Pre-training without Natural Images
–Supplementary Material–

Hirokatsu Kataoka[1], Kazushige Okayasu[1,2], Asato Matsumoto[1,3], Eisuke Yamagata[4], Ryosuke Yamada[1,2], Nakamasa Inoue[4], Akio Nakamura[2], and Yutaka Satoh[1,3]

[1] National Institute of Advanced Industrial Science and Technology (AIST)
[2] Tokyo Denki University
[3] University of Tsukuba
[4] Tokyo Institute of Technology

Abstract. The document is supplementary material of *Pre-training without Natural Images*.

1 All categories in FractalDB-1k

Figure 1 show all categories in FractalDB-1k.

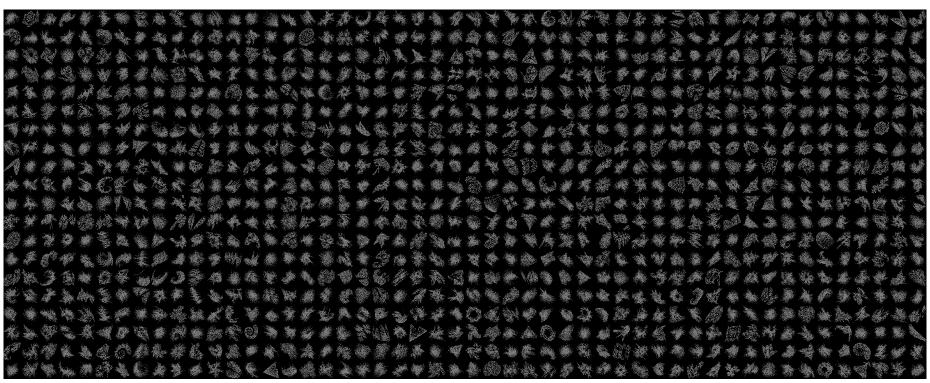


Fig. 1. All categories in FractalDB-1k dataset. In the figure, we list 1,000 fractal categories which are rendered by Iterated Function Systems (IFS). Surprisingly, a CNN architecture classify the kind of image patterns with close to 100% in a training time.

2 Detailed categories in ImageNet-100 (IN100) and Places-30 datasets (P30)

Table 1 and 2 indicate detailed categories used in IN100 and P30, respectively. The both datasets are used in the main paper.

Table 1. Categories in ImageNet-100.

n01560419	n01860187	n02058221	n02169497	n02655020	n02823750	n03146219	n03344393
n03483316	n03891332	n04141975	n04404412	n07860988	n01630670	n01910747	n02090622
n02356798	n02667093	n02835271	n03201208	n03372029	n03485794	n03938244	n04152593
n04442312	n09246464	n01632458	n01985128	n02093859	n02361337	n02727426	n02894605
n03207743	n03388549	n03495258	n03954731	n04254680	n04456115	n09468604	n01632777
n02002724	n02096437	n02422699	n02797295	n02930766	n03208938	n03394916	n03770439
n03956157	n04259630	n04525038	n15075141	n01694178	n02011460	n02099712	n02445715
n02802426	n02948072	n03240683	n03404251	n03770679	n03958227	n04346328	n06874185
n01728920	n02017213	n02106662	n02494079	n02804414	n03028079	n03255030	n03443371
n03777754	n03980874	n04355933	n07248320	n01776313	n02018795	n02112137	n02510455
n02804610	n03062245	n03272010	n03447447	n03874293	n04074963	n04366367	n07717410
n01833805	n02051845	n02113712	n02607072	n02808304	n03126707	n03325584	n03481172
n03876231	n04118538	n04399382	n07730033				

Table 2. Categories in Places-30.

amphitheater	cliff	food court	kennel outdoor	park
beauty salon	creek	grotto	manufactured home	promenade
boat deck	escalator indoor	harbor	medina	restaurant patio
bowling alley	flea market indoor	heliport	motel	roof garden
butchers shop	florist shop indoor	industrial area	mountain	server room
sky	skyscraper	tundra	valley	volcano

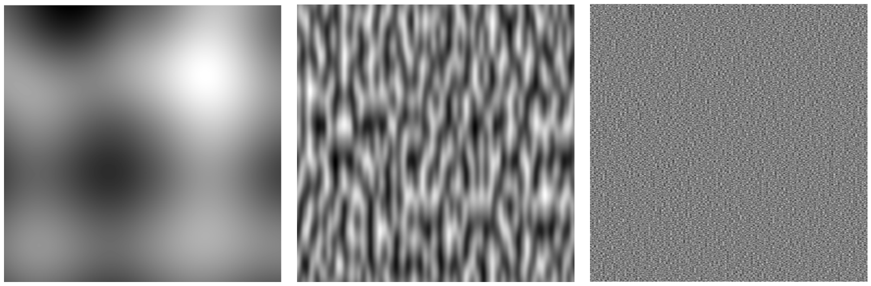
3 Formula-driven Image Projections

3.1 Perlin Noise

Perlin Noise is a widely used method for generating textures in computer graphics. Just like Fractals, it is formula driven and is capable of constructing a database without human annotation. It matches the concept of *Pre-training without Natural Images*, and thus we implemented a PerlinDB in comparison to the FractalDB.

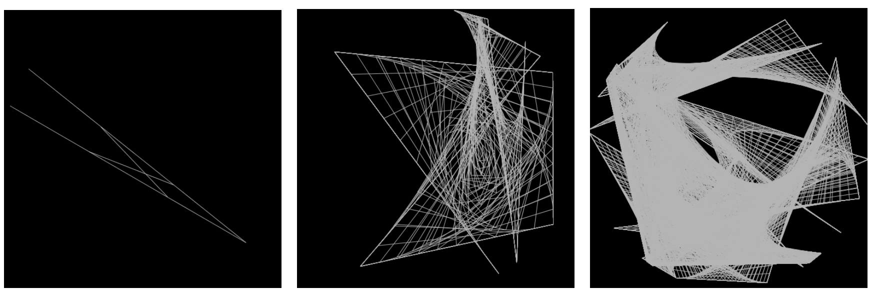
Generating Perlin Noise can be divided into three steps: definition of a 2D-grid, calculation based on an argument point, and interpolation. First, a 2D-grid is defined with a random gradient vector given at each grid point. Next, the value of each argument point is computed by a dot product between gradient vectors at the four corners of the cell the point belongs to, and distance vectors between the argument point and the corresponding grid points. Finally, through an interpolation between the four values computed at step 2, the final value of the argument point is determined. Through this simple process, Perlin Noise can be generated.

The interval gradient vectors are defined affects the complexity of the generated noise. For example, compared to a noise computed from a grid with a gradient vector at every grid point, a noise computed from a grid with a gradient vector at every two grid points will be rougher in terms of complexity



(a) Perlin 1 (b) Perlin 3 (c) Perlin 9

Fig. 2. Example images of Perlin noise



(a) Bezier 3 (b) Bezier 10 (c) Bezier 25

Fig. 3. Example images of Bezier curve

of the noise. In the implementation of the PerlinDB, we used this difference in complexity to generate categories. For example, in the Perlin_100 dataset, we defined a 1024*1024 grid with gradient vectors2^{10-n}grid points vertically and 2^{10-m} grid points horizontally $(n, m = 1, 2, ..., 10)$, which makes category n_m. As a result, we managed to create 100 categories: 01_01, 01_02, ... , 10_09, 10_10. 01_01 is the category with the roughest noises, whereas 10_10 is the one with the most detailed noises.

As for the instances within each of the categories, we changed the gradient vectors. The angles that the gradient vectors are defined at each grid point are determined randomly. Therefore, redefining the gradient vectors would result in different gradient vectors, and thus a different noise. Using this simple method, Perlin_100cat is made of 10,000 instances per category.

Comparing several datasets, Perlin_100, Perlin_324, Perlin_1296, it seemed that the more categories there are, the better the accuracy is. Also, between datasets with the same number of categories but with different number of instances, datasets with more instances performed better. This tendency is the same as that of FractalDB.

3.2 Bezier Curve

Just like PerlinDB, we also implemented a BezierCurveDB in comparison to the FractalDB. The BezierCurveDB consistes of images of Bezier Curve. It is a method for generating smooth curves in computer graphics. Bezier Curve is also formula driven and is capable of constructing a dataset without human annotation.

Bezier Curves is $n-1$-dimensional curves generated from n points. De Casteljau's algorithm is a widely used method for drawing the curves. Bezier Curves is generated by the following procedure:

1. Plot n dots
2. Form lines between thoes dots.
3. Plot dots dividing each of lines int to $t : 1-t$
4. Repeat (2) to (3) until only one dot is left

We implemented a BezierCurveDB for pre-training, and describe the dataset categories and instances. Note that generating images have a curve and lines formed to render the curves. The image categories is defined by a pair of n and s. The n is a number of dots plotting first in generating Bezier Curves. The s is a number of steps of dividing lines, in othe words, s is a number how many a line is divided epually. For example, in the Bezier_1024 dataset, we difined category n_s by combining 32 numbers (n,s = 3, 4, ..., 33, 34). As a result, we create 1024 categories: 03_03, 03_04, 03_05, ..., 34_32, 34_33, 34_34. 03_03 is the category of 2D curves generated with dividing lines into 3 equal parts. Next, as for the instances within each of the categories, we changed the place of first dots. By plotting dots randomly, BezierCurveDB is made of 1,000 instances per category.

We compared several datasets and BezierCurveDB (Bezier_144 and Bezier_1024) like PerlinDB. As a result, it found that BezierCurveDB has the same tendency as FractalDB and PerlinDB. The tendency is that the more the number of categories or instances, the higher the accuracy is.