2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) | 978-1-6654-0915-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/WACV51458.2022.00247

Improving Fractal Pre-training

Connor Anderson
Brigham Young University
connor.anderson@byu.edu

Ryan Farrell
Brigham Young University
farrell@cs.byu.edu

Abstract

The deep neural networks used in modern computer vision systems require enormous image datasets to train them. These carefully-curated datasets typically have a million or more images, across a thousand or more distinct categories. The process of creating and curating such a dataset is a monumental undertaking, demanding extensive effort and labelling expense and necessitating careful navigation of technical and social issues such as label accuracy, copyright ownership, and content bias.

What if we had a way to harness the power of large image datasets but with few or none of the major issues and concerns currently faced? *This paper extends the recent work of Kataoka* et al. *[15], proposing an improved pre-training dataset based on dynamically-generated fractal images. Challenging issues with large-scale image datasets become points of elegance for fractal pre-training: perfect label accuracy at zero cost; no need to store/transmit large image archives; no privacy/demographic bias/concerns of inappropriate content, as no humans are pictured; limitless supply and diversity of images; and the images are free/open-source. Perhaps surprisingly, avoiding these difficulties imposes only a small penalty in performance. Leveraging a newly-proposed pre-training task—multi-instance prediction—our experiments demonstrate that fine-tuning a network pre-trained using fractals attains 92.7-98.1% of the accuracy of an ImageNet pre-trained network. Our code is publicly available.*[1]

1. Introduction

One of the leading factors in the improvement of computer vision systems over the last decade has been the access to ever-expanding datasets that can be used for pre-training deep learning models. Nearly all state-of-the-art systems these days have been trained on millions, tens-of-millions, or even hundreds-of-millions of images. Collecting, labeling, managing, and distributing these datasets requires monumental effort—indispensable effort—to achieve the power found in today's models. However, the list of challenges and concerns around using these datasets is growing as well. Along with technical challenges and high costs, there have been questions of privacy, ownership, inappropriate content, and unfair bias (for example, see [3, 32]). Clearly there are complex issues that still need to be overcome, and many of them elude simple solutions.

What if we had a way to harness the power of large image datasets with few or none of the major issues and concerns currently faced? In this paper, we discuss the possibility of using abstract, computer-generated fractal images to pre-train modern computer vision models. We expand on the work of Kataoka et al. [15], from whom we take our inspiration. There are several distinct advantages to using fractal images for pre-training:

- Fractals are complex geometric structures that often emerge from a very small set of parameters or equations; thus, as images, they are highly compressible—often a handful of bytes is sufficient to describe them, along with a generic program for rendering them. Therefore, **the need to store and transmit large datasets of image files can be circumvented**.
- Since the data is synthetic, **we get labels for free**: no massive manual labeling effort is required.
- Since fractals are abstract geometric objects, **there are no issues surrounding the depictions of people**: concerns about privacy, inappropriate content, and biases related to gender, race, or any other human factor can be laid to rest as far as the pre-training data is concerned.
- Fractals are "free and open-source": they are defined by fairly simple mathematics and anyone can produce the images with only a few dozen lines of code. Thus, there are no issues surrounding copyrights and ownership of images. **Anyone can use fractal-generated data to train models for any purpose, commercial or otherwise**.
- Fractals provide **a near-limitless supply of diverse images**. Small changes to selected parameters can result in entirely new datasets.

[1] catalys1.github.io/fractal-pretraining/

978-1-6654-0915-5/21/$31.00 ©2021 IEEE
DOI 10.1109/WACV51458.2022.00247

- In some cases, fractal images can be very efficient to render. In fact, with the right approach, fractal images can be generated on-the-fly, fast enough to keep up with the throughput of neural network training—even when using fairly large batch sizes and distributed training on multiple GPUs. This both **eliminates the need to generate and store a fixed set of data** up front, and **removes the disk-IO bottleneck** that subsequently can become a problem while reading large volumes of data from persistent storage.

A few of the preceding claims are obviously true by virtue of the nature of the data. We show the remaining claims to be true by analysis and experimental evaluation in this paper. The remaining question, then, is how well do fractal-image pre-trained models perform on real-world, natural-image tasks? We show that, while not yet as good as ImageNet pre-training in most cases, the gap is not as large as you might expect.

We emphasize that we are not the first to propose pre-training with fractal images. Kataoka et al. [15] recently introduced the idea, and showed that it was a viable approach. Our work builds on some of the core ideas from their paper—particularly, the use of a large set of randomly sampled affine Iterated Function Systems (IFS) for generating training data. We make several fairly significant deviations from their approach, however, and show that these deviations make a significant difference in the results obtained. Figure 1 gives a high-level view of our approach.

The core contributions of our paper can be summarized as follows:

- We propose a novel, principled approach for sampling IFS codes (see Section 3.2). Our approach leads to highly efficient sampling of large numbers of codes (fractal parameters), simultaneously improving the quality of the resulting fractal images, leading to more effective representation learning.
- We introduce large-scale multi-instance (multi-label) prediction as a pre-training task/method (see Section 4.1), and show that it is more effective for fractal pre-training than normal multi-class classification, as evaluated on a set of natural-image recognition tasks.
- We show that using fractal images generated with color and backgrounds (see Section 3.1) for pre-training leads to better transfer learning (fine-tuning).
- We show that fractal-image pre-training can be quite effective when transferred to tasks with limited training data, such as fine-grained visual categorization and medical image segmentation (see Section 5).
- We show that we can use "just-in-time" image generation during training, without ever needing to create and store a database of images beforehand (see Section 4.2).

2. Related Work

Over the past decade, "ImageNet pre-training" has become an integral part of training computer vision models. The default process is to train a model to perform supervised classification on the 1,000-class ImageNet dataset [30] and then fine-tune the model on a different dataset, which has proven to be very effective [14, 18]. Recent work has attempted to probe the limits of this "supervised classification for pre-training" approach, showing that huge amounts of data can improve the pre-training transfer performance [33]—even when the labels are only weakly associated with the image content [25]. Large-scale domain-specific datasets, such as iNaturalist [36], have also proven effective for pre-training [5]; large-scale, weakly-labeled data [19] has also proven surprisingly effective. Other work has suggested that pre-training isn't the best approach in some domains, such as COCO [24] object detection [12, 39]; for many problems, particularly those with limited data, however, pre-training provides a substantial performance boost.

For some domains, it can be challenging or impossible to collect and/or annotate enough images to sufficiently pre-train a model. One way to address this issue is to use synthetically generated data [28]. Such data can be generated using 3D models [34] or generative models [4, 35]. Usually the data is modeled after natural images and real-world objects.

In contrast to using synthetic data modeled after real-world images, Kataoka *et al*. [15] recently proposed the use of fractal images for pre-training. Fractal images are both synthetic and abstract, though they have some similarity to fractal structures in nature. Fractals have been admired for their visual complexity and beauty, and can be used to create beautiful artwork [6]; but they have also found practical application in image compression [10], and have even inspired neural network architectures [23]. Farmer [9] provides a detailed treatment of applications of fractals in computer vision. Dym *et al*. [7] contributed a recent study of piecewise-linear functions generated by fractal IFSs and their similarity to those generated by deep neural networks. In another interesting recent contribution, Marasca *et al*. [27] utilize fractals to assess dataset classification complexity. Early work used fractal principles for texture segmentation [17], and Kocic [17] discussed how fractals could be used to model natural forms.

In this work, we build on [15] and use fractal images to pre-train visual recognition models.

3. Fractal Images

Fractal images are generally produced by iteration of a simple formula. For example, the well known Mandelbrot and Julia sets can be generated from the equation

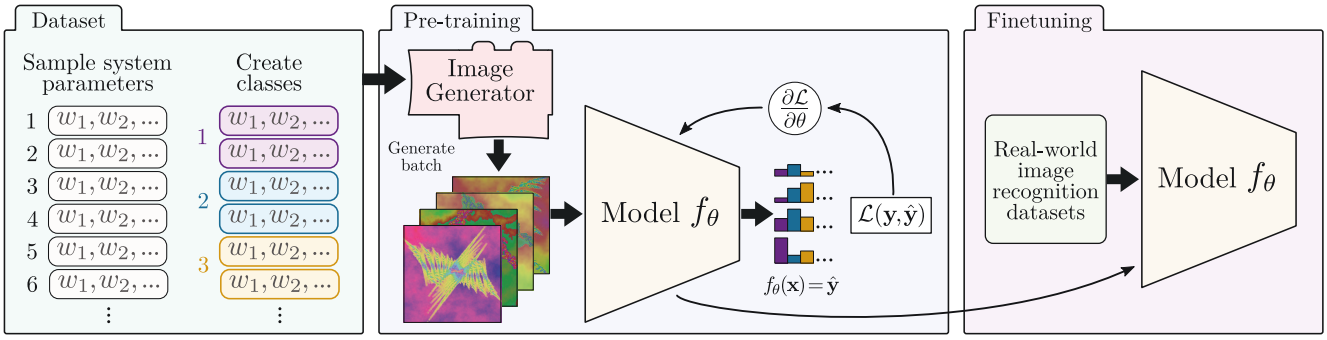

Figure 1. Fractal pre-training. We generate a dataset of IFS codes (fractal parameters), which are used to generate images on-the-fly for pre-training a computer vision model, which can then be fine-tuned for a variety of real-world image recognition tasks.

$z_{k+1} = z_k^2 + c$, for $z, c \in \mathbb{C}$, by treating pixel coordinates as points in the complex plane and iterating until z "escapes" toward infinity or remains bounded for some number of iterations. Information about whether or not the point escaped, and how long it took to do so, can be used to color the pixels, revealing rich complexity.

An Iterated Function System (IFS) can also be used to generate fractal images. An IFS consists of a set of two or more functions and an associated set of probabilities. The set of functions, which we refer to as a *system* or a *code*, have an associated *attractor*—a set of points with a particular geometric structure—such that iterative application of the functions in the system will bring points in the associated space onto the attractor. Sec. 3.1 describes how fractal images can be rendered from IFS codes. These images exhibit complex patterns and self-similarity.

As proposed in [15], we use affine Iterated Function Systems to create a dataset of fractal images for pre-training computer vision models. In an affine IFS, the functions in the system are affine transformations: they consist of a linear function, represented by a matrix A, and a translation vector $\mathbf{b}$, so that $w(x) = A\mathbf{x} + \mathbf{b}$. Affine functions have several advantages: particularly, it is easy to evaluate whether they are contractive functions (which is generally necessary for IFS) and they are simple and fast to evaluate numerically.

Iterated Function Systems We now provide a more formal definition of Iterated Function Systems. An IFS code $\mathcal{S}$ defined on a complete metric space $\mathcal{X}$ (we will assume that the metric space is $\mathcal{X} = (\mathbb{R}^2, \|\cdot\|_2)$) is a set of transformations $w_i : \mathcal{X} \to \mathcal{X}$ and their associated probabilities p_i:

$$\mathcal{S} = \{(w_i, p_i) : i = 1, 2, \ldots, N\}, \tag{1}$$

which satisfy the average contractivity condition

$$\prod_{i=1}^{N} s_i^{p_i} < 1, \tag{2}$$

where s_i is the Lipschitz constant for w_i. The *attractor* $\mathcal{A}_\mathcal{S}$ is a unique geometric structure [2], a subset of $\mathcal{X}$ defined by $\mathcal{S}$. The shape of $\mathcal{A}_\mathcal{S}$ depends on the functions w_i and not the probabilities p_i; however, the p_i do affect the distribution of points across $\mathcal{A}_\mathcal{S}$[2]. We choose $p_i \propto |\det A_i|$, as done in [15]. We provide a visual comparison between determinant-proportional and uniform p_i in Appendix D.4.

3.1. Rendering Fractal Images

We can render an approximation of $\mathcal{A}_\mathcal{S}$ to obtain a fractal image. The random iteration method, or "chaos game", can be used to to generate a subset of $\mathcal{A}_\mathcal{S}$ as follows: choose an initial point $\mathbf{x}_0 \in \mathcal{X}$; randomly choose w_i with probability p_i and apply it to $\mathbf{x}_0$ to get $\mathbf{x}_1 = w_i(\mathbf{x}_0)$; repeat this process for a sufficiently large number of iterations K to get the set of points $\hat{\mathcal{A}} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K\} \subseteq \mathcal{A}_\mathcal{S}$. The larger K is, the closer the approximation will be to $\mathcal{A}_\mathcal{S}$.

The next step is to render the points in $\hat{\mathcal{A}}$ to an image X. We map a rectangular region $\mathcal{R} \in \mathbb{R}^2$, nominally defined by the min and max x and y values in $\hat{\mathcal{A}}$, to an $M \times M$ pixel grid. Pixels can be rendered as binary values, indicating that at least one point in $\hat{\mathcal{A}}$ maps to that pixel; or they can be rendered as continuous values, indicating the density of points that map to each pixel.

3.2. Sampling Iterated Function Systems

3.2.1 What Makes a "Good" IFS Code?

So far we have defined Iterated Function Systems and how we use them to render fractal images. We now turn our attention to the question of how to get the IFS codes in the first place. Let $N = |\mathcal{S}|$, the number of functions in the code $\mathcal{S}$. In [15], they choose codes by sampling $N \sim U(\{2, 3, \ldots, 8\})$[3], and then sampling the six values $(A_k, \mathbf{b}_k)$ for each of the N affine transformations from

[2] See [1] for a more thorough introduction to Iterated Function Systems.

[3] Throughout the paper, we use $U(a, b)$ to mean a continuous uniform distribution on the interval $[a, b]$, and $U(\{\cdot\})$ to mean a discrete uniform distribution over elements of the set $\{\cdot\}$.

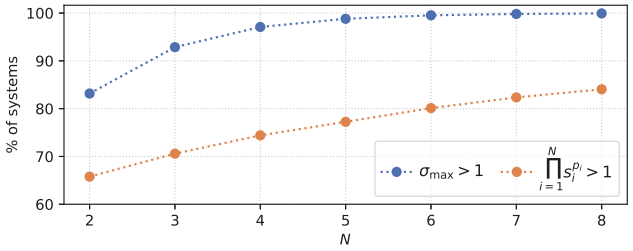

Figure 2. For systems with $N = 2, \ldots, 8$ transforms, we show the percentage of systems (out of 100,000 randomly sampled) that have their largest singular value greater than 1 (in red), and also those which violate average contractivity (in blue: $p_i \propto \det(A_i)$, where A_i is the linear part of the affine transform).

$U(-1, 1)$. Repeating this process thousands of times produces a dataset of IFS codes. Each code can then be treated as a "class", for the purpose of doing supervised multi-class pre-training.

There are a few problems with the sampling approach taken in [15]. First, in order to be a true IFS, the system must be a contraction (Eq. 2). Sampling random transforms with values between -1 and 1 does not guarantee a contraction. In fact, the majority of codes thus sampled will not be. To demonstrate this, we sampled 100,000 random systems with parameters in $U(-1, 1)$ for each of $N = 2, \ldots, 8$. Figure 2 shows (i) in blue, the percentage of those systems that had *at least* one singular value greater than 1—an affine transform must have singular values less than 1 to be a contraction, as we describe shortly—and (ii) in red, the percentage that violate the average contractivity condition. Clearly, a naive sampling approach is quite inefficient, as the majority of systems will not be contractions, and when a system is not contractive it will, under iteration, produce sequences that diverge to infinity. Such sequences cause numerical difficulties and yield unsatisfactory images.

The second problem is that even when a system is a contraction, it might not produce fractals with "good" geometric properties. What do we mean by "good" geometric properties? Consider the fractal images shown in Figure 3. Those on the left are very sparse, consisting of mostly blank space and perhaps a few small structures (note that this is similar to the idea of "filling rate" as discussed in [15]). Those on the right look like blurry smudges. The ones in the middle, however, contain complex and varied structure. This set is the most visually interesting; we hypothesize (and experimentally validate in Section 5.2) that it is the most useful for representation learning.

3.2.2 Effectively Sampling IFS Codes

We will now describe an approach to sampling IFS codes that addresses the two concerns just raised (contractivity and good geometry). Our approach is based on the Singular Value Decomposition (SVD) of A_k, the linear part of the affine transform. First, in order to ensure that an IFS is contractive, it is sufficient to ensure that each function w_i is a contraction; that is, it satisfies

$$\|w_i(\mathbf{x}_1) - w_i(\mathbf{x}_2)\| \leq \|\mathbf{x}_1 - \mathbf{x}_2\|, \ \forall\, \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X} \quad (3)$$

For affine functions $w_i(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^2 = A\mathbf{x}+\mathbf{b}$, we require

$$\begin{aligned} & \|A\mathbf{x}_1 + \mathbf{b} - A\mathbf{x}_2 - \mathbf{b}\|_2 \leq \|\mathbf{x}_1 - \mathbf{x}_2\|_2 \\ \Rightarrow \quad & \frac{\|A(\mathbf{x}_1 - \mathbf{x}_2)\|_2}{\|\mathbf{x}_1 - \mathbf{x}_2\|_2} \leq 1 \\ \Rightarrow \quad & \sigma_{\max}(A) \leq 1 \end{aligned} \quad (4)$$

where $\sigma_{\max}(A)$ denotes the maximum singular value of A [4]. Thus, it is sufficient to ensure that the singular values of A are less than 1, which we can do by construction. Recall that by the Singular Value Decomposition, $A = U\Sigma V^T$, where U and V are orthogonal matrices and Σ is a diagonal matrix containing the singular values of A, σ_1 and σ_2, ordered by decreasing magnitude. Since U and V are orthogonal, they act as rotation matrices (with possible reflection, *i.e.* the determinant can be ± 1). Let $U = \mathcal{R}_\theta$ be a rotation by angle θ, and let $V^T = \mathcal{R}_\phi$ be a rotation by angle ϕ. Let D be a diagonal matrix with diagonal elements $d_1, d_2 \in \{-1, 1\}$. Then $A = U\Sigma V^T = \mathcal{R}_\theta \Sigma \mathcal{R}_\phi D$. We can sample A by appropriately sampling $(\theta, \phi, \sigma_1, \sigma_2, d_1, d_2)$, composing the corresponding matrices, as above, and then multiplying them together to obtain A. By sampling σ_1 and σ_2 in the range $(0, 1)$, we ensure that the system is a contraction.

Sampling the SVD parameters directly guarantees a contraction mapping, however, it does not address the question of good geometry. It's not immediately clear what the relationship between the fractal geometry and the system parameters is, nor whether there is a simple and concise relationship at all. Intuition, however, hinted that the singular values might play an important role. The magnitudes of the singular values define how quickly an affine contraction map converges to its fixed point under iteration; small singular values would lead to quick collapse, while singular values near 1 would be more likely to result in "wandering" trajectories which don't converge to a clear geometric structure. Closer investigation revealed that there is indeed a correlation between some property of the singular values and whether the resulting fractal possesses good geometry. To probe this relationship, we labeled by hand several hundred size-2 systems according to whether they had good geometry or not (subjectively), and fit a linear Support Vector Machine (SVM) classifier to the labels using the singular values of the system as features. The SVM was able to distinguish between the systems with

[4] The final line of Eq. 4 follows from the definition of the spectral norm, or l_2 operator norm: $\|A\|_2 = \sup_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \sigma_{\max}(A)$

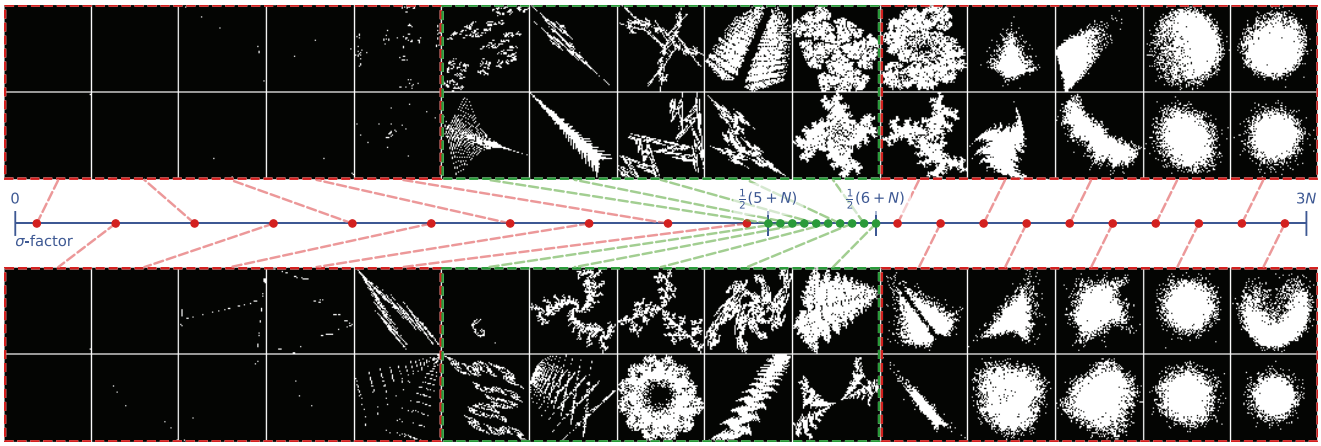

Figure 3. Fractal systems by σ-factor (see Eq. 5). IFS codes with a σ-factor in the range $[\frac{1}{2}(5+N), \frac{1}{2}(6+N)]$ (where N is the size of the system) tend to yield images with good geometry, while many codes with a σ-factor outside this range yield images with degenerate geometry. Images were generated randomly at each σ-factor, with $N = 2$

nearly perfect accuracy. We repeated this experiment several times for different system sizes. As we then investigated the decision boundaries learned by the classifiers, we discovered a simple and general pattern. Let $\sigma_{i,1}$ and $\sigma_{i,2}$ be the singular values for A_i, the ith function in the system, and let $\mathbf{x}_\sigma = \begin{bmatrix}\sigma_{1,1} & \sigma_{1,2} & \dots & \sigma_{N,1} & \sigma_{N,2}\end{bmatrix}^T$, and $\mathbf{w}_\sigma = \begin{bmatrix}1 & 2 & \dots & 1 & 2\end{bmatrix}^T$. We find that a large majority of the systems with good geometry satisfy $\alpha_l \leq \mathbf{w}_\sigma^T \mathbf{x}_\sigma \leq \alpha_u$; in other words, confining the weighted sum of a system's singular values

$$\alpha = \sum_{i=1}^{N} (\sigma_{i,1} + 2\sigma_{i,2}) \tag{5}$$

to the appropriate range (α_l, α_u) will produce systems with mostly good geometry, while systems outside of that range tend to have less desirable geometry. We refer to the quantity α in Eq. 5 as the σ-factor of the system. The appropriate range (α_l, α_u) depends on N, the size of the system; but empirically, we discovered that setting $\alpha_l = \frac{1}{2}(5+N)$ and $\alpha_u = \frac{1}{2}(6+N)$ works very well for $N = 2, \dots, 8$—although a wider range might also work. Figure 3 shows the effect of sampling images at different σ-factors.

We now know that we can confidently tell whether a system will have good geometry by looking at its σ-factor. We next describe an algorithm to randomly sample a set of singular values, $(\sigma_{1,1}, \sigma_{1,2}, \dots, \sigma_{N,1}, \sigma_{N,2})$ that satisfy the conditions

$$0 \leq \sigma_{i,2} \leq \sigma_{i,1} \leq 1 \tag{6}$$

and Eq. 5 for some $0 \leq \alpha \leq 3N$. We take an iterative approach, sampling one singular value at a time and updating the constraints on the next one accordingly. We start with $\sigma_{1,1}$; it could achieve its smallest possible value if every other were maximized. Assume that every other singular value was maximized according to Eq. 6, then we have $\alpha = \sigma_{1,1} + 2\sigma_{1,2} + 3(N-1)$, and the lower bound on $\sigma_{1,1}$ is $\max(0, \frac{1}{3}(\alpha - 3(N-1)))$. Similarly, $\sigma_{1,1}$ could achieve its maximum value when all others are minimized, so we set them to 0 and get that the upper bound on $\sigma_{1,1}$ is $\min(1, \alpha)$. We then sample $\sigma_{1,1}$ uniformly according to the bounds just established, and it becomes a constant in all further bounds calculations. We follow this same process for all but the last two singular values, calculating upper and lower bounds for—and then sampling—each singular value in turn, and updating the constraints on future values. For the last pair, it is more convenient to first sample $\sigma_{N,2}$, at which point $\sigma_{N,1}$ becomes fixed in order to satisfy Eq. 5.

The sampling constraints given by Eqs. 5 and 6 lead to a problem that resembles a Linear Program, except that instead of trying to find a minimizing point, we need to sample a point on the surface of the resulting $2N$-polytope. The algorithm described above does this by iteratively sampling a value independently in each dimension, restricting the available sampling volume for the remaining dimensions. This approach does not necessarily sample points uniformly across the volume, but it is fast and should be sufficient to sample a diverse set of IFS codes.

We now have a process, described formally as `sample-svs` in Alg. 1 (in Supplementary Material, Appendix A), for sampling singular values so that the resulting systems exhibit good geometry. Our algorithm using this process to sample IFS codes via SVD is described as `sample-system` in Alg. 2 (in Supplementary Material, Appendix A).

4. Pre-training Procedures

4.1. Pre-training Tasks

We are focused on using fractal images to learn good representations which will benefit natural-image recognition tasks via fine-tuning. There are several pre-training tasks that could be used to learn these representations. We could use unsupervised/self-supervised pre-training strategies as described earlier in related work (Section 2), however, these seem less compelling when labels are both accurate and abundant. Supervised multi-class classification is routinely used for pre-training on ImageNet, and is the pre-training task adopted in [15]. We too utilize this widely-accepted approach; we additionally propose a new pre-training method which is uniquely suited to synthetically-generated data such as fractal images, a task which we call *multi-instance prediction*. Multi-instance prediction is a type of large-scale multi-label classification, where each image may contain examples of multiple classes. We describe each of these approaches in greater detail below.

4.1.1 Multi-class Classification

For multi-class classification, we follow the same basic approach taken in [15]. We choose a fixed number of classes C, and assign IFS codes to each class. We use the standard cross-entropy objective function to train the model to predict the corresponding class for each image.

Assigning classes to IFS codes Kataoka *et al*. [15] assign each IFS code its own class label, and then augment each class by scaling each of the six parameters in $(A_k, \mathbf{b}_k)$ individually, essentially yielding a set of codes for each class. In principle, these codes will be related, since there is a smoothness to the space of fractals defined by IFS. Small perturbations to the parameters can still yield large differences in the resulting fractals, however (see example in Appendix D); and simply scaling the parameters of the affine transformations may additionally cause the singular values of the system to become too large or too small, producing sparse or degenerate images.

Our approach is to sample more systems according to Algorithm 2, and assign a single class label to a *group* of IFS codes. In our experiments, we show that this approach outperforms the parameter-scaling method of [15]. However, we also experiment with some parameter augmentation methods and find that they can still help performance.

4.1.2 Multi-instance Prediction

Multi-instance classification is a supervised classification task; like multi-class classification, we define a fixed number of classes C and assign one or more IFS codes to each class. But unlike multi-class classification, the images we use in the multi-instance setting may contain multiple fractal instances from multiple classes—hence "multi-instance". During training, the model performs multi-label prediction, trying to determine the presence or absence of each of the C classes in each image. In other words, each class can be considered as an attribute, and the model tries to predict which attributes are present.

Multi-instance prediction is significantly more challenging than multi-class classification, as evidenced in our experiments. Each image contains a variable number of fractals, yielding a vast space of possible image configurations. In training, the model must learn to pay attention to each fractal "attribute" that is present. Our experiments show that this added task complexity leads to pre-trained features that generalize significantly better for downstream tasks.

The images for training multi-instance classification models looks different than for regular multi-class, the process for generating them is different, and there are some special considerations that need to be accounted for. We discuss this in Section 4.2.2.

We train multi-instance models using a binary cross-entropy loss, averaged across all the classes. Since the number of positive examples in each image is so small compared to number of classes (e.g. 5 out of 1000), we apply a large weight to the positive examples to balance the loss. For instance, when using 1000 classes and a maximum of 5 instances per image, we multiply the loss of the positive classes in each image by 200. Without applying this weighting factor, the model fails to learn.

4.2. Pre-training Datasets

Each pre-training task operates on different types of images: single-fractal images for multi-class classification, and multi-fractal images for multi-instance prediction. For both tasks, images are not generated or stored up-front; images are generated "just-in-time", as needed during training.

With the correct procedure, we are able to *generate* all images "on the fly" as they're needed for training. This is significant, as we circumvent the typical need to store or transmit a huge quantity of data. The entire dataset can be generated from the set of IFS codes, which can be stored in tens or hundreds of megabytes (depending on the number and size of the systems). For example, an ImageNet-sized (1.28M images) fractal dataset requires only 184.5MB to store its parameters instead of the 150GB of storage needed to store ImageNet (ILSVRC 20212). We leverage an efficient Numba [22] implementation, along with a *rendering cache* of recently generated images, in order to achieve the necessary throughput during training. Please see Appendix B.2 in the Supplementary Material for details.

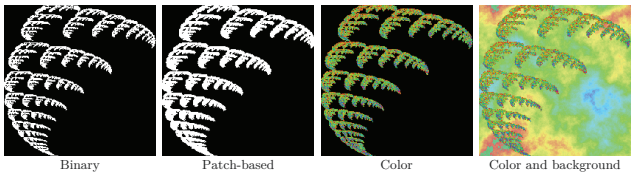

Figure 4. Rendered fractal images.

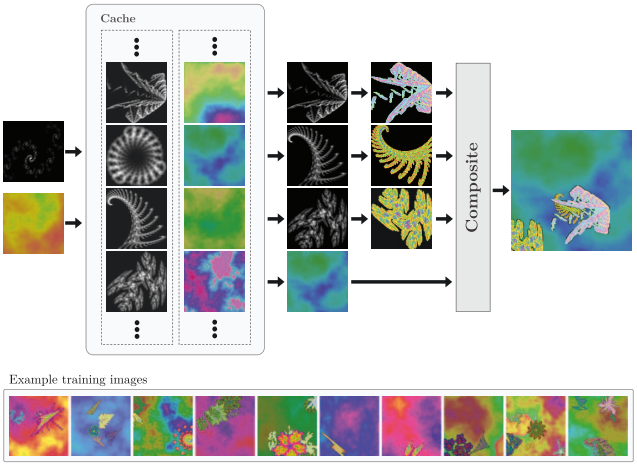

Figure 5. Rendering multi-instance images. A cache of fractals and backgrounds is regularly updated with new samples. Each training image is composed of a random selection of fractals from the cache, randomly augmented and composited on top of a random background image. Best viewed digitally, zoom in for details.

4.2.1 Single-instance Images

To generate a given fractal image, we start with the process described in Section 3.1. We then add three additional elements: patch-based rendering, as described in [15]; adding color to the fractal; and adding a randomly generated background using the "diamond-square" algorithm [11]. Full details are provided in Appendix B.1 in the Supplementary Material. In addition, we randomly scale and translate the region $\mathcal{R}$ (see Section 3.1), which scales and translates the resulting fractal. We also apply random flips and 90° rotations, and random Gaussian blur to the final image.

4.2.2 Multi-instance Images

We create multi-instance images by compositing one or more fractals and a background into a single image. For each image, we randomly sample the number of fractals, n, uniformly from $\{1, \ldots, N_{\max}\}$. Then we randomly sample n classes and generate their fractal images; we generate the fractals at a lower resolution (such as 128 instead of 224), for efficiency and because of how we composite them. We do not apply scaling or translation at this stage. We also generate a random background. In our experiments, we set $n_{\max} = 5$, to produce images which aren't overly cluttered.

Once we have the n fractals and a background, we composite them together. We randomly rescale each fractal and add it to the image at a random location. Fractals may end up partially occluded by other fractals, or partially outside the image, resulting in complex, varied, and challenging images for recognition (see Figure 5, bottom).

Rendering multiple fractals for every training image will almost certainly be too slow; in this case, a rendering cache becomes particularly useful. Every k_p training images (we set $k_p = 2$), we generate a new grayscale fractal image and new background, and update the cache. To generate a training image, we choose n random fractals and a background from the cache; we randomly flip and colorize each fractal, and add it to the background image as described previously. This allows us to create multi-instance images with roughly the same computation as in the single-instance case. This process is illustrated in Figure 5.

5. Experiments

Our basic fractal pre-training dataset consists of 50,000 IFS codes grouped into 1,000 classes. The IFS codes are sampled uniformly for $N \in \{2, 3, 4\}$, and the parameters are sampled as described in Section 3.2.2. We also employ a parameter augmentation method, which randomly selects one of the transforms $(A_k, \mathbf{b}_k)$ in the system and scales it by a factor $\gamma \sim U(0.8, 1.1)$ (while making sure not to overflow the singular values) to get $(\gamma A_k, \gamma \mathbf{b}_k)$. We found this to be more effective than the other augmentation methods we explored and plan to investigate why in future work.

Setup/Implementation

Our experiments use a ResNet50 [13] CNN model. We pre-train each model for 90 epochs, with 1,000,000 training samples per epoch, and with an image resolution of 224×224. Most models are pre-trained using 8 NVIDIA GTX 1080 Ti GPUs, with a total batch size of 512. Some of the ablations were run using 4 Tesla P100 GPUs and a batch size of 256.

We evaluate the effectiveness of the pre-trained representations by fine-tuning on several different tasks. For image classification, we use CUB-2011 [37], Stanford Cars [20], Stanford Dogs [16], FGVC Aircraft [26] and CIFAR-100 [21]. We also fine-tune models for medical image segmentation on the GlaS dataset [31]. For classification, we fine-tune for 150 epochs with a batch size of 96—we found that longer fine-tuning led to better performance. For segmentation, we fine-tune for 90 epochs with a batch size of 8.

We compare our proposed methods with three baselines: training from scratch (no pre-training); fine-tuning from ImageNet [30] pre-trained weights (available through PyTorch [29]); and, fine-tuning from FractalDB [15] pre-

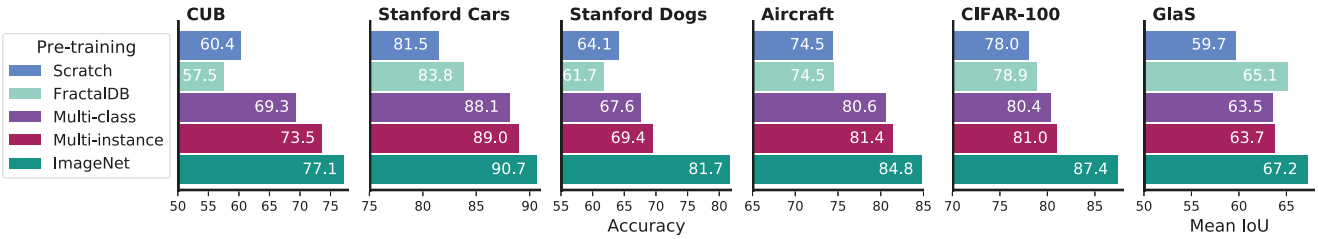

Figure 6. Fine-tuning evaluation results (classification accuracy) using different pre-training methods. The five datasets to the left are image classification datasets; the rightmost dataset, GlaS, is a medical image segmentation dataset.

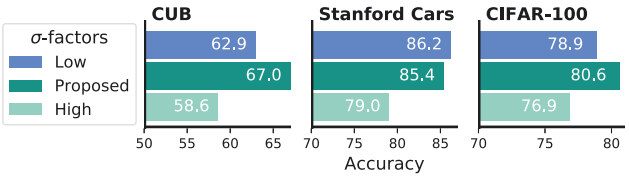

Figure 7. Performance from pre-training on systems with different σ-factor ranges: (**low**) $[1, \frac{1}{2}(5+N)]$; (**proposed**) $[\frac{1}{2}(5+N), \frac{1}{2}(6+N)]$; (**high**) $[3N-1, 3N]$.

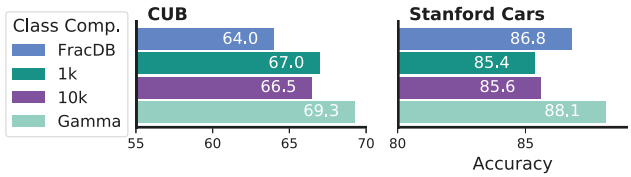

Figure 8. Performance from pre-trained models with different class composition: (**FracDB**) 1,000 classes and 1 system per class, with parameter augmentation used in [15]; (**1k**) 1,000 classes and 50 systems per class, without parameter augmentation; (**10k**) 10,000 classes and 50 systems per class without parameter augmentation; (**Gamma**) 1,000 classes and 50 systems per class, with γ-scaling augmentation (see beginning of Section 5).

trained weights, which we obtained using the dataset and code made available by the authors[5].

Our experiments use PyTorch [29] and the PyTorch-Lightning framework [8], with Hydra [38] for configuration. To facilitate reproducible research, all code and configuration files will be made publicly available.

5.1. Fine-tuning Results

Fig. 6 shows the fine-tuning performance on each evaluation task using different pre-training methods, along with training from scratch. Fractal pre-training provides a clear and consistent boost over both training from scratch as well as pre-training with FractalDB. Using multi-instance prediction as the pre-training task is also consistently better than using multi-class classification. In fact, multi-instance prediction models can provide more than 90% of the accuracy achieved by ImageNet pre-training—and in some cases, such as for Stanford Cars, the model obtains over 98% of the ImageNet performance.

5.2. Ablation Experiments

This section provides ablation experiment results isolating the impact of different parts of the proposed pre-training method. Additional results can be found in the Appendix.

Impact of σ-factors First, we consider the effects of using different ranges of σ-factors. Figure 7 shows the results of pre-trainng models using three different σ-factor ranges (see figure caption for details) and evaluating performance when fine-tuning for the CUB, Stanford Cars and CIFAR-100 datasets. Across all three datasets, the `high` σ-factors perform poorly relative to the `proposed` "good" geometry range. Similarly, the proposed σ-factor range outperforms the `low` σ-factors model on two of the three datasets: CUB and CIFAR-100.

Impact of Class Composition We also compare different choices for the number of classes/systems and parameter augmentation, with results presented in Figure 8. The γ-scaling augmentation appears to make the biggest difference. Interestingly, we don't seem to see any consistent improvement from using more classes and systems overall. This could merit further exploration, as it conflicts with the findings of [15].

6. Conclusion

We have shown that by carefully designing the processes for sampling and rendering affine IFSs, fractal pre-training can yield strong representations that are useful for real-world image recognition tasks. We have also proposed a pre-training task—multi-instance prediction—which greatly improves over multi-class classification as a pre-training task. Finally, we have shown that the fractal images used for pre-training can be generated "on-the-fly" in real-time during training, removing the need to generate, store, or transmit a large volume of data.

Acknowledgments This work was supported by the National Science Foundation under Grant No. IIS1651832. We gratefully acknowledge the support of NVIDIA for their donation of multiple GPUs used in this research.

[5] hirokatsukataoka16.github.io/Pretraining-without-Natural-Images

References

[1] M.F. Barnsley. Fractal modelling of real world images. In Heinz-Otto Peitgen and Dietmar Saupe, editors, *The Science of Fractal Images*, chapter 5, pages 219–242. Springer Verlang, 1988.

[2] Michael F. Barnsley and John H. Elton. A new class of markov processes for image encoding. *Advances in Applied Probability*, 20(1):14–32, 3 1988.

[3] Abeba Birhane and Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? In *WACV*, pages 1536–1546. IEEE, 6 2021.

[4] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and Daniel Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks, 2018.

[5] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large Scale Fine-Grained Categorization and Domain-Specific Transfer Learning. In *CVPR*, pages 4109–4118, 2018.

[6] Scott Draves and Erik Reckase. The fractal flame algorithm. *Citeseerx. Recuperado de http://citeseerx. ist. psu. edu/viewdoc/summary*, 2008.

[7] Nadav Dym, Barak Sober, and Ingrid Daubechies. Expression of fractals through neural network functions. *IEEE Journal on Selected Areas in Information Theory*, 1(1):57–66, 2020.

[8] WA Falcon *et al*. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

[9] Michael E Farmer. *Application of chaos and fractals to computer vision*. Bentham Science Publishers, 2015.

[10] Y. Fisher. *Fractal Image Compression: Theory and Application*. Springer New York, 2012.

[11] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, 6 1982.

[12] Kaiming He, Ross Girshick, and Piotr Dollar. Rethinking ImageNet Pre-Training. In *CVPR*, pages 4918–4927, 2019.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, pages 770–778, 2016.

[14] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. What makes ImageNet good for transfer learning? *ArXiv e-prints*, 8 2016.

[15] Hirokatsu Kataoka, Kazushige Okayasu, Asato Matsumoto, Eisuke Yamagata, Ryosuke Yamada, Nakamasa Inoue, Akio Nakamura, and Yutaka Satoh. Pre-training without Natural Images. In *ACCV*, 2020.

[16] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for Fine-Grained Image Categorization. In *CVPR Workshops (FGVC)*, 2011.

[17] Ljubiša M Kocić. Fractals and their applications in computer graphics. *Filomat*, pages 207–231, 1995.

[18] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do Better ImageNet Models Transfer Better? In *CVPR*, pages 2661–2671, 2019.

[19] Jonathan Krause, Benjamin Sapp, Andrew Howard, Howard Zhou, Alexander Toshev, Tom Duerig, James Philbin, and Li Fei-Fei. The Unreasonable Effectiveness of Noisy Data for Fine-Grained Recognition. In *ECCV*, volume 9907 LNCS, pages 301–320. Springer, Cham, 2016.

[20] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[21] Alex Krizhevsky, Geoffrey Hinton, and others. Learning multiple layers of features from tiny images. 2009.

[22] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT Compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15*, 2015.

[23] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. FractalNet: Ultra-Deep Neural Networks without Residuals. 5 2016.

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8693 LNCS, pages 740–755. Springer, Cham, 2014.

[25] Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the Limits of Weakly Supervised Pretraining. In *ECCV*, pages 181–196, 2018.

[26] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew B Blaschko, and Andrea Vedaldi. Fine-Grained Visual Classification of Aircraft. *arXiv.org*, 2013.

[27] André Luiz Marasca, Dalcimar Casanova, and Marcelo Teixeira. Assessing classification complexity of datasets using fractals. *International Journal of Computational Science and Engineering*, 20(1):102–119, 2019.

[28] Sergey I. Nikolenko. Synthetic data for deep learning, 2019.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[30] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision 2015 115:3*, 115(3):211–252, 4 2015.

[31] Korsuk Sirinukunwattana, Josien P.W. Pluim, Hao Chen, Xiaojuan Qi, Pheng Ann Heng, Yun Bo Guo, Li Yang Wang, Bogdan J. Matuszewski, Elia Bruni, Urko Sanchez, Anton Böhm, Olaf Ronneberger, Bassem Ben Cheikh, Daniel Racoceanu, Philipp Kainz, Michael Pfeiffer, Martin Urschler,

David R.J. Snead, and Nasir M. Rajpoot. Gland segmentation in colon histology images: The glas challenge contest. *Medical Image Analysis*, 35:489–502, 1 2017.

[32] Ryan Steed and Aylin Caliskan. Image Representations Learned With Unsupervised Pre-Training Contain Human-like Biases. *FAccT 2021 - Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, pages 701–713, 10 2020.

[33] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *ICCV*, pages 843–852, 2017.

[34] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training Deep Networks With Synthetic Data: Bridging the Reality Gap by Domain Randomization. In *CVPR*, pages 969–977, 2018.

[35] Aleksei Triastcyn and Boi Faltings. Generating Artificial Data for Private Deep Learning. *Proceedings of the PAL: Privacy-Enhancing Artificial Intelligence and Language Technologies, AAAI Spring Symposium Series*, 2019.

[36] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The INaturalist Species Classification and Detection Dataset. In *CVPR*, pages 8769–8778, 2018.

[37] C Wah, S Branson, P Welinder, P Perona, and S Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, California Institute of Technology, 2011.

[38] Omry Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019.

[39] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking Pre-training and Self-training. In *NeurIPS*, volume 33, pages 3833–3845, 2020.