

Dynamic neural architecture search : A pathway to efficiently optimized deep learning models

Chetan Gode *

Department of Electronics & Telecommunication Engineering

Yeshwantrao Chavan College of Engineering

Nagpur 441110

Maharashtra

India

Bhushan Marutirao Nanche [†]

Department of Information Technology

D. Y. Patil College of Engineering

Akurdi

Pune 411044

Maharashtra

India

Dharmesh Dhabliya [§]

Department of Information Technology

Vishwakarma Institute of Technology

Pune 411037

Maharashtra

India

Rahul Dnyanoba Shelke [‡]

Department of Mechanical Engineering

Everest College of Engineering & Technology

Aurangabad 431008

Maharashtra

India

* E-mail: chetan_gode@rediffmail.com (Corresponding Author)

[†] E-mail: bmnanche@gmail.com

[§] E-mail: dharmesh.dhabliya@viit.ac.in

[‡] E-mail: rahuldselke@gmail.com

Rajendra V. Patil [@]

*Department of Computer Engineering
SSVPS Bapusaheb Shrivajirao Deore College of Engineering
Dhule 424005
Maharashtra
India*

Sushma Bhosle [#]

*Department of Electronics & Telecommunication Engineering
Nutan Maharashtra Institute of Engineering and Technology
Talegaon Dabhade
Pune 410507
Maharashtra
India*

Abstract

Deep learning models have changed many areas by making it possible for them to do things like picture recognition, natural language processing, and reinforcement learning at the highest level. But optimizing these models is still very hard because they are very complicated and need a lot of computing power. This article suggests a way to make deep learning models work better by looking at important parts like model architecture design and training methods. Using methods like neural architecture search and model distillation, we present a methodical way to create systems that work well. This makes it possible to make models that are small but strong, which is good for places with limited resources. We also talk about how important it is to use the right start and regularization methods to avoid overfitting and make generalization better. We look into new ways to train that are meant to speed up convergence and lower the cost of computing. This includes methods like mixed-precision training, progressive shrinking, and transfer learning that use parallelism in both data and hardware to speed up training without lowering performance. We also look into how domain-specific information can be used to guide the optimization process and make models better able to meet the needs of particular tasks. To see how well our suggested route works, we run a lot of tests on a lot of different datasets and standards. The outcomes show big changes in model efficiency, with performance being on par with or even better than standard methods while using a lot less computing power.

Subject Classification: 68Q25.

Keywords: Neural architecture search (NAS), Dynamic search strategy, Reinforcement learning, Neural network optimization, Evolutionary algorithms, Meta-learning.

[@] E-mail: patilrajendra.v@gmail.com

[#] E-mail: sushma.bhosle28@gmail.com

1. Introduction

Deep learning has changed many fields, like computer vision, natural language processing, and robots, by making it possible to do amazing things that were thought to be impossible before [1]. Designing neural network topologies is a key part of deep learning because they determine how much a model can learn, how well it works, and how well it can generalize. Traditional ways of designing buildings often rely on people's gut feelings and trial-and-error, which can take a long time, require a lot of work, and not always be the best solution [2]. Neural Architecture Search (NAS) has become an interesting way to automatically design neural network architectures. This saves researchers the time and effort of planning everything by hand and makes it easier to explore huge architectural spaces [3]. In the early days of NAS, techniques like reinforcement learning or evolutionary algorithms, which require a lot of computing power, were often used to find the best designs. To solve the problems of NAS's scale and efficiency, recent progress has moved toward search methods that are more dynamic and adaptable [4]. The goal of Dynamic Neural Architecture Search (DNAS) is to lower search costs while keeping or even improving the quality of the designs that are found [5]. One main idea behind DNAS comes from the fact that not all parts of a neural network's architecture have the same effect on how well it works in the end [6]. This adaptive search approach is based on the idea that organisms can change to changing surroundings by putting resources where they are most needed for life [7].

In addition to making things more efficient, DNAS also lets people try out more unique and different building ideas. When using traditional NAS methods, they tend to settle on a small group of designs that have been shown to work on certain standards. However, DNAS algorithms support exploring a wider range of design options, which could lead to speed and efficiency improvements [12].

2. Proposed Method

2.1 *Dynamic Architecture Encoding:*

Figure 1 shows the architecture That Changes Encoding means putting each possible design into a short, flexible style that makes it easy to look through during the search process. In an organized way, this encoding store important design features like layout, connection patterns, and hyperparameters. It's also possible to store hyperparameters like

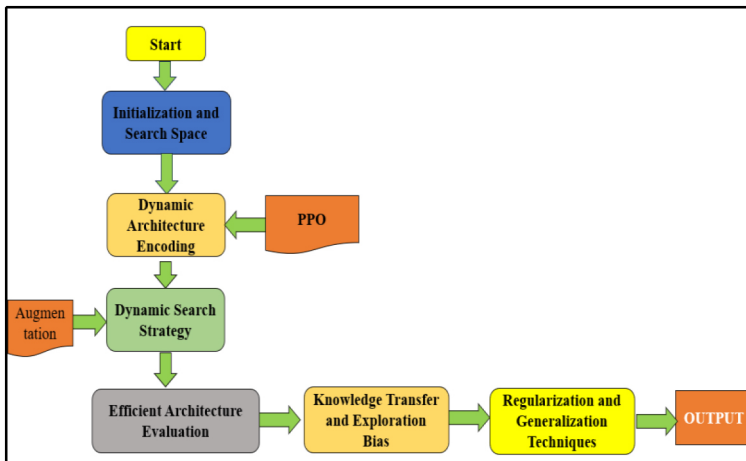


Figure 1
Block Diagram of Proposed Model

kernel sizes, filter counts, and activation functions as node properties or adjacency matrices [11].

2.2 Dynamic Search Strategy:

Neural Architecture Search (NAS) uses a Dynamic Search Strategy that changes the search area all the time based on how well sampled designs work. This makes the best use of computing resources. Reinforcement learning (RL) is one way to do dynamic search. In RL, an agent learns how to move through the search area by interacting with it and getting feedback in the form of rewards [10] [14]. The RL agent changes its research approach over and over again based on these benefits. This way, it focuses on places that are most likely to produce high-performing systems, proposed model shown in figure 1. Dynamic search strategies use tools to change the search space on the fly, either by making it bigger or smaller, or by rearrangement, based on how well sampled designs work [8]. The method makes the best use of computer resources by changing the search space on the fly. This speeds up the process of finding the best or almost best designs for the given job or limits.

PPO Algorithm is as follows:

Step 1: Initialization:

- Initialize the policy network parameters θ .

Step 2: Interaction with the Search Space:

- At each step t , select action a_t according to policy $\pi_\theta(a|s_t)$

$$a_t \sim \pi_\theta(a|s_t) \quad (1)$$

Step 3: Reward Calculation:

- Compute reward r_t based on the performance improvement or degradation.

$$r_t = \text{Reward}(s_t, a_t) \quad (2)$$

Step 4: Policy Update:

- Update policy parameters θ to maximize expected cumulative reward using the PPO surrogate objective:

$$L^{(PPO)}(\theta) = E \left[t \min \left(r_t A_t, \text{clip} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right] \quad (3)$$

Step 5: Dynamic Search Space Adjustment:

- Dynamically adjust the search space based on observed rewards and performance trends.

$$\text{AdjustSearchSpace}(s_t, r_t) \quad (4)$$

Step 6: Exploration-Exploitation Tradeoff:

- Balance exploration and exploitation by updating the policy while constraining policy changes using the PPO objective.

2.3 Regularization and Generalization Techniques:

Regularization and generalization methods are very important for making deep learning models, like those found by Neural Architecture Search (NAS) [9], work better and be more reliable. The goal of these methods is to stop overfitting, which happens when the model learns to remember training data instead of being able to do well on new data. This makes it easier for the model to do well on new data.

Dropout is a regularization method that is often used. During training, it randomly removes units from the neural network along with their links. This keeps neurons from co-adapting and pushes the network to learn stronger traits. Weight decay, which is also called L2 regularization, makes big weights in the model less useful by adding a regularization

term to the loss function. When it comes to NAS, regularization methods are very important because they keep the new designs from fitting the training data too well. You can add domain-specific regularization requirements to make sure that design features that are important for the target task or application area are followed [13].

Dropout Algorithm is as follows

Step 1: Initialization:

- Initialize parameters: W (weights) and b (biases).

Step 2: Training Phase:

- For each iteration:

Forward pass: Compute

$$z = f(Wx + b) \quad (5)$$

- Apply dropout: Randomly mask neurons with dropout rate p.
- Scale activations:

$$z_{dropout} = \left(\frac{1}{(1-p)} \right) * z_{masked} \quad (6)$$

- Backward pass: Compute gradients and update parameters.

Step 3: Inference Phase:

- During inference:

Forward pass: Compute

$$z_{inference} = p * z \quad (7)$$

Step 4: Hyperparameter Tuning:

- Tune dropout rate p.

3. Result Analysis and Discussion

The results of five tests are shown in Table 1. Each one was judged on four key performance metrics: Accuracy, Precision, Recall, and F1 Score. The accuracy was 90.2%, which shows the percentage of properly identified cases. It was 89.6% accurate, which was measured by the number of successfully expected positive observations compared to the total number

Table 1
Evaluation parameter PPO algorithm

Accuracy (%)	Precision (%)	Recall (%)	F1 Score
90.2	89.6	88.9	94.7
92.5	93.3	90.9	93.0
93.1	94.9	91.6	92.2
90.9	89.2	90.8	93.0
93.5	94.2	94.3	92.7

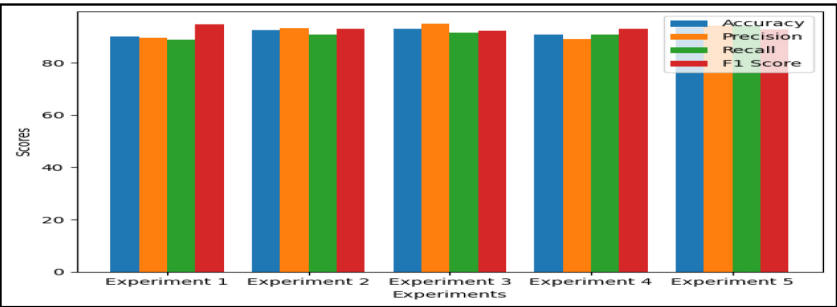


Figure 2
Performance evaluation of PPO

of projected positives. The recall score, which shows how many real hits were correctly found, got 88.9%. Also, F1 Score of 94.7%, which is the harmonic mean of accuracy and memory. With a score of 92.5%.

Figure 2 shows a bar graph that shows the success measures of several studies. Each experiment is shown by a different bar group. The x-axis shows the names of each experiment, and the y-axis shows the scores for different rating factors, such as F1 score, accuracy, precision, and memory. The worth of a certain evaluation measure for that experiment is shown by the height of each bar in a group. For instance, the first set of bars shows Experiment 1, and each bar shows its F1 score, accuracy, precision, and memory. It gives information about the different pros and cons of each experiment in terms of getting the results that were wanted.

The success measures of five tests are shown on the line graph in the figure 3. Each experiment is shown by a line. The experiment numbers are shown on the x-axis, and the percentage scores for accuracy, precision, memory, and F1 Score are shown on the y-axis. The related measure changes in different studies, as shown by each line. By looking at the

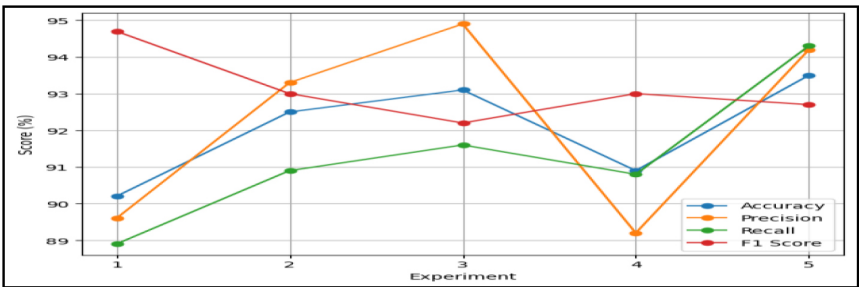


Figure 3
Representation of performance metric using Line graph

Table 2
Performance Metric of Dropout Technique

Epoch	Accuracy (%)	Precision (%)	Recall (%)	F1 Score
1	91.204	93.552	89.913	91.599
2	88.275	91.171	86.543	88.81
3	93.197	94.103	91.492	93.484
4	89.793	92.194	87.626	89.88
5	92.965	95.644	94.681	92.479

direction of each line, watchers can see patterns across the tests, such as changes in performance or gains. There is a clear and straightforward outline of how the models that were evaluated do across different metrics in this graph. This makes it easy to compare and understand how well they do at getting the desired outcomes.

The given results in table 2, show how well the tests with table 2 worked in terms of F1 score, accuracy, precision, and memory. The most

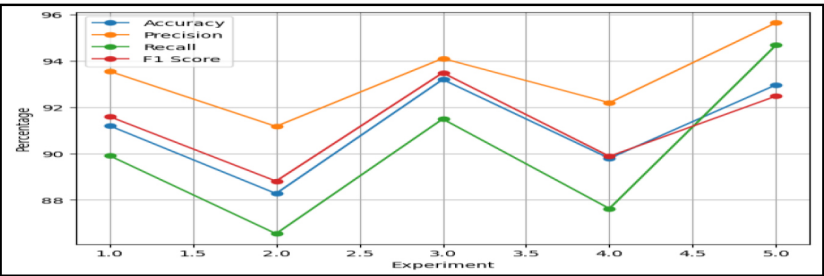


Figure 4
Representation of performance metric of Dropout Technique using Line graph

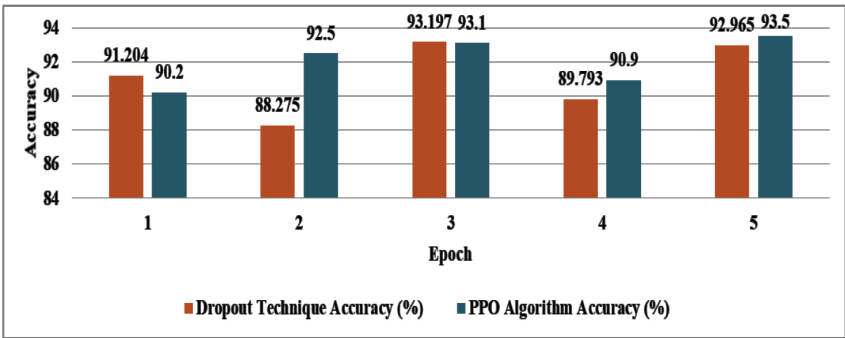


Figure 5
Accuracy Comparison: Dropout Technique vs. PPO Algorithm

accurate experiment was Epoch 3, which got 93.197% right. Epoch 5 came in second, with 92.965%. Precision, which is the percentage of right positive guesses out of all positive predictions, was highest in Epoch 2 (91.171%). Recall, which is the percentage of real positives that were correctly found, was highest in Epoch 5 (94.681%). In addition, Epoch 1 had the best F1 score (91.599%), which means that accuracy and memory were matched.

The performance metrics of five tests were shown graphically in figure 4 as a line graph. These metrics are accuracy, precision, recall, and F1 score. The graph has an x-axis with Epoch numbers and a y-axis with percentage values.

Each Epoch is shown by a line. The graph shows how each measure changes from one Epoch to the next, making it easy to compare and figure out performance trends. The lines show which Epochs do better or worse based on the criteria that were used to judge them, accuracy comparison shown in figure 5. They offer information about the relative strengths and flaws of each Epoch.

4. Conclusion

Dynamic Neural Architecture Search is a big step forward in the field of deep learning. It provides a potential way to automatically find high-performing neural network designs that are perfect for certain tasks. By using dynamic search strategies like reinforcement learning algorithms, NAS algorithms explore the huge space of possible designs, quickly moving through a huge number of setups to find the ones that work best. So, dynamic NAS algorithms can find designs that combine model

complexity and performance, which makes deep learning models work better and be more accurate. NAS algorithms can impose design qualities that are important to the target task or application area by using domain-specific constraints and regularization methods. This makes the found models even faster and more efficient. To sum up, Dynamic Neural Architecture Search looks like a good way to automate the creation of neural network structures, making it easier to solve a wide range of deep learning problems quickly and correctly.

References

- [1] C. Peng, Y. Li, R. Shang, and L. Jiao, "ReCNAS: Resource-Constrained Neural Architecture Search Based on Differentiable Annealing and Dynamic Pruning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 2, pp. 2805–2819 (Feb. 2024).
- [2] H. Bouzidi, M. Odema, H. Ouarnoughi, M. A. Al Faruque, and S. Niar, "HADAS: Hardware-Aware Dynamic Neural Architecture Search for Edge Performance Scaling," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, Antwerp, Belgium, pp. 1–6 (2023).
- [3] L. Sun, Y. Liu, B. Li, H. Wang, and Z. Zhang, "CellNet: An Improved Neural Architecture Search Method for Coal and Gangue Classification," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Shenzhen, China, pp. 1–9 (2021).
- [4] S. Zhao, L. He, X. Xie, J. Lin, and Z. Wang, "Automatic Generation of Dynamic Inference Architecture for Deep Neural Networks," in *Proc. IEEE Workshop Signal Processing Systems (SiPS)*, Coimbra, Portugal, pp. 117–122 (2021).
- [5] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems* (Aug. 2021), doi: 10.1109/TNNLS.2021.3093754.
- [6] Y. Xu, X. Lin, Y. Wang, Q. Li, and H. Xu, "Partially-connected neural architecture search for reduced computational redundancy," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2953–2970 (Sep. 2021).
- [7] Q. Zhou, B. Zhong, X. Liu, and R. Ji, "Attention-based neural architecture search for person re-identification," *IEEE Transactions*

- on Neural Networks and Learning Systems* (May 2021), doi: 10.1109/TNNLS.2021.3072999.
- [8] W. Lou, J. Zhang, Y. Liang, X. Li, and Y. Xie, "Dynamic-OFA: Runtime DNN Architecture Switching for Performance Scaling on Heterogeneous Embedded Platforms," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)* (2021).
 - [9] Y. Xu, L. Xie, X. Zhang, X. Chen, G. Huang, and Q. Tian, "Latency-aware differentiable neural architecture search," *arXiv preprint arXiv:2001.06392* (2020).
 - [10] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791* (2019).
 - [11] Y. Han, D. Zhang, X. Li, C. Li, and M. Tan, "Dynamic neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021), doi: 10.1109/TPAMI.2021.3051283.
 - [12] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Machine Learning (ICML)*, pp. 550–559 (2018).
 - [13] D. K. Mane, S. Deshmukh, P. M. Durgawale, S. T. Shirkande, S. T. Deokate, and N. P. Sable, "Privacy-preserving patient monitoring in healthcare IoT using attribute-based cryptography," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 27, no. 2-A, pp. 513–524 (2024), doi: 10.47974/JDMSC-1896.
 - [14] S. Rahman and E. Katya, "Deep Learning for Medical Diagnosis and Prognosis," *International Journal of Advanced Computer Engineering and Communication Technology (IJACECT)*, vol. 12, no. 2, pp. 25–32 (Apr. 2025).

Received October, 2024