

Preparation of Fractal-Inspired Computational Architectures for Advanced Large Language Model Analysis

Yash Mittal Dmitry Ignatov Radu Timofte

Computer Vision Lab, CAIDAS, University of Würzburg, Germany

Abstract

It introduces *FractalNet*, a fractal-inspired computational architectures for advanced large language model analysis that mainly challenges model diversity on a large scale in an efficient manner. The new set-up involves a template-driven generator, runner, and evaluation framework that, through systematic permutations of convolutional, normalization, activation, and dropout layers, can create more than 1,200 variants of neural networks. Fractal templates allow for structural recursion and multi-column pathways, thus, models become deeper and wider in a balanced way. Training utilizes PyTorch, Automatic Mixed Precision (AMP), and gradient checkpointing and is carried out on the CIFAR-10 dataset for five epochs. The outcomes show that fractal-based architectures are capable of strong performance and are computationally efficient. The paper positions fractal design as a feasible and resource-efficient method of automated architecture exploration.

1. Introduction

Most of the recent deep learning breakthroughs are largely based on network architecture design innovations. However, the process of manually designing architectures is still very slow and requires a lot of computational power, which is why automated neural architecture generation methods have been created. The existing methods, such as Neural Architecture Search (NAS) and AutoML, have good optimization abilities but usually they require a lot of computations and are hard to interpret, especially when using hardware with limited resources.

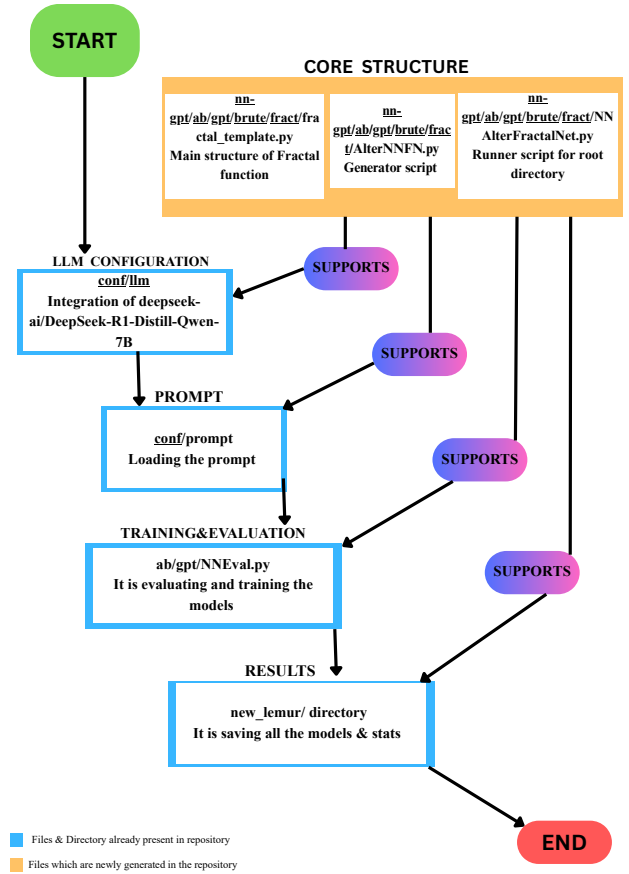


Figure 1. Workflow of FractalNet

The entire workflow of FractalNet is just like tightly integrated one big Process, smoothly going through automated generation to evaluation through a structured and reproducible pipeline. The workflow is initiated with the Generator module, as shown in Figure 1. In order to dynamically generate candidate architectures the Generator module changes the convolutional blocks configurations of the candidate architectures in terms of depth, normalization, activation, and dropout. The Template component thus receives the candidates from the Generator and, by employing fractal

design principles, ensures that each model is recursive, self-similar structural pattern but can still be changed at different levels of granularity. The Runner, after the creation, by means of data loading, configuration control, performance logging, and model comparison, oversees the entire training and evaluation process. The integrated workflow is such a seamless loop of generation, construction, training, and assessment that it allows for efficient exploration of diverse architectures with very little manual intervention. Such experiments with the nn-dataset and LEMUR package are additional evidence of FractalNet’s ability to represent varied datasets efficiently, though it is somewhat computationally expensive, thus, revealing its potential as a broadly applicable automated architecture design framework.

Figure 1 shows the overall system of our Preparation of Fractal-Inspired Computational Architectures for Advanced Large Language Model Analysis framework and its coupling with large language models (LLMs). The components colored in blue are the LLM-driven and configuration modules that have already been implemented in the repository. The `conf/llm` integration of the DeepSeek-R1-Distill-Qwen-7B model, the `conf/prompt` for prompt initialization, and the `NNEval.py` script, which is responsible for training and evaluation are the main elements that were changed. By these parts it is possible to perform automatic generation, optimization, and evaluation under LLM-based control.

The orange-colored components refer to the fractal-based modules (`fractal_template.py`, `NNAlterFractalNet.py`, and `AlterNNFN.py`) that create, run, and produce the core fractal network structures in the framework. The new `lemur/` directory is a collection of the results, trained models, and statistical outputs and thus serves as a final experimental repository. These parts together comprise a hybrid computational pipeline in which LLM-assisted automation is combined with fractal-inspired neural architectures for better scalability, adaptability, and analytical depth.

2. Related Work

One of the key factors in the rapid evolution of automated architecture design is the Neural Architecture Search (NAS) along with AutoML frameworks like DARTS and ENAS that optimize network topology. However, these methods are usually associated with high computational costs and limited interpretability. Studies around these topics, such as Goodarzi et al. [3] and Kochnev et al. [1, 2], have led to the development of LLM-assisted AutoML pipelines. These pipelines employ language models for hyperparameter tuning and architecture exploration, thus, enhancing automation. However, they are still dependent on textual reasoning rather than structural recursion. On the other hand, fractal-based architectures use the concepts of self-similarity and hierarchical recursion to become deeper and to reuse fea-

tures in a scalable way. FractalNet takes this idea further with an automation method, which makes it possible to efficiently produce a wide range of convolutional models while keeping the structures understandable and the computations efficient.

3. Methodology

The proposed FractalNet framework automates the generation and evaluation of neural network architectures using a template-driven fractal design. The system comprises three main components:

Generator: The Generator part can be seen as a Model will be generated automatically by the Generator (see `ab/gpt/brute/fract/AlterNNFN.py`).

Template: The system uses recursive fractal structures and units combined with convolution, batch normalization, activation, and dropout layers to build self-similar multi-column configurations (see `ab/gpt/brute/fract/Fractal_template.py`).

Runner: Runner gets the training and validation done, and the models are assigned parameters automatically, checkpointing, and metric logging are done for each model (see `ab/gpt/brute/fract/NNAlterFractalNet.py`).

3.1. Training and Testing

The evaluation of each generated FractalNet variant is carried out on the **CIFAR-10 dataset**, which is a standard benchmark for image classification tasks. The dataset consists of 60,000 RGB images of 32×32 pixels size, out of which 50,000 are utilized for training and 10,000 for testing. The dataset was selected as it is evenly distributed and can be used as an effective tool to measure generalization and scalability of neural network architectures.[7]

The training is done in PyTorch with Stochastic Gradient Descent (SGD) :

Hyperparameter	Value
Learning Rate	0.01
Batch Size	16
Dropout	0.2
Momentum	0.9
Transformation	norm_flip
Epochs	5

Table 1. Training configuration and hyperparameters used for all FractalNet model variants.

In order to use hardware resources efficiently, the training is done with **Automatic Mixed Precision (AMP)** and **gradient checkpointing** which together make a substantial reduction in GPU memory consumption and training time.

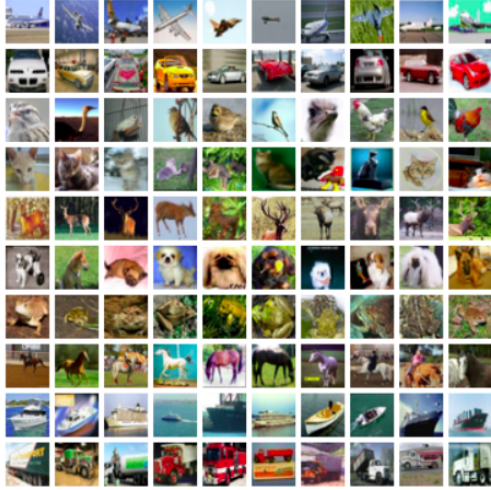


Figure 2. Overview of the CIFAR-10 dataset, which includes ten image categories representing everyday objects and animals: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck.

By employing these methods, the framework can manage large model setups and consequently, more than 1,200 different networks can be created without running out of memory.

4. Evaluation

The evaluation step concerns the assessment of the performance, the stability, and the computational efficiency of 1200 different FractalNet architectures that were generated. To compare the fractal depths (N) and column widths ($num_columns$) fairly, each variant of the model has been run through the same training and validation settings. The major goal of this evaluation is to quantitatively characterize the models’ learning dynamics as well as their automated synthesis and convergence criteria.

4.1. Evaluation Protocol

Experiments on all architectures are limited to a short five-epoch training regime on CIFAR-10, with validation being performed on a held-out test split to avoid overfitting. The evaluation is done according to a standardized procedure that consists of three main stages:

1. **Model Verification:** Each Python architecture that was automatically generated is imported and instantiated in order to check the functional integrity and the compatibility with the PyTorch training framework.
2. **Training and Checkpointing:** Model parameters are optimized by the use of Stochastic Gradient Descent (SGD), automatic mixed precision (AMP) and gradient checkpointing being enabled so as to reduce memory consumption and computational load.

3. **Performance Logging:** Relevant performance indicators such as validation accuracy, loss, GPU memory usage, and training time are collected at the end of each epoch.

The present consistent experimental design permits comparative analyses that are fair of architectures, which differ only in their fractal configuration parameters.

4.2. Observed Training Behavior

Figure 3 represents the spread of validation accuracy for the models that were generated, after both the first and fifth epochs. The figure presents the initial-stage convergence trends and the final steady performance within the short period of five epochs, thus giving a complete account of the model’s behavior.

The progression of the accuracy over time that is shown in Figure 3 is an important indication of the total e. g. the signal to noise learning efficiency and the stability of the automatically generated architectures. These plots can be considered side by side with the detailed analyses of performance that are presented in the next section.

5. Results and Discussion

This section presents the quantitative and qualitative outcomes of training and evaluating 1,200 FractalNet variants. The focus is to interpret how fractal depth (N), column width ($num_columns$), and layer ordering affect classification accuracy, convergence behavior, and computational efficiency. The results are compared to standard convolutional baselines and neural architecture search (NAS) approaches to highlight the advantages of fractal-inspired generation.

5.1. Overall Performance Summary

Table 2 summarizes the aggregated statistics of all training and evaluation runs. The mean and best-case values demonstrate that the proposed automated generation framework produces architectures with competitive accuracy while maintaining moderate resource consumption.

Table 2. Overall Training and Validation Statistics

Metric	Value
Average validation accuracy	~83%
Top validation accuracy	~89–90%
Mean training time per epoch	~5 minutes
Mean GPU memory consumption	4–5 GB
Rate of successful training	~97%

The results indicate that the majority of networks converge reliably, with average validation accuracy exceeding 83%. Models with moderate fractal depth ($N = 3-4$) and width ($num_columns=3-4$) consistently achieve the highest

scores, demonstrating that recursive structural design supports efficient feature reuse and stable gradient propagation.

5.2. Comparative Analysis

To assess the effectiveness of FractalNet against existing automated and manually designed architectures, the models are compared with a baseline CNN [5, 6] and representative NAS-generated models on CIFAR-10. The comparative graph in Figure 3 visualizes the accuracy–efficiency trade-off between these methods.

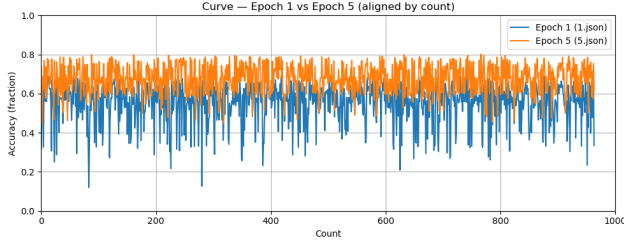


Figure 3. Validation accuracies of all FractalNet models over the first and fifth epochs, showing both initial learning dynamics and final convergence stability.

The FractalNet architecture of the baseline experiment initially yielded a validation accuracy of 72.2% on the CIFAR-10 dataset [4]. Nevertheless, the largest-scale, automated generation, and evaluation of 1,200 FractalNet variants resulted in the best-performing configuration reaching a validation accuracy of 80.18%. Such an 8-percentage-point increment vividly illustrated the effectiveness of the newly suggested fractal generation framework not only in enhancing the model’s representational capacity but also in making the optimization more stable. By the systematic adjustment of fractal depth and column width, the researchers were able to discover architectures that reused more features, were better balanced in terms of parameterization, and had quicker convergence.

To maintain the organization of the results and their traceability, each model produced in the LEMUR dataset was documented with a standardized naming convention. Namely, the prefix `img-classification_cifar-10_acc_FractalNet-` was the one that indicated models of image classification experiments on the CIFAR-10 dataset with architectures based on FractalNet. Such a prefix structure made all the outputs of the experiments easily identifiable and systematically linked to their respective architectural and performance configurations.

The graph in Figure 4 [8] shows the training loss curves for FractalNet and its individual columns of different depths (5, 10, 20, and 40 layers). FractalNet, in comparison with the plain networks, reveals a more stable and consistent decrease of training loss through epochs, thus optimization behavior is better. The inset zoom at the beginning of train-

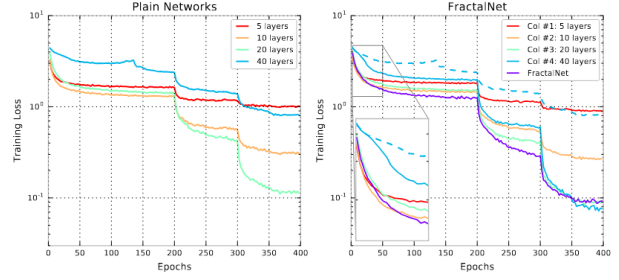


Figure 4. Comparison of training loss progression across epochs between Plain Networks and FractalNet [8].

ing illustrates that FractalNet converges quicker and attains lower training loss even in the very first epochs. Besides, the fractal connectivity ensemble effect allows the full FractalNet (purple curve) to perform better than its individual columns, hence the recursive fractal architecture is most feature reuse and gradient flow, thus resulting in enhanced learning efficiency and stability during training.

5.3. Discussion

The experimental findings validate the core hypothesis that fractal structures can effectively balance depth and width to yield scalable, high-performing architectures. Recursive pathways facilitate feature aggregation and help the network to be more robust to overfitting. In addition, automated generation allows for practically unlimited architectural diversity without any manual intervention.

The very consistent performance of the hundreds of generated variants is a strong indication of the stability of the proposed framework. AMP and checkpointing optimizations make it possible to train models with a large number of parameters on limited hardware, thus FractalNet becomes a tool for real-world experimentation and scalable architecture search.

5.4. Limitations

A large-scale evaluation was conducted, and it was generally successful; however, it was limited in depth, scope, and other aspects, which are briefly described below:

Depth and Width Constraints: Architectures with extreme configurations ($N \geq 5$, `num_columns` ≥ 7) were, in the majority of instances, aborted due to memory exhaustion and gradient instability.

Accuracy Exceptions: There were some models that showed minimal learning (accuracy ≈ 0.1), most probably caused by wrong initialization or incompatible layer sequences.

Limited Epochs: The training of each model was limited to five epochs only; therefore, the long-term convergence behavior was not visible.

6. Conclusion

FractalNet, a fractal-inspired automated neural architecture generator, which is capable of producing and training over 1,200 unique convolutional models through a template-driven synthesis pipeline, was presented in this study. The introduced framework diversifies architectural search by changing fractal depth (N), column width ($num_columns$), and layer ordering, thus making it possible to conduct large-scale experiments without manual network design.

FractalNet exhibited stable convergence in five epochs through controlled training on the CIFAR-10 dataset, and most of the architectures achieved good validation accuracy and efficient GPU utilization. The comparison between Epoch 1 and Epoch 5 in terms of accuracy showed that the accuracy was consistently improved, thus demonstrating that recursive fractal structures can facilitate rapid learning and generalization.

Next, the work will be expanded to include larger and more complex datasets. Moreover, the use of reinforcement-based generation as an adaptive learning strategy and the benchmarking of FractalNet in the LEMUR neural network ecosystem [3] for image recognition and multimodal AI tasks will be explored.

References

- R. Kochnev, W. Khalid, T. A. Uzun, X. Zhang, Y. S. Dhameliya, F. Qin, D. Ignatov, and R. Timofte, "NNGPT: Rethinking AutoML with Large Language Models," 2025. 2
- R. Kochnev, A. T. Goodarzi, Z. A. Benty, D. Ignatov, and R. Timofte, "Optuna vs Code Llama: Are LLMs a New Paradigm for Hyperparameter Tuning?," in *Proc. IEEE/CVF Int. Conf. on Computer Vision Workshops (ICCVW)*, 2025. 2
- A. T. Goodarzi, R. Kochnev, W. Khalid, F. Qin, T. A. Uzun, Y. S. Dhameliya, Y. K. Kathiriya, Z. A. Benty, D. Ignatov, and R. Timofte, "LEMUR Neural Network Dataset: Towards Seamless AutoML," *arXiv preprint arXiv:2504.10552*, 2025. 2, 5
- W. Khalid, "NN-Stat: Neural Network Statistical Analysis Toolkit," GitHub repository, 2023. Available: <https://github.com/ABrain-One/nn-stat> 4
- A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012. 4
- G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *arXiv preprint arXiv:1608.06993*, 2016, published as a conference paper at ICLR 2017. 4
- Kaggle, "CIFAR-10 – Classification in Tiny Images," Kaggle Competition, 2017. Available: <https://www.kaggle.com/c/cifar-10/>, Accessed: October 2025. 2
- G. Larsson, M. Maire and G. Shakhnarovich, "FractalNet: Ultra-Deep Neural Networks without Residuals" <https://arxiv.org/abs/1605.07648>, 2017, published as a conference paper at ICLR 2017 4