^{(t-1)} + \boldsymbol{S}_{\mathrm{b}}[:, z^{(t)}]$;
end

Output : $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$

points iteratively on a canvas. The point transition is governed by a small set of 2×2 affine transformations $\mathcal{S}$:

$$\mathcal{S} = \left\{ \left(\boldsymbol{A}_n \in \mathbb{R}^{2\times 2}, \boldsymbol{b}_n \in \mathbb{R}^2, p_n \in \mathbb{R} \right) \right\}_{n=1}^{N}. \quad (1)$$

Here, an $(\boldsymbol{A}_n, \boldsymbol{b}_n)$ pair specifies an affine transformation:

$$\boldsymbol{A}_n \boldsymbol{v} + \boldsymbol{b}_n, \quad (2)$$

which transforms a 2D position $\boldsymbol{v} \in \mathbb{R}^2$ to a new position. The p_n in Equation 1 is the corresponding probability of each transformation, i.e., $\sum_n p_n = 1$ and $p_n \geq 0, \forall n \in [N]$.

Given $\mathcal{S}$, an IFS generates an image as follows. Starting from an initial point $\boldsymbol{v}^{(0)} \in \mathbb{R}^2$, an IFS repeatedly samples a transformation $(\boldsymbol{A}^{(t)}, \boldsymbol{b}^{(t)})$ from $\{(\boldsymbol{A}_n, \boldsymbol{b}_n)\}_{n=1}^{N}$ with replacement following the probability $\{p_n\}_{n=1}^{N}$, and applies

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}^{(t)}\boldsymbol{v}^{(t-1)} + \boldsymbol{b}^{(t)} \quad (3)$$

to arrive at the next 2D point. This stochastic process can continue forever, but in practice, we set a pre-defined number of iterations T. The traveled points $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$ are then used to synthesize a fractal image. For example, one can quantize them into integer pixel coordinates and render each pixel as a binary (i.e., 1) value on a black (i.e., 0) canvas. We summarize the IFS fractal generation process in Algorithm 1. Due to the randomness in sampling $(\boldsymbol{A}^{(t)}, \boldsymbol{b}^{(t)})$, one $\mathcal{S}$ can create different but geometrically-similar fractal images. We call $\mathcal{S}$ the fractal parameters.

Simplified parameters. We follow (Kataoka et al. 2020; Anderson and Farrell 2022) to set p_n by the determinant of the corresponding $\boldsymbol{A}_n$. That is, $p_n \propto |\mathtt{det}(\boldsymbol{A}_n)|$. In the following, we will ignore p_n when defining $\mathcal{S}$:

$$\mathcal{S} = \{(\boldsymbol{A}_n, \boldsymbol{b}_n)\}_{n=1}^{N}. \quad (4)$$

End-to-End Learnable Fractals by Gradient Descent

In this paper, we study the problem: *given a target image, can we find the parameters $\mathcal{S}$ such that the generated IFS fractal image looks like it?* Let us first provide the problem definition and list the challenges, followed by our algorithm and implementation.

Problem and Challenges

Problem. Given a gray-scaled image $\boldsymbol{I} \in [0,1]^{H\times W}$, where H is the height and W is the width, we want to find the fractal parameters $\mathcal{S}$ (cf. Equation 4) such that the generated IFS fractal image $\boldsymbol{I}' \in [0,1]^{H\times W}$ is close to $\boldsymbol{I}$. We note that $\boldsymbol{I}$ may not be a fractal image generated by an IFS.

Let us denote the IFS generation process by $\boldsymbol{I}' = G(\mathcal{S})$, and the image-to-image loss function by $\mathcal{L}$. This problem can be formulated as an optimization problem:

$$\min_{\mathcal{S}} \mathcal{L}(G(\mathcal{S}), \boldsymbol{I}). \quad (5)$$

We mainly consider the pixel-wise square loss $\mathcal{L}(\boldsymbol{I}', \boldsymbol{I}) = \|\boldsymbol{I}' - \boldsymbol{I}\|_2^2$, but other loss functions can be easily applied. *We note that G is stochastic, and we will discuss how to deal with it in subsection 6.*

Goal. We aim to solve Equation 5 via gradient descent (GD) — as shown in Equation 3, each iteration in IFS is differentiable. Specifically, we aim to develop our algorithm as a module such that it can be easily implemented by modern deep learning frameworks like PyTorch and TensorFlow. This can largely promote our algorithm's usage, e.g., to combine with other modules for end-to-end training.

It is worth mentioning that while $\boldsymbol{I}'$ depends on $\{p_n\}_{n=1}^{N}$ and $\{p_n\}_{n=1}^{N}$ depends on $\{\boldsymbol{A}_n\}_{n=1}^{N}$ (cf. Equation 4), we do not include this path in deriving the gradients w.r.t. $\{\boldsymbol{A}_n\}_{n=1}^{N}$.

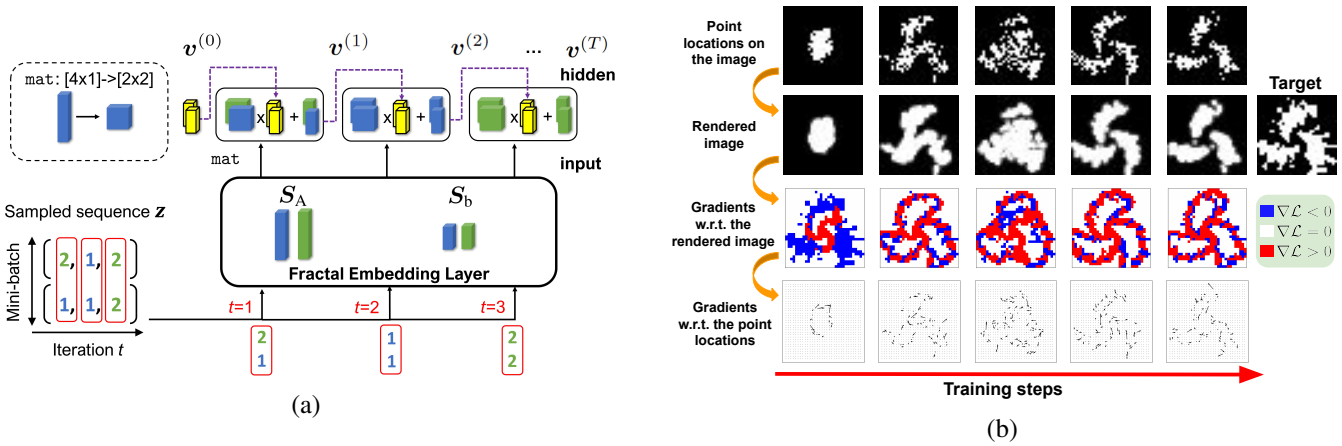
Challenges. We identify challenges in achieving our goal.

- First, it is nontrivial to implement and parallelize the IFS generation process (for multiple fractals) using modern deep learning frameworks.
- Second, the whole IFS generation process is not immediately differentiable. Specifically, an IFS generates a sequence of points, not directly an image. We need to back-propagate the gradients from an image to the points.
- Third, there are several technical details to make the optimization effective and stable. For example, an IFS is stochastic. The IFS generation involves hundreds or thousands of iterations (i.e., T); the back-propagation would likely suffer from exploding gradients.

In the rest of this section, we describe our solution to each challenge. We give an overview of our approach in Figure 1.

Fractal Generation as a Deep Learning Model

IFS as an RNN. We start with how to implement the IFS fractal generation process as a deep learning model. We note that an IFS performs Equation 3 for T iterations. Every iteration involves an affine transformation, which is sampled with replacement from the fractal parameters $\mathcal{S}$. Let us assume



(a) (b)

Figure 2: (a) Fractal embedding layer in subsection 6. We show a case of $T = 3$ and batch size $= 2$. The purple dashed lines indicate the point transition (like the hidden vector in RNNs). (b) Differentiable rendering in subsection 6. We show that the gradients w.r.t. the rendered image can be translated into 2D gradient vectors w.r.t. the IFS points, i.e., $\nabla_{\boldsymbol{v}^{(t)}}\mathcal{L} \in \mathbb{R}^2, \forall t \in \{1, \cdots, T\}$ (zoom-in for better resolutions).

that $\mathcal{S}$ contains only one pair $(\boldsymbol{A}, \boldsymbol{b})$, then an IFS can essentially be seen as a special recurrent neural network (RNN), which has an identity activation function but no input vector:

$$\boldsymbol{v}^{(t)} = \boldsymbol{A}\boldsymbol{v}^{(t-1)} + \boldsymbol{b}. \tag{6}$$

Here, $\boldsymbol{v}^{(t)}$ is the RNN's hidden vector. That is, an IFS can be implemented like an RNN if it contains only one transformation. In the following, we extend this notion to IFS with multiple transformations.

IFS rearrangement. To begin with, let us rearrange the IFS generation process. Instead of sampling an affine transformation from $\mathcal{S}$ at each of the T IFS iterations, we choose to sample all T of them before an IFS starts. We can do so because the sampling is independent of the intermediate results of an IFS. Concretely, according to $\{p_n\}_{n=1}^N$, we first sample an index sequence $\boldsymbol{z} = [z^{(1)}, \cdots, z^{(t)}]$, in which $z^{(t)} \in \{1, \cdots, N\}$ denotes the index of the sampled affine transformation from $\mathcal{S}$ at iteration t. With the pre-sampled $\boldsymbol{z}$, we can perform an IFS in a seemingly deterministic way. This is reminiscent of the re-parameterization trick commonly used in generative models (Kingma and Welling 2013).

Fractal embedding layer. We now introduce a *fractal embedding* (FE) layer, which leverages the rearrangement to implement an IFS like an RNN. The FE layer has two sets of parameters, $\boldsymbol{S}_\text{A} \in \mathbb{R}^{4\times N}$ and $\boldsymbol{S}_\text{b} \in \mathbb{R}^{2\times N}$, which record all the parameters in $\mathcal{S}$ (cf., Equation 4). Specifically, the n^th column of $\boldsymbol{S}_\text{A}$ (denoted as $\boldsymbol{S}_\text{A}[:, n]$) records the 2×2 parameters of $\boldsymbol{A}_n$; i.e., $\boldsymbol{A}_n = \texttt{mat}(\boldsymbol{S}_\text{A}[:, n])$, where $\texttt{mat}$ is a reshaping operation from a column vector into a matrix. The n^th column of $\boldsymbol{S}_\text{b}$ (denoted as $\boldsymbol{S}_\text{b}[:, n]$) records $\boldsymbol{b}_n$, i.e., $\boldsymbol{b}_n = (\boldsymbol{S}_\text{b}[:, n])$. Let us define one more notation $\mathbf{1}_{z^{(t)}} \in \mathbb{R}^N$, which is an N-dimensional one-hot vector whose $z^{(t)\text{th}}$ element is 1. With these parameters, the FE layer carries out the IFS computation at iteration t (cf. Equation 3) as follows

$$\begin{aligned}
&\text{FE}(z^{(t)}, \boldsymbol{v}^{(t-1)}; \{\boldsymbol{S}_\text{A}, \boldsymbol{S}_\text{b}\}) \\
&= \texttt{mat}(\boldsymbol{S}_\text{A}\mathbf{1}_{z^{(t)}})\boldsymbol{v}^{(t-1)} + \boldsymbol{S}_\text{b}\mathbf{1}_{z^{(t)}} \\
&= \texttt{mat}(\boldsymbol{S}_\text{A}[:, z^{(t)}])\boldsymbol{v}^{(t-1)} + \boldsymbol{S}_\text{b}[:, z^{(t)}] \\
&= \boldsymbol{A}^{(t)}\boldsymbol{v}^{(t-1)} + \boldsymbol{b}^{(t)}.
\end{aligned} \tag{7}$$

That is, the FE layer takes the index of the sampled affine transformation of iteration t (i.e., $z^{(t)}$) and the previous point $\boldsymbol{v}^{(t-1)}$ as input, and outputs the next point $\boldsymbol{v}^{(t)}$. To perform an IFS for T iterations, we simply call the FE layer recurrently for T times. This analogy between IFS and RNNs makes it easy to implement the IFS generation process in modern deep learning frameworks. Please see Algorithm 2 for a summary.

Extension to mini-batch versions. The FE layer can easily be extended into a mini-batch version to generate multiple fractal images, if they share the same fractal parameters $\mathcal{S}$. See Figure 2 (a) for an illustration. Interestingly, even if the fractal images are based on different fractal parameters, a simple parameter concatenation trick can enable mini-batch computation. Let us consider a mini-batch of size 2 with fractal parameters $\mathcal{S}_1$ and $\mathcal{S}_2$. We can combine $\mathcal{S}_1$ and $\mathcal{S}_2$ into one $\boldsymbol{S}_\text{A} \in \mathbb{R}^{4\times(2N)}$ and one $\boldsymbol{S}_\text{b} \in \mathbb{R}^{2\times(2N)}$, where the number of columns in $\boldsymbol{S}_\text{A}$ and $\boldsymbol{S}_\text{b}$ are doubled to incorporate the affine transformations in $\mathcal{S}_1$ and $\mathcal{S}_2$. When sampling the index sequence for each fractal, we just need to make sure that the indices align with the columns of $\boldsymbol{S}_\text{A}$ and $\boldsymbol{S}_\text{b}$ to generate the correct fractal images.

Differentiable Rendering

The FE layer enables us to implement an IFS via modern deep learning frameworks, upon which we can enjoy automatic back-propagation to calculate the gradients $\nabla_{\mathcal{S}}\mathcal{L}$ (cf. Equation 5) if we have obtained the gradients $\nabla_{\boldsymbol{v}^{(t)}}\mathcal{L}, \forall t \in \{1, \cdots, T\}$. However, obtaining the latter is not trivial. An IFS generates a sequence $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$ of 2D points, but it is not immediately clear how to render an image $\boldsymbol{I}'$ such that subsequently, we can back-propagate the gradient $\nabla_{\boldsymbol{I}'}\mathcal{L}(\boldsymbol{I}', \boldsymbol{I})$ to obtain $\nabla_{\boldsymbol{v}^{(t)}}\mathcal{L}$.

Concretely, $\nabla_{\boldsymbol{I}'}\mathcal{L}$ is a 2D tensor of the same size as $\boldsymbol{I}'$, where $\nabla_{\boldsymbol{I}'[h,w]}\mathcal{L} \in \mathbb{R}$ indicates if the pixel $[h, w]$ of $\boldsymbol{I}'$ should get brighter or darker. Let us follow the definition in section 6: IFS draws points on a black canvas. Then if $\nabla_{\boldsymbol{I}'[h,w]}\mathcal{L} \in \mathbb{R} < 0$, ideally we want to have some IFS points closer to or located at $[h, w]$, to make $\boldsymbol{I}'[h, w]$ brighter. To do so, however, requires a translation of $\nabla_{\boldsymbol{I}'[h,w]}\mathcal{L} < 0$ into a "pulling" force towards $\boldsymbol{I}'[h, w]$ for some of the points in $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$.

To this end, we propose to draw the IFS points $\{\boldsymbol{v}^{(0)}, \cdots, \boldsymbol{v}^{(T)}\}$[1] onto $\boldsymbol{I}'$ using an RBF kernel — every point "diffuses" its mass to nearby pixels. This is inspired by the differentiable soft quantization module (Qian et al. 2020) proposed for 3D object detection. The rendered image I' is defined as

$$\boldsymbol{I}'[h, w] = \sum_t \exp(-\left\|[h, w]^\top - \boldsymbol{v}^{(t)}\right\|_2^2 / \tau), \quad h \in [H], w \in [W], \tag{8}$$

where $\tau > 0$ governs the kernel's bandwidth. This module not only is differentiable w.r.t. $\boldsymbol{v}^{(t)}$, but also fulfills our requirement. The closer the IFS points to $[h, w]$ are, the larger the intensity of $\boldsymbol{I}'[h, w]$ is. Moreover, if $\nabla_{\boldsymbol{I}'[h,w]}\mathcal{L} < 0$, then $\frac{\partial \mathcal{L}}{\partial \boldsymbol{I}'[h,w]} \times \frac{\partial \boldsymbol{I}'[h,w]}{\partial \boldsymbol{v}^{(t)}}$ will be a 2D vector positively proportional to $(\boldsymbol{v}^{(t)} - [h, w]^\top)$. When GD is applied, this will pull $\boldsymbol{v}^{(t)}$ towards $[h, w]^\top$. See Figure 2 (b) for an illustration. Since $\boldsymbol{I}'[h, w]$ may have a value larger than 1, we clamp it back to the range $[0, 1]$ by $\min(\boldsymbol{I}'[h, w], 1)$.

Objective and Optimization

In subsection 6 and subsection 6, we present an end-to-end differentiable framework for learning $\mathcal{S}$. In this subsection, we discuss the objective function and present important technical insights to stabilize the optimization.

Objective function. As pointed out in the subsequent paragraph of Equation 5, since an IFS is stochastic, the objective function in Equation 5 is ill-posed. To tackle this, let us rewrite the stochastic $G(\mathcal{S})$ by $G(\mathcal{S}; \boldsymbol{z})$, in which we explicitly represent the stochasticity in $G(\mathcal{S})$ by $\boldsymbol{z}$: the sampled index sequence according to $\mathcal{S}$ (see subsection 6). We then present two objective functions:

$$\begin{aligned} \textbf{Expectation: } & \mathbb{E}_{\boldsymbol{z}\sim\mathcal{S}}\left[\mathcal{L}(G(\mathcal{S}, \boldsymbol{z}), \boldsymbol{I})\right], \\ \textbf{Fixed: } & \mathbb{E}_{\boldsymbol{z}\in\mathcal{Z}}\left[\mathcal{L}(G(\mathcal{S}, \boldsymbol{z}), \boldsymbol{I})\right], \end{aligned} \tag{9}$$

where $\boldsymbol{z} \sim \mathcal{S}$ indicates that $\boldsymbol{z}$ is sampled according to $\mathcal{S}$; $\mathcal{Z}$ is a set of pre-sampled fixed sequences. For instance, we can sample $\mathcal{Z}$ according to the initial $\mathcal{S}$ and fix it throughout the optimization process. The Expectation objective encourages every $\boldsymbol{z} \sim \mathcal{S}$ to generate $\boldsymbol{I}$. The learned $\mathcal{S}$ thus would easily generate images $\boldsymbol{I}'$ close to $\boldsymbol{I}$, but the diversity among $\boldsymbol{I}'$ might be limited. The Fixed objective is designed to alleviate this limitation by only forcing a (small) set of fixed sequences $\mathcal{Z}$ to generate $\boldsymbol{I}$. However, since $\mathcal{Z}$ is pre-sampled, their probability of being sampled from the learned $\mathcal{S}$ might be low. Namely, it might be hard for the learned $\mathcal{S}$ to generate images like $\boldsymbol{I}$. We evaluate both in section 6.

[1]Following (Anderson and Farrell 2022), we linearly translate and re-scale the points to fit them into the $H \times W$ image plane.

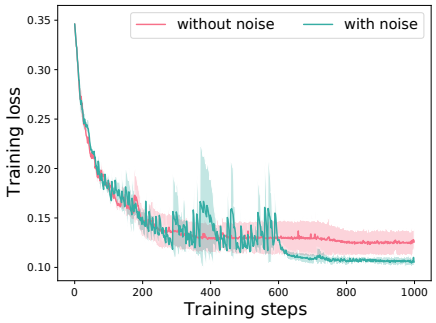


Figure 3: The training loss along the SGD steps w/ and w/o adding random noise to fractal parameters $\mathcal{S}$. We add a Gaussian noise $\sim \mathcal{N}(0, 0.1)$ to each element of $\mathcal{S}$ every 5 steps. This makes the final loss lower and the optimization robust. The shaded area is the standard deviation over random seeds.

Optimization. We apply mini-batch stochastic gradient descent (SGD) to optimize the objectives w.r.t. $\mathcal{S}$ in Equation 9. To avoid confusion, we use "iteration" for IFS generation, and "step" for SGD optimization. The term "stochastic" refers to sampling a mini-batch of $\boldsymbol{z}$ at every optimization step from either $\mathcal{S}$ or $\mathcal{Z}$, generating the corresponding fractal images $\boldsymbol{I}'$ by IFS, and calculating the average gradients w.r.t. $\mathcal{S}$.

We note that, unlike neural networks that are mostly over-parameterized, $\mathcal{S}$ has much fewer, usually tens of parameters. This somehow makes the optimization harder. Specifically, we found that the optimization easily gets stuck at poor local minima. (In contrast, over-parameterized models are more resistant to it (Du et al. 2019; Allen-Zhu, Li, and Liang 2019; Arora et al. 2019).) To resolve this, whenever we calculate the stochastic gradients, we add a small Gaussian noise to help the current parameters escape local minima, inspired by (Welling and Teh 2011). Figure 3 shows that this design stabilizes the optimization process and achieves lower loss.

Reparameterized $\mathcal{S}$. In the extreme case where $\mathcal{S}$ contains only one affine transformation $(\boldsymbol{A}, \boldsymbol{b})$, recurrently applying it multiple times could easily result in exploding gradients when the maximum singular value of $\boldsymbol{A}$ is larger than 1 (Pascanu, Mikolov, and Bengio 2013). This problem happens when $\mathcal{S}$ contains multiple affine transformations as well, if most of the matrices have such a property. To avoid it, we follow (Anderson and Farrell 2022) to decompose "each" $\boldsymbol{A}$ in $\mathcal{S}$ into rotation, scaling, and flipping matrices

$$\boldsymbol{A} = \underbrace{\begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{bmatrix}}_{\text{scaling}} \underbrace{\begin{bmatrix}\cos(\phi) & -\sin(\phi)\\ \sin(\phi) & \cos(\phi)\end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix}d_1 & 0\\ 0 & d_1\end{bmatrix}}_{\text{flipping}}, \tag{10}$$

where $\sigma_1, \sigma_2 \geq 0; d_1, d_2 \in \{-1, 1\}$. That is, $\boldsymbol{A}$ is *reparameterized* by $\{\theta, \phi, \sigma_1, \sigma_2, d_1, d_2\}$. We note that Anderson and Farrell (2022) used this decomposition to sample $\boldsymbol{A}$. The purpose was to generate diverse fractal images for pre-training. Here, we use this decomposition for a drastically different purpose: to stabilize the optimization of $\boldsymbol{A}$. We leverage one important property — $\max(\sigma_1, \sigma_2)$ is exactly the maximum singular value of $\boldsymbol{A}$ — and clamp both σ_1 and σ_2 into $[0, 1]$ to avoid exploding gradients.

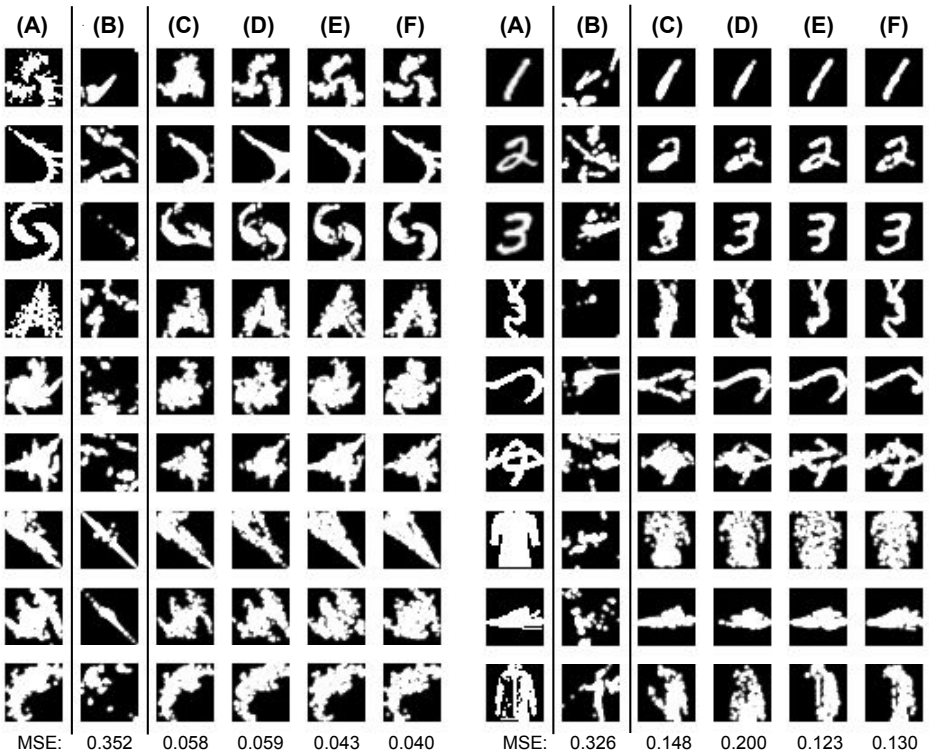


Figure 4: Image reconstruction examples. Left: FractalDB images. Right: MNIST (row 1-3), KMNIST (row 4-6), and FMNIST images (row 7-9). (A): the target images. (B): an inverse encoder baseline. (C-F): variants of our approach (corresponding to the "Index" column in Table 1). The last row shows two examples and the corresponding MSE.

Index	Method	Objective	Clamp	Noisy	FractalDB	MNIST	KMNIST	FMNIST
(B)	Inverse encoder	-	-	-	0.3889 ± 0.0000	0.1724 ± 0.0000	0.3016 ± 0.0000	0.3658 ± 0.0000
(C)	Ours	Fixed	✗	✗	0.0790 ± 0.0006	0.0475 ± 0.0012	0.1023 ± 0.0016	0.0692 ± 0.0010
(D)		Fixed	✓	✗	0.0752 ± 0.0027	0.0312 ± 0.0002	0.0979 ± 0.0008	0.0765 ± 0.0034
(E)		Expectation	✓	✗	0.0630 ± 0.0008	0.0223 ± 0.0004	0.0826 ± 0.0010	0.0547 ± 0.0005
(F)		Expectation	✓	✓	$\mathbf{0.0617 \pm 0.0009}$	$\mathbf{0.0202 \pm 0.0005}$	$\mathbf{0.0781 \pm 0.0015}$	$\mathbf{0.0538 \pm 0.0006}$
	Ours (w/o Reparam. $\mathcal{S}$)	Expectation	✓	✓	Exploded	Exploded	Exploded	Exploded

Table 1: Averaged MSE (mean±std over 3 runs) of image reconstruction on FractalDB/MNIST/KMNIST/FMNIST dataset.

Extension

Besides the pixel-wise square loss $\mathcal{L}(\boldsymbol{I}', \boldsymbol{I}) = \|\boldsymbol{I}' - \boldsymbol{I}\|_2^2$, our approach can easily be compatible with other loss functions on $\boldsymbol{I}'$. The loss may even come from a downstream task. In subsection 6, we experiment with one such extension. Given a set of images $\{\boldsymbol{I}_m\}_{m=1}^M$, we aim to learn a set of fractal parameters $\{\mathcal{S}_j\}_{j=1}^J$ such that their generated images are distributionally similar to $\{\boldsymbol{I}_m\}_{m=1}^M$. We apply a GAN loss (Goodfellow et al. 2014), which relies on training a discriminator to tell real images from generated images.

Experiments

We conduct two experiments to validate our approach. These include (1) **image reconstruction:** learning an IFS fractal to reconstruct each given target image by gradient descent, using the mean squared error on pixels; (2) **learning fractals with a GAN loss:** given a set of target images, extending our approach to learn a set of fractal parameters that generates images distributionally similar to the targets.

We note that *we do not intend to compete with other state-of-the-art in these tasks that do not involve fractals*. Our experiments are to demonstrate that our learnable fractals can capture meaningful information in the target images. *Please see the supplementary material for more details and analyses.*

Setups

Datasets. We first reconstruct random fractal images generated following **FractalDB** (Kataoka et al. 2020; Anderson and Farrell 2022). We then consider images that are not generated by fractals, including **MNIST** (hand-written digits) (LeCun et al. 1998), **FMNIST** (fashion clothing) (Xiao,

Rasul, and Vollgraf 2017), and **KMNIST** (hand-written characters) (Clanuwat et al. 2018). For reconstruction, we use 32×32 binarized images as targets. For the GAN extension, we use the training/test splits in the three non-fractal datasets.

Technical details about fractals. Throughout the experiments, we sample $\boldsymbol{v}^{(0)}$ and initialize $\mathcal{S}$ by the random procedure in (Anderson and Farrell 2022). The number of transformations in $\mathcal{S}$ is fixed to $N = 10$, and we set $T = 300$ per image. These apply to the FractalDB generation as well. To learn $\mathcal{S}$ using our approach, we set $\tau = 1$ (RBF kernel bandwidth) and apply an Adam optimizer (Kingma and Ba 2015).

Image Reconstruction and Ablation Study

In the image reconstruction task, our goal is to verify our gradient descent-based method can effectively recover target shapes. Specifically, given a target image, we learn an $\mathcal{S}$ using our approach to reconstruct the target shape by minimizing the **mean squared error (MSE)** between the generated and the target images. Each $\mathcal{S}$ is learned with a batch size 50 and a learning rate 0.05 for $1K$ SGD steps. As we would recover any given target, no training and test splits are considered. We generate $2K$ target images using FractalDB, and randomly sample $2K$ target images from each MNIST/FMNIST/KMNIST. After $\mathcal{S}$ is learned, we generate 100 images per $\mathcal{S}$ (with different sampled sequences $\boldsymbol{z}$) and report the minimum MSE to reduce the influence of random sequences. We consider the following variants of our method:

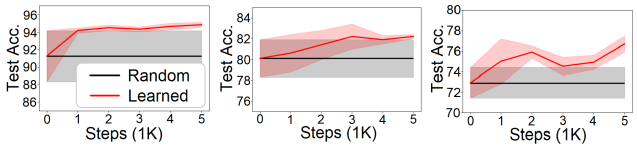
- Using the *Expectation* objective or the *Fixed* objective discussed in subsection 6.
- In differentiable rendering (see Equation 8), *clamping* each pixel to be within $[0, 1]$ or not.
- Applying the *noisy* SGD trick or not (see subsection 6).
- Learning the reparameterized $\mathcal{S}$ or the original IFS parameters $\mathcal{S}$ (see subsection 6).

Besides our variants, we include an **inverse encoder** baseline similar to (Graham and Demers 2019) that learns a ResNet18 (He et al. 2016) to predict the fractal parameters $\mathcal{S}$ given an image (i.e., as a regression problem). We train such a regressor on $20K$ different FractalDB images (using the corresponding $\mathcal{S}$ as labels) and use it on the target images.

Quantitatively, the averaged MSE in Table 1 verifies our design choices of applying the Expectation objective, clamping pixel values, and using noisy SGD. Notably, without reparameterization, we can hardly learn $\mathcal{S}$ since the gradients quickly explode. Qualitatively, Figure 4 shows some reconstruction examples. Our approach works well on fractal shapes. Surprisingly, we can also recover non-fractal images to a certain extent. This reveals that IFS can be more expressive if the parameters are learned in a proper way.

Learning Fractals as GAN Generators

As discussed in subsection 6, our approach is a flexible deep learning module that can be learned with other losses besides MSE. To demonstrate such flexibility, we learn fractals like GANs (Goodfellow et al. 2014) by formulating a set-to-set learning problem — matching the distributions of images generated from fractals to the distributions of (real) target images.



(a) MNIST (b) FMNIST (c) KMNIST

Figure 5: Learning fractals as GANs. We conduct linear evaluations using features pre-trained (for $0 \sim 5K$ SGD steps) on the learned/random fractals. Test accuracy of means (lines) and standard deviations (shades) over 10 runs are plotted.

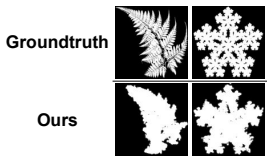


Figure 6: Inversion on complex (fractal) images.

Specifically, we start from 100 randomly sampled fractals $\mathcal{S}$ and train them on the target training set of MNIST/FMNIST/KMINIST in an unsupervised way using the GAN loss with a discriminator (a binary LeNet (LeCun et al. 1998) classifier). Next, we evaluate the learned fractals by using them for pre-training (Kataoka et al. 2020).

We argue that if the learned fractals $\mathcal{S}$ can produce image distributions closer to the targets, pre-training on the generated images should provide good features w.r.t. the (real) target images. We expect pre-training on the images generated from the learned $\mathcal{S}$ to yield better features for the target datasets, compared to pre-training on random fractals.

Concretely, we pre-train a 100-way classifier (using the LeNet architecture again) from scratch. We generate 600 images from each $\mathcal{S}$ and treat each $\mathcal{S}$ as a pseudo-class following (Kataoka et al. 2020) to form a 100-class classification problem. Then, we conduct a linear evaluation on the pre-trained features using the real training and test sets.

Figure 5 shows the results, in which we compare pre-training on the random fractals (gray lines) and on the learned fractals (red lines), after every $1K$ SGD steps. As we pre-train $\mathcal{S}$ with more steps, the learned fractals more accurately capture the target image distribution, evidenced by the better pre-trained features that lead to higher test accuracy. This study shows the helpfulness of our method for pre-training.

Conclusion, Limitations, and Future Work

We present a gradient descent-based method to find the fractal parameters for a given image. Our approach is stable, effective, and can be easily implemented using popular deep learning frameworks. It can also be extended to finding fractal parameters for other purposes, e.g., to facilitate downstream vision tasks, by plugging in the corresponding losses.

We see some limitations yet to be resolved and leave them as our future work. For example, IFS itself cannot handle color images. Our current approach is hard to recover very detailed structures (e.g., Figure 6). We also plan to further investigate our approach on 3D point clouds.

Acknowledgments

This research is supported in part by grants from the National Science Foundation (IIS-2107077, OAC-2118240, and OAC-2112606), the OSU GI Development funds, and Cisco Systems, Inc. We are thankful for the generous support of the computational resources by the Ohio Supercomputer Center and AWS Cloud Credits for Research. We thank all the feedback from the review committee and have incorporated them.

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 265–283.

AL-Bundi, S. S.; and Abd, M. S. 2020. A Review on Fractal Image Compression Using Optimization Techniques. *Journal of Al-Qadisiyah for computer science and mathematics*, 12(1): Page–38.

Al-Bundi, S. S.; Al-Saidi, N. M.; and Al-Jawari, N. J. 2016. Crowding optimization method to improve fractal image compressions based iterated function systems. *International Journal of Advanced Computer Science and Applications*, 7(7): 392–401.

Allen-Zhu, Z.; Li, Y.; and Liang, Y. 2019. Learning and generalization in overparameterized neural networks, going beyond two layers. *Advances in neural information processing systems*, 32.

Anderson, C.; and Farrell, R. 2022. Improving Fractal Pre-Training. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 1300–1309.

Arora, S.; Du, S.; Hu, W.; Li, Z.; and Wang, R. 2019. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, 322–332. PMLR.

Baradad Jurjo, M.; Wulff, J.; Wang, T.; Isola, P.; and Torralba, A. 2021. Learning to see by looking at noise. *Advances in Neural Information Processing Systems*, 34: 2556–2569.

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

Camuto, A.; Deligiannidis, G.; Erdogdu, M. A.; Gurbuzbalaban, M.; Simsekli, U.; and Zhu, L. 2021. Fractal structure and generalization properties of stochastic optimization algorithms. *Advances in Neural Information Processing Systems*, 34.

Clanuwat, T.; Bober-Irizar, M.; Kitamoto, A.; Lamb, A.; Yamamoto, K.; and Ha, D. 2018. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*.

Deng, Z.; Yu, H.; and Long, Y. 2021. Fractal Pyramid Networks. *arXiv preprint arXiv:2106.14694*.

Du, S. S.; Zhai, X.; Poczos, B.; and Singh, A. 2019. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*.

Dym, N.; Sober, B.; and Daubechies, I. 2020. Expression of fractals through neural network functions. *IEEE Journal on Selected Areas in Information Theory*, 1(1): 57–66.

Enright, M. B.; and Leitner, D. M. 2005. Mass fractal dimension and the compactness of proteins. *Physical Review E*, 71(1): 011912.

Fisher, Y. 1994. Fractal image compression. *Fractals*, 2(03): 347–361.

Forte, B.; and Vrscay, E. 1995. Solving the inverse problem for measures using iterated function systems: A new approach. *Advances in applied probability*, 27(3): 800–820.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems*, 27.

Graham, L.; and Demers, M. 2019. Applying Neural Networks to a Fractal Inverse Problem. In *International Conference on Applied Mathematics, Modeling and Computational Science*, 157–165. Springer.

Guérin, E.; and Tosan, E. 2005. Fractal inverse problem: approximation formulation and differential methods. In *Fractals in Engineering*, 271–285. Springer.

Gupta, S.; O'Connell, T. P.; and Egger, B. 2021. Beyond Flatland: Pre-training with a Strong 3D Inductive Bias. *arXiv preprint arXiv:2112.00113*.

Gutiérrez, J. M.; Cofiño, A. S.; and Ivanissevich, M. L. 2000. An hybrid evolutive-genetic strategy for the inverse fractal problem of IFS models. In *Advances in Artificial Intelligence*, 467–476. Springer.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Hendrycks, D.; Zou, A.; Mazeika, M.; Tang, L.; Song, D.; and Steinhardt, J. 2021. PixMix: Dreamlike Pictures Comprehensively Improve Safety Measures. *arXiv preprint arXiv:2112.05135*.

Iacus, S. M.; and La Torre, D. 2005. Approximating distribution functions by iterated function systems. *Journal of Applied Mathematics and Decision Sciences*, 2005(1): 33–46.

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

Karperien, A.; Jelinek, H. F.; Leandro, J. J.; Soares, J. V.; Cesar Jr, R. M.; and Luckie, A. 2008. Automated detection of proliferative retinopathy in clinical practice. *Clinical ophthalmology (Auckland, NZ)*, 2(1): 109.

Kataoka, H.; Okayasu, K.; Matsumoto, A.; Yamagata, E.; Yamada, R.; Inoue, N.; Nakamura, A.; and Satoh, Y. 2020. Pre-training without natural images. In *Proceedings of the Asian Conference on Computer Vision*.

Kingma, D. P.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

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

Kolen, J. F. 1994. Recurrent networks: State machines or iterated function systems. In *Proceedings of the 1993 Connectionist Models Summer School*, 203–210. Erlbaum Associates Hillsdale.

Kya, B.; and Yang, Y. 2001. Optimization of fractal iterated function system (IFS) with probability and fractal image generation. *International Journal of Minerals, Metallurgy and Materials*, 8(2): 152–156.

Lankhorst, M. M. 1995. *Iterated function systems optimization with genetic algorithms*. University of Groningen, Department of Mathematics and Computing Science.

Larsson, G.; Maire, M.; and Shakhnarovich, G. 2016. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Ma, Y.; Hua, Y.; Deng, H.; Song, T.; Wang, H.; Xue, Z.; Cao, H.; Ma, R.; and Guan, H. 2021. Self-Supervised Vessel Segmentation via Adversarial Learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 7536–7545.

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

Menassel, R.; Nini, B.; and Mekhaznia, T. 2018. An improved fractal image compression using wolf pack algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(3): 429–439.

Nakashima, K.; Kataoka, H.; Matsumoto, A.; Iwata, K.; and Inoue, N. 2021. Can vision transformers learn without natural images? *arXiv preprint arXiv:2103.13023*.

Nettleton, D. J.; and Garigliano, R. 1994. Evolutionary algorithms and a fractal inverse problem. *Biosystems*, 33(3): 221–231.

Otaki, J. M. 2021. The Fractal Geometry of the Nymphalid Groundplan: Self-Similar Configuration of Color Pattern Symmetry Systems in Butterfly Wings. *Insects*, 12(1): 39.

Pandey, A.; and Singh, A. 2015. Fractal Image Compression using Genetic Algorithm with Variants of Crossover. *International Journal of Electrical, Electronics and Computer Engineering*, 4(1): 73–81.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, 1310–1318. PMLR.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Poli, M.; Xu, W.; Massaroli, S.; Meng, C.; Kim, K.; and Ermon, S. 2022. Self-Similarity Priors: Neural Collages as Differentiable Fractal Representations. *arXiv preprint arXiv:2204.07673*.

Qian, R.; Garg, D.; Wang, Y.; You, Y.; Belongie, S.; Hariharan, B.; Campbell, M.; Weinberger, K. Q.; and Chao, W.-L. 2020. End-to-end pseudo-lidar for image-based 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 5881–5890.

Stark, J. 1991. Iterated function systems as neural networks. *Neural Networks*, 4(5): 679–690.

Sweet, D.; Ott, E.; and Yorke, J. A. 1999. Topology in chaotic scattering. *Nature*, 399(6734): 315–316.

Tino, P.; Cernansky, M.; and Benuskova, L. 2004. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1): 6–15.

Tino, P.; and Dorffner, G. 1998. Recurrent neural networks with iterated function systems dynamics. In *IFAC Symposium on Neural Computation*.

Ullah, A.; D'Addona, D. M.; Seto, Y.; Yonehara, S.; and Kubo, A. 2021. Utilizing Fractals for Modeling and 3D Printing of Porous Structures. *Fractal and Fractional*, 5(2): 40.

Venkatasekhar, D.; Aruna, P.; and Parthiban, B. 2013. Fast search strategies using optimization for fractal image compression. *International Journal of Computer and Information Technology*, 2(3): 437–441.

Vrscay, E. R. 1991. Iterated function systems: theory, applications and the inverse problem. In *Fractal geometry and analysis*, 405–468. Springer.

Vrscay, E. R.; and Roehrig, C. J. 1989. Iterated function systems and the inverse problem of fractal construction using moments. In *Computers and Mathematics*, 250–259. Springer.

Welling, M.; and Teh, Y. W. 2011. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 681–688. Citeseer.

Xia, W.; Zhang, Y.; Yang, Y.; Xue, J.-H.; Zhou, B.; and Yang, M.-H. 2021. GAN inversion: A survey. *arXiv preprint arXiv:2101.05278*.

Xiao, H.; Rasul, K.; and Vollgraf, R. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *ArXiv*, abs/1708.07747.

Zhu, J.-Y.; Krähenbühl, P.; Shechtman, E.; and Efros, A. A. 2016. Generative visual manipulation on the natural image manifold. In *European conference on computer vision*, 597–613. Springer.