FractalNet: Ultra-Deep Neural Networks without Residuals

Gustav Larsson
University of Chicago
larsson@cs.uchicago.edu

Michael Maire
TTI Chicago
mmaire@ttic.edu

Gregory Shakhnarovich
TTI Chicago
greg@ttic.edu

Abstract

We introduce a design strategy for neural network macro-architecture based on self-similarity. Repeated application of a single expansion rule generates an extremely deep network whose structural layout is precisely a truncated fractal. Such a network contains interacting subpaths of different lengths, but does not include any pass-through connections: every internal signal is transformed by a filter and nonlinearity before being seen by subsequent layers. This property stands in stark contrast to the current approach of explicitly structuring very deep networks so that training is a residual learning problem. Our experiments demonstrate that residual representation is not fundamental to the success of extremely deep convolutional neural networks. A fractal design achieves an error rate of 22.85% on CIFAR-100, matching the state-of-the-art held by residual networks.

Fractal networks exhibit intriguing properties beyond their high performance. They can be regarded as a computationally efficient implicit union of subnetworks of every depth. We explore consequences for training, touching upon connection with student-teacher behavior, and, most importantly, demonstrating the ability to extract high-performance fixed-depth subnetworks. To facilitate this latter task, we develop drop-path, a natural extension of dropout, to regularize co-adaptation of subpaths in fractal architectures. With such regularization, fractal networks exhibit an anytime property: shallow subnetworks provide a quick answer, while deeper subnetworks, with higher latency, provide a more accurate answer.

1 Introduction

ResNet [8] is a recent and dramatic increase in both depth and accuracy of convolutional neural networks, facilitated by constraining the network to learn residuals. ResNet variants [8, 9, 11] and related architectures [31] employ the common technique of initializing and anchoring, via a pass-through channel, a network to the identity function. Training now differs in two respects. First, the objective changes to learning residual outputs, rather than unreferenced absolute mappings. Second, these networks exhibit a type of deep supervision [18], as near-identity layers effectively reduce distance to the loss. He *et al.* [8] speculate that the former, residual formulation itself, is crucial.

We show otherwise, by constructing a competitive extremely deep architecture that does not rely on residuals. Our design principle is pure enough to communicate in a single word, fractal, and a simple diagram (Figure 1). Yet, fractal networks implicitly recapitulate many properties hard-wired into previous successful architectures. Deep supervision not only arises automatically, but also drives a type of student-teacher learning [1, 34] internal to the network. Modular building blocks of other designs [32, 20] are almost special cases of a fractal network's nested substructure.

For fractal networks, simplicity of training mirrors simplicity of design. A single loss, attached to the final layer, suffices to drive internal behavior mimicking deep supervision. Parameters are randomly initialized. As they contain subnetworks of many depths, fractal networks are robust to choice of overall depth; make them deep enough and training will carve out a useful assembly of subnetworks.
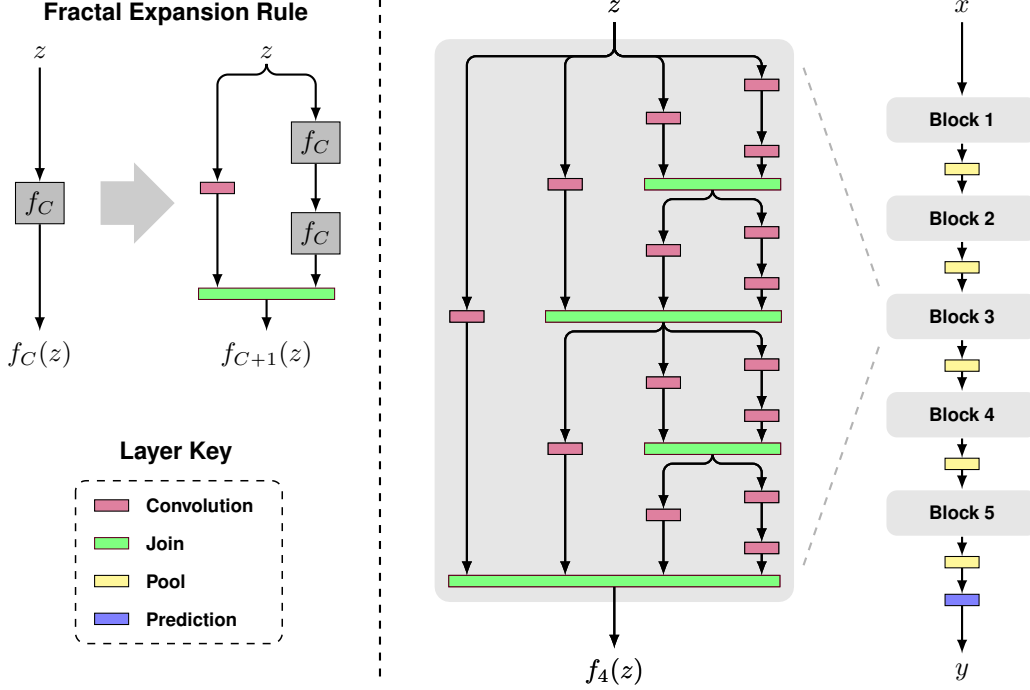
Figure 1: **Fractal architecture.** *Left:* A simple expansion rule generates a fractal architecture with C intertwined columns. The base case, $f_1(z)$, has a single layer of the chosen type (*e.g.* convolutional) between input and output. Join layers compute element-wise mean. *Right:* Deep convolutional networks periodically reduce spatial resolution via pooling. A fractal version uses f_C as a building block between pooling layers. Stacking B such blocks yields a network whose total depth, measured in terms of convolution layers, is $B \cdot 2^{C-1}$. This example has depth 40 ($B = 5, C = 4$).

The entirety of emergent behavior resulting from a fractal design may erode the need for recent engineering tricks intended to achieve similar effects. These tricks include residual functional forms with identity initialization, manual deep supervision, hand-crafted architectural modules, and student-teacher training regimes. Section 2 reviews this large body of related techniques. Hybrid designs could certainly integrate any of them with a fractal architecture; we leave open the question of the degree to which such hybrids are redundant or synergistic.

Our main contribution is twofold:

- We introduce FractalNet, the first alternative to ResNet in the realm of extremely deep convolutional neural networks. FractalNet is scientifically surprising; it shows that residual learning is not required for ultra-deep networks.

- Through analysis and experiments, we elucidate connections between FractalNet and an array of phenomena engineered into previous deep network designs.

As an additional contribution, we develop drop-path, a novel regularization protocol for ultra-deep fractal networks. Existing work on deep residual networks actually lacks demonstration of an effective regularization technique, instead relying solely on data augmentation [8, 11]. In the absence of data augmentation, fractal networks, trained with dropout [10] and drop-path together, far exceed the reported performance of residual networks.

Drop-path constitutes not only an intuitive regularization strategy, but also provides means of optionally guaranteeing that trained fractal networks exhibit anytime behavior. Specifically, a particular schedule of dropped paths during learning prevents subnetworks of different depths from co-adapting. As a consequence, both shallow and deep subnetworks must individually produce correct output. Querying a shallow subnetwork at test time thus yields a quick and moderately accurate result in advance of completion of the full network.

Section 3 elaborates the technical details of fractal networks and drop-path. Section 4 provides experimental comparisons to residual networks across the CIFAR-10, CIFAR-100 [14], and SVHN [25] datasets. We also evaluate regularization and data augmentation strategies, investigate subnetwork student-teacher behavior during training, and benchmark anytime networks obtained using drop-path. Section 5 provides synthesis. By virtue of encapsulating many known, yet seemingly distinct, design principles, self-similar structure may materialize as a fundamental component of neural architectures.

2 Related Work

Deepening feed-forward neural networks has generally returned dividends in performance. For example, see the improvement on the ImageNet [3] classification task when transitioning from AlexNet [15] to VGG-16 [28] to ResNet [8]. Unfortunately, greater depth also makes training more challenging, at least when employing a first-order optimization algorithm with a randomly initialized stack of layers. As the network grows deeper and more non-linear, the linear approximation of a gradient step becomes increasingly inappropriate. Desire to overcome these difficulties has driven research on both optimization techniques and network architectures.

On the optimization side, much work has yielded improvements. To prevent vanishing gradients, ReLU activation functions have widely replaced sigmoid and tanh units [24]. This subject remains an area of active inquiry, with various tweaks on ReLUs (*e.g.* PReLUs [7], ELUs [2]). Even with ReLUs, employing batch normalization [12] speeds training by reducing internal covariate shift. Good initialization can also partially ameliorate this problem [4, 23]. Path-SGD [26] offers an alternative normalization scheme. Progress in optimization is somewhat orthogonal to our architectural focus, with the expectation that advances in either are ripe for combination.

Notable ideas in architecture reach back to skip connections, the earliest example of a nontrivial routing pattern within a neural network. Recent work further elaborates upon them [22, 6]. Highway networks [31] and ResNet [8, 9] offer additional twists in the form of parameterized pass-through and gating. To date, they share distinction as the only designs scalable to hundreds of layers. ResNet's building block uses the identity map as an anchor point and explicitly parameterizes an additive correction term (the residual). Identity initialization also appears in the context of recurrent networks [16]. A tendency of ResNet and highway networks to fall-back to the identity map may make their effective depth much smaller than their nominal depth.

Some prior results hint at what we experimentally demonstrate in Section 4. Namely, reduction of effective depth is key to training extremely deep networks; residuals are incidental. Huang *et al.* [11] provide one clue by showing that randomly dropping layers from ResNet during training, thereby shrinking depth, improves performance. The success of deep supervision [18] provides another. Here, an auxiliary loss, forked off from mid-level layers, introduces a shorter path during backpropagation. The layer at the fork receives two gradients, originating from the main loss and the auxiliary loss, that are added together. Deep supervision is now common, being adopted, for example, by GoogLeNet [32]. However, irrelevance of the auxiliary loss at test time introduces the drawback of having a discrepancy between the actual objective and that used for training.

Exploration of the student-teacher paradigm [1] illuminates the potential for interplay between networks of different depth. In the model compression scenario, a deeper network (previously trained) guides and improves the learning of a shallower and faster student network [1, 34]. This is accomplished by feeding unlabeled data through the teacher and having the student mimic the teacher's soft output predictions. FitNets [27] explicitly couple students and teachers, forcing mimic behavior across several intermediate points in the network. Our fractal networks capture yet another alternative, in the form of implicit coupling, with the potential for bidirectional information flow between shallow and deep subnetworks.

Widening networks, by using larger modules in place of individual layers, has also produced performance gains. For example, an Inception module [32] concatenates results of convolutional layers of different receptive field size. Stacking these modules forms the GoogLeNet architecture. Liao and Carneiro employ a variant with maxout in place of concatenation [20]. Figure 1 makes apparent our connection with this work. As a fractal network deepens, it also widens. Moreover, note that stacking two 2D convolutional layers with the same receptive field (say 3×3) achieves a larger (5×5) receptive field. Instantiating our fractal network in this manner makes a horizontal cross-section somewhat reminiscent of an Inception module, except with additional joins due to recursive self-similarity.

3 Fractal Networks

We begin with a more formal presentation of the ideas sketched in Figure 1. Convolutional neural networks serve as our running example and, in the subsequent section, our experimental platform. However, it is worth emphasizing that our framework is more general. In principle, convolutional layers in Figure 1 could be replaced by a different layer type, or even a custom-designed module or subnetwork, in order to generate other fractal architectures.

Let C denote the index of the truncated fractal $f_C(\cdot)$. Our network's structure, connections and layer types, is defined by $f_C(\cdot)$. A network consisting of a single convolutional layer is the base case:

$$f_1(z) = \text{conv}(z) \tag{1}$$

We define successive fractals recursively:

$$f_{C+1}(z) = [(f_C \circ f_C)(z)] \oplus [\text{conv}(z)] \tag{2}$$

where $\circ$ denotes composition and $\oplus$ a join operation. When drawn in the style of Figure 1, C corresponds to the number of columns, or width, of network $f_C(\cdot)$. Depth, defined to be the number of conv layers on the longest path between input and output, scales as 2^{C-1}. Convolutional networks for classification typically intersperse pooling layers. We achieve the same by using $f_C(\cdot)$ as a building block and stacking it with subsequent pooling layers B times, yielding total depth $B \cdot 2^{C-1}$.

The join operation $\oplus$ merges two feature blobs into one. Here, a blob is the result of a conv layer: a tensor holding activations for a fixed number of channels over a spatial domain. The channel count corresponds to the size of the filter set in the preceding conv layer. As the fractal is expanded, we collapse neighboring joins into a single join layer which spans multiple columns, as shown on the right side of Figure 1. The join layer merges all of its input feature blobs into a single output blob.

Several choices seem reasonable for the action of a join layer, including concatenation and addition. We instantiate each join to compute the element-wise mean of its inputs. This is appropriate for convolutional networks in which channel count is set the same for all conv layers within a fractal block. Averaging might appear similar to ResNet's addition operation, but there are critical differences:

- ResNet makes clear distinction between pass-through and residual signals. In FractalNet, no signal is privileged. Every input to a join layer is the output of an immediately preceding conv layer. The network structure alone cannot identify any as being primary.
- Drop-path regularization, as described next in Section 3.1, forces each input to a join to be individually reliable. This reduces the reward for even implicitly learning to allocate part of one signal to act as a residual for another.
- Experiments show that we can extract high-performance subnetworks consisting of a single column (Section 4.2). Such a subnetwork is effectively devoid of joins, as only a single path is active throughout. They produce no signal to which a residual could be added.

Together, these properties ensure that join layers are not an alternative method of residual learning.

3.1 Regularization via Drop-path

Dropout [10] and drop-connect [35] modify interactions between sequential network layers in order to discourage co-adaptation. Since fractal networks contain additional macro-scale structure, we propose to complement these techniques with an analogous coarse-scale regularization scheme.

Figure 2 illustrates Drop-path. Just as dropout prevents co-adaptation of activations, drop-path prevents co-adaptation of parallel paths by randomly dropping operands of the join layers. This discourages the network from using one input path as an anchor and another as a corrective term (a configuration that, if not prevented, is prone to overfitting). We consider two sampling strategies:

- **Local**: a join drops each input with fixed probability, but we make sure at least one survives.
- **Global**: a single path is selected for the entire network. We restrict this path to be a single column, thereby promoting individual columns as independently strong predictors.

As with dropout, signals may need appropriate rescaling. With element-wise means, this is trivial; the join computes the mean of only the active inputs.
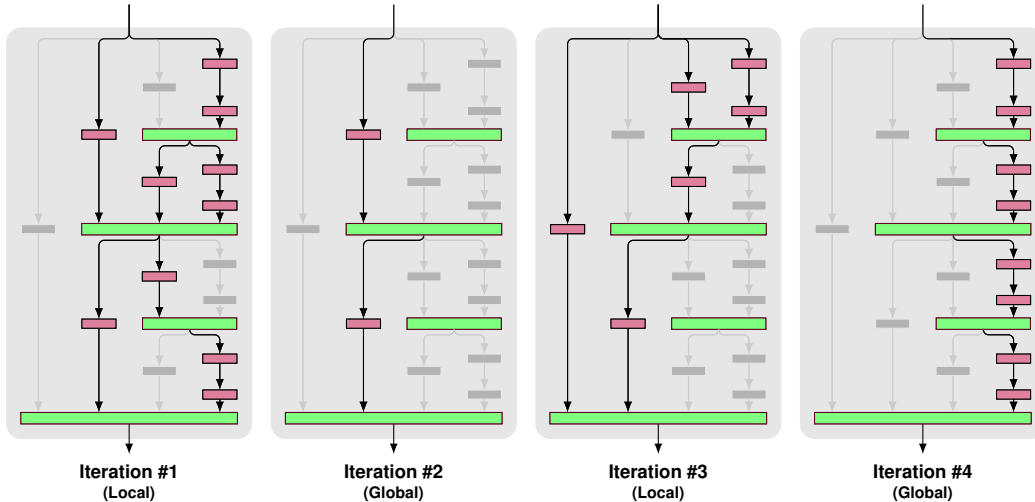
Figure 2: **Drop-path.** A fractal network block functions with some connections between layers disabled, provided some path from input to output is still available. Drop-path guarantees at least one such path, while sampling a subnetwork with many other paths disabled. During training, presenting a different active subnetwork to each mini-batch prevents co-adaptation of parallel paths. A global sampling strategy returns a single column as a subnetwork. Alternating it with local sampling encourages the development of individual columns as performant stand-alone subnetworks.

In experiments, we train with dropout and a mixture model of 50% local and 50% global sampling for drop-path. We sample a new subnetwork each mini-batch. With sufficient memory, we can opt to simultaneously evaluate one local sample and all global samples for each mini-batch by keeping separate networks and tying them together via weight sharing.

Global drop-path serves not only as a regularizer, but also as a diagnostic tool. Monitoring performance of individual columns provides insight into both the network and training mechanisms, as further discussed in Section 4.3). Individually strong columns of various depths also give users choices in the trade-off between fast (shallow) and more accurate (deep).

3.2 Data Augmentation

Data augmentation can greatly reduce the need for regularization. ResNet demonstrates this, achieving 27.22% error rate on CIFAR-100 with augmentation compared to 44.76% without [11]. While data augmentation benefits fractal networks, we make it a point to show that drop-path provides highly effective regularization, allowing them to achieve competitive results even without data augmentation.

3.3 Optimization

We train fractal networks using stochastic gradient descent with momentum. As now standard, we employ batch normalization together with each conv layer (convolution, batch norm, then ReLU).

4 Experiments

We implement FractalNet using Caffe [13]. Purely for convenience during implementation, we flip the order of pool and join layers at the end of a block in Figure 1. We pool individual columns immediately before the joins spanning all columns, rather than pooling once immediately after them.

CIFAR-100, CIFAR-10 [14], and SVHN [25] serve as testbeds for comparison to prior work and analysis of FractalNet's internal behavior. We evaluate performance on the standard classification task associated with each dataset. As these datasets all consist of 32×32 images, we fix our fractal network to have 5 blocks ($B = 5$) with 2×2 non-overlapping max-pooling and subsampling applied

Method	C100	C100+	C100++	C10	C10+	C10++	SVHN
Network in Network [21]	35.68	-	-	10.41	8.81	-	2.35
Generalized Pooling [17]	32.37	-	-	7.62	6.05	-	1.69
Recurrent CNN [19]	31.75	-	-	8.69	7.09	-	1.77
Competitive Multi-scale [20]	27.56	-	-	6.87	-	-	1.76
FitNet [27]	-	35.04	-	-	8.39	-	2.42
Deeply Supervised [18]	-	34.57	-	9.69	7.97	-	1.92
All-CNN [30]	-	33.71	-	9.08	7.25	4.41	-
Highway Network [31]	-	32.39	-	-	7.72	-	-
ELU [2]	-	24.28	-	-	6.55	-	-
Scalable BO [29]	-	-	27.04	-	-	6.37	1.77
Fractional Max-Pooling [5]	-	-	26.32	-	-	3.47	-
FitResNet (LSUV) [23]	-	27.66	-	-	5.84	-	-
ResNet [8]	-	-	-	-	6.61	-	-
ResNet (reported by [11])	44.76	27.22	-	13.63	6.41	-	2.01
ResNet: Stochastic Depth [11]	37.80	24.58	-	11.66	5.23	-	1.75
ResNet: Identity Mapping [9]	-	22.68	-	-	4.69	-	-
ResNet in ResNet [33]	-	22.90	-	-	5.01	-	-
FractalNet	35.34	23.30	22.85	10.18	5.22	5.11	2.01
FractalNet+dropout/drop-path	28.20	23.73	23.36	7.33	4.60	4.59	1.87
↳ Deepest column alone	29.05	24.32	23.60	7.27	4.68	4.63	1.89

Table 1: **CIFAR-100/CIFAR-10/SVHN.** We compare test error (%) with other leading methods, trained with either no data augmentation, translation/mirroring (+), or more substantial augmentation (++). Our main point of comparison is ResNet. We closely match its state-of-the-art results using data augmentation, and outperform it by large margins without data augmentation. Training with drop-path, we can extract from FractalNet simple single-column networks that are highly competitive.

after each. This reduces the input's 32×32 spatial resolution to 1×1 over the course of the entire network. A softmax prediction layer attaches at the end of the network.

Unless otherwise noted, we set the number of filter channels within blocks 1 through 5 as $(64, 128, 256, 512, 512)$. This progression is similar to that of VGG-16 [28] and ResNet [8]. It mostly matches the convention of doubling the number of channels after halving spatial resolution.

4.1 Training

For experiments using dropout, we fix drop rate per block at $(0\%, 10\%, 20\%, 30\%, 40\%)$ (similar to [2]). Local drop-path uses 15% drop rate across the entire network.

We train for 400 epochs on CIFAR-10/CIFAR-100 and 20 epochs on SVHN and drop the learning rate by a factor 10 every time we have halved the number of remaining epochs (*e.g.* at 200, 300, 350, 375). Our learning rate starts at 0.02 and we train using stochastic gradient decent with batch size 100 and momentum 0.9. We use Xavier initialization [4] for parameters.

A standard scheme for data augmentation on CIFAR-10 and CIFAR-100 is widely employed [21, 2, 31, 8, 9, 11, 33]. It consists only of translation and mirroring. Translation offsets are uniformly sampled from $\{-4, \ldots, 4\}$ for both image axes separately. The image is zero-padded where needed, after mean subtraction. Half of the images are flipped horizontally. We denote results achieved using no more than this degree of training data augmentation by appending a "+" to the dataset name (*e.g.* CIFAR-100+). A "++" marks results reliant on more data augmentation; here exact schemes may vary. Our entry in this category is modest and simply changes the zero-padding to reflect-padding.

4.2 Results

Table 1 reports performance of FractalNet, with $C = 3$ columns, on CIFAR and SVHN classification tasks. Scores of all competing methods are shown to provide context. FractalNet outperforms the original ResNet across the board. With data augmentation, FractalNet's performance on CIFAR-100 is

Cols.	Depth	Params.	Error (%)
1	5	0.3M	37.32
2	10	0.8M	30.71
3	20	2.1M	27.69
4	40	4.8M	27.38
5	80	10.2M	26.46
6	160	21.1M	27.38

Table 2: **Fractal Expansion.** (CIFAR-100++) Adding more levels greatly improves results. The benefit of depth in fractal networks still has a point of diminishing performance returns. However, contrast this with plain networks, where increasing depth eventually leads to the inability to train (Table 3).

Model	Depth	Train Loss	Error (%)
Plain	5	0.786	36.62
Plain	10	0.159	32.47
Plain	20	0.037	31.31
Plain	**40**	**0.580**	**38.84**
Fractal Col #1	5	0.677	37.23
Fractal Col #2	10	0.141	32.85
Fractal Col #3	20	0.029	31.31
Fractal Col #4	**40**	**0.016**	**31.75**
Fractal Full	**40**	**0.015**	**27.40**

Table 3: **Fractal Columns.** (CIFAR-100++) Training plain deep networks can lead to worse convergence as the depth increases. However, as a column trained within, and then extracted from, a fractal network with mixed drop-path, we overcome such depth limitation (possibly due to a student-teacher effect).

nearly identical to that of the best ResNet variant. With neither data augmentation nor regularization, our performance on CIFAR-100 and CIFAR-10 is superior to that of both ResNet and ResNet with stochastic depth. This suggests that FractalNet may be less prone to overfitting than ResNet. All methods perform similarly well on SVHN, making that dataset a poor differentiator.

Experiments without data augmentation highlight the power of drop-path regularization. On CIFAR-100, adding drop-path reduces FractalNet's error rate from 35.34% to 28.20%. Unregularized ResNet is far behind (44.76%) and ResNet regularized by stochastic depth (37.80%) does not even catch up to our unregularized starting point of 35.34%. CIFAR-10 mirrors this story.

In combination with data augmentation, drop-path either provides a boost (CIFAR-10), or does not significantly influence FractalNet's performance (CIFAR-100). With drop-path, FractalNet has the best error rate on CIFAR-10 (4.60%) of any model limited to standard data augmentation.

Note that for CIFAR-10, the performance of the deepest column of the fractal network is statistically indistinguishable from the full network. This suggests that, for this task, the fractal network may be more important as a learning framework than as a final model architecture.

Table 2 demonstrates that FractalNet is resistant to performance degradation as we increase C to obtain extremely deep networks (160 layers for $C = 6$). Scores in this table are not comparable to those in Table 1. For time and memory efficiency, we reduced block-wise feature channels to $(16, 32, 64, 128, 128)$ and the batch size to 50 for the supporting experiments in Tables 2 and 3.

Table 3 provides a baseline showing that training of plain deep networks begins to degrade by the time their depth reaches 40 layers. In our experience, a plain 160-layer completely fails to converge. Table 3 also highlights the ability to use FractalNet and drop-path as an engine for extracting trained networks (columns) with the same topology as plain networks, but much higher test performance.

4.3 Introspection

With Figure 3, we look into the evolution of a 40-layer FractalNet during training. Tracking individual columns (recording their losses when run as stand-alone networks), we observe that the 40-layer column initially improves slowly, but picks up once the loss of the rest of the network begins to stabilize. Contrast with a plain 40-layer network trained alone (dashed blue line), which never makes fast progress. The column has the same initial plateau, but subsequently improves after 25 epochs, producing a loss curve uncharacteristic of plain networks.

We hypothesize that the fractal structure triggers effects akin to deep supervision and lateral student-teacher information flow. Column #4 joins with column #3 every other layer, and in every fourth layer this join involves no other columns. Once the fractal network partially relies on the signal going through column #3, drop-path puts pressure on column #4 to produce a replacement signal
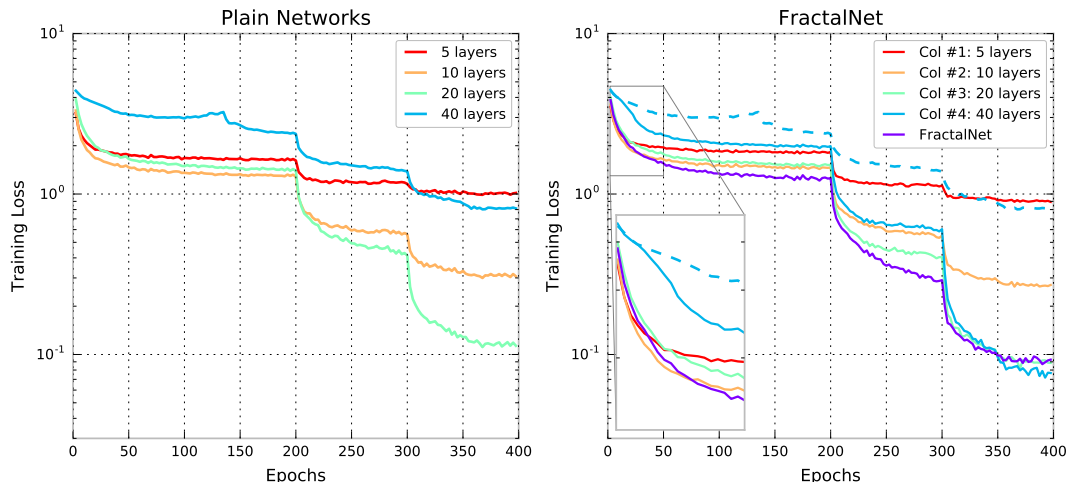
Figure 3: **Implicit deep supervision.** *Left:* Evolution of loss for plain networks of depth 5, 10, 20 and 40 trained on CIFAR-100. Training becomes increasingly difficult for deeper networks. At 40 layers, we are unable to train the network satisfactorily. *Right:* We train a 4 column fractal network with mixed drop-path, monitoring its loss as well as the losses of its four subnetworks corresponding to individual columns of the same depth as the plain networks. As the 20-layer subnetwork starts to stabilize, drop-path puts pressure on the 40-layer column to adapt, with the rest of the network as its teacher. This explains the elbow-shaped learning curve for Col #4 that occurs around 25 epochs.

when column #3 is dropped. This task has constrained scope. A particular drop only requires two consecutive layers in column #4 to substitute for one in column #3, a mini student-teacher problem.

5 Conclusion

FractalNet demonstrates that path length is fundamental for training ultra-deep neural networks; residuals are incidental. Key is the shared characteristic of FractalNet and ResNet: large nominal network depth, but effectively shorter paths for gradient propagation during training. Fractal architectures are arguably the simplest means of satisfying this requirement, and match or exceed ResNet's experimental performance. They are resistant to being too deep; extra depth may slow training, but does not impair accuracy.

With drop-path, regularization of extremely deep fractal networks is intuitive and effective. Drop-path doubles as a method of enforcing latency/accuracy tradeoffs within fractal networks, for applications where fast answers have utility.

Our analysis connects the emergent internal behavior of fractal networks with phenomena built into other designs. Their substructure is similar to hand-designed modules used as building blocks in some convolutional networks. Their training evolution may emulate deep supervision and student-teacher learning.

References

[1] J. Ba and R. Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[2] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *ICLR*, 2016.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[4] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[5] B. Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[6] B. Hariharan, P. Arbelaez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. *CVPR*, 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

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

[9] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[11] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *arXiv preprint arXiv:1408.5093*, 2014.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[15] A. Krizhevsky, S.Ilya, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[16] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[17] C.-Y. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *AISTATS*, pages 464–472, 2016.

[18] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeply-supervised nets. *NIPS Workshop on Deep Learning and Representation Learning*, 2014.

[19] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *CVPR*, pages 3367–3375, 2015.

[20] Z. Liao and G. Carneiro. Competitive multi-scale convolution. *arXiv preprint arXiv:1511.05635*, 2015.

[21] M. Lin, Q. Chen, and S. Yan. Network in network. *ICLR*, 2013.

[22] M. Maire, S. X. Yu, and P. Perona. Reconstructive sparse code transfer for contour detection and semantic labeling. *ACCV*, 2014.

[23] D. Mishkin and J. Matas. All you need is a good init. *ICLR*, 2016.

[24] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

[25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[26] B. Neyshabur, R. Salakhutdinov, and N. Srebro. Path-sgd: Path-normalized optimization in deep neural networks. *NIPS*, 2015.

[27] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *ICLR*, 2015.

[28] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[29] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Ali, R. P. Adams, et al. Scalable bayesian optimization using deep neural networks. *ICML*, 2015.

[30] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *ICLR (workshop track)*, 2014.

[31] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *ICML*, 2015.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[33] S. Targ, D. Almeida, and K. Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[34] G. Urban, K. J. Geras, S. Ebrahimi Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson. Do deep convolutional nets really need to be deep (or even convolutional)? *ArXiv e-prints arXiv:1603.05691*, Mar. 2016.

[35] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, pages 1058–1066, 2013.