Research Article

Alexandr Udeneev, Petr Babkin, and Oleg Bahteev

Surrogate assisted diversity estimation in neural ensemle search.

https://doi.org/10.1515/sample-YYYY-XXXX
Received Month DD, YYYY; revised Month DD, YYYY; accepted Month DD, YYYY

Abstract: Automated search for optimal neural architectures (NAS) is expensive, and extending it to ensembles (NES) can lead to exponential growth in computational cost due to the combinatorial number of architecture combinations. To address this, we propose a surrogate-based approach: each candidate architecture is encoded as a graph, its predictions on a held-out dataset form the training set for a surrogate model, and this model guides an efficient NES framework that selects both diverse and high-performing networks. Our final ensemble matches human-level accuracy on CIFAR-10 and outperforms other one-shot NES methods, demonstrating the practical effectiveness of the approach.

Keywords: NES, surrogate function, triplet loss.

1 Introduction

Neural network ensembles often demonstrate better accuracy compared to single models, especially in classification and regression tasks [1, 2]. This fact gives rise to the problem of constructing an efficient ensemble of models (NES) [3]. NES, in turn, relies on Neural Architecture Search (NAS) methods, which are extensively studied and applied to search for individual neural network architectures, such as evolutionary algorithms [4, 5], reinforcement learning [6–8], and Bayesian optimization [9, 10]. Selecting an optimal architecture for even a single model is a challenging task, particularly when considering data-specific constraints and computational limitations [11].

The simplest approach for ensemble construction is DeepEns [12]. It involves a random search for several architectures, which are then combined into an ensemble. Despite its simplicity in implementation and hyperparameter tuning, this method is computationally expensive. More sophisticated adaptation of NAS techniques are presented in some recent works [3, 13, 14], which are designed to efficiently combine multiple networks into an ensemble.

Our research also adapts ideas from NAS for NES, specifically utilisation a surrogate function [15–17]. Some modern NAS methods widely use surrogate functions to estimate architecture quality without requiring full model training. These functions significantly reduce computational costs, expanding the applicability of such methods. For example, in [15], evolutionary algorithms were proposed in combination with surrogate models for real-time semantic segmentation. In [17], a Surrogate-assisted Multiobjective Evolutionary-based Algorithm (SaMEA) is used for 3D medical image segmentation.

In this work, we propose a method for constructing neural network ensembles using a surrogate function that accounts for both model classification accuracy and architectural diversity. Diversity is crucial because ensembles consisting of similar models often fail to provide a significant performance gain. Two surrogate functions are used in the work: the first to represent the architecture in the latent space [18], the second to predict the accuracy of the architecture. Since a neural network architecture is represented as a graph, using a Graph Attention Network (GAT) [19] as a surrogate function [20] seems natural. To train it to predict model diversity, we use Triplet Loss [21], similar to [18]. We validate this approach on CIFAR-10, demonstrating the effectiveness of the surrogate function for predicting diversity and constructing ensembles.

We claim that ensembles constructed in this manner achieve state-of-the-art accuracy compared to one-shot NES algorithms, such as DeepEns [12].

Main Contributions:

1) We propose a method for encoding the DARTS [22] search space into a representation suitable for training a Graph Attention Network [19] (GAT), where graph nodes correspond to operations within the network.

2) We propose a way for training the surrogate function to predict the diversity of architectures.

3) We adapt surrogate functions for ensemble construction, taking into account both predictive performance and architectural diversity.

2 Problem statement

2.1 Neural Architecture Search

Let us consider a set of nodes $\mathcal{V} = \{x_1, \dots, x_N\}$, representing the layers of a neural network. Additionally, let $\mathcal{O}$ denote the set of possible operations that can be applied to these nodes (e.g., convolutions or poolings). Furthermore, let $\mathcal{A}$ be the set of feasible architectures, represented as vectors.

Denote $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ as the training and validation losses, respectively. The NAS problem can then be formulated as the search for an optimal architecture α^* that minimizes $\mathcal{L}_{val}(\alpha^*, \omega^*)$, under the constraint that the weights are obtained by minimizing the training loss:

$$\omega^* = \arg\min_{\omega \in \mathcal{W}} \mathcal{L}_{train}(\alpha^*, \omega)$$

This can be expressed as the following optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}_{val}(\omega^*(\alpha), \alpha) \\ &\text{s.t.} \quad \omega^*(\alpha) = \arg\min_{\omega \in \mathcal{W}} \mathcal{L}_{train}(\omega, \alpha) \end{aligned} \tag{1}$$

The primary challenge in this optimization lies in the immense search space of possible architectures (e.g., in DARTS [22], it is approximately 10^{25}).

2.2 Neural Ensemble Search

The primary objective of NES is to find an optimal ensemble of neural networks whose architectures lie within the NAS search space.

As before, we denote $\alpha \in \mathcal{A}$ as a network architecture and $\omega(\alpha)$ as its corresponding weights. The action of this network on an input x is denoted by $f_\alpha(x, \omega(\alpha))$. Let $S \subset \mathcal{A}$ be a subset of architectures. Then, the NES problem can be formally described as follows:

$$\begin{aligned} &\min_{\mathcal{S}} \mathcal{L}_{val}\left(\frac{1}{|S|}\sum_{\alpha \in S} f_\alpha(x, \omega^*(\alpha))\right) \\ &\text{s.t.} \quad \forall \alpha \in \mathcal{S} : \; \omega^*(\alpha) = \arg\min_{\omega(\alpha)} \mathcal{L}_{train}(f_\alpha(x, \omega(\alpha))) \end{aligned} \tag{2}$$

Thus, in addition to searching over a vast number of architectures, we now also need to find the optimal ensemble composition.

3 Method

In this work, we consider the transformation of the architecture space proposed in DARTS [22] for application in Graph Attention Networks (GAT) (see Section 3.1). In Section 3.2, we present the architecture of the surrogate function and describe its working principle, while Section 3.3 provides a detailed discussion of the ensemble construction method based on this surrogate function.

3.1 Architecture Search Space

A critical aspect of our methodology involves constructing a training dataset for the surrogate model. While we adopt the DARTS framework [22], which defines normal and reduction cells, our approach diverges fundamentally in how these cells are generated. Specifically, instead of optimizing cells through continuous relaxation, we generate both cell types via discrete random sampling.

Each cell is represented as a directed acyclic graph (DAG) comprising n nodes and m edges, where every edge corresponds to an operation selected from the DARTS operation set $\mathcal{O}$. Formally, a cell architecture is defined as:

$$\alpha_{\text{cell}} = (G, \mathbf{o}), \quad G = (\mathcal{V}, \mathcal{E}), \quad \mathbf{o} = \{o_e\}_{e \in \mathcal{E}}, \quad o_e \in \mathcal{O}, \tag{3}$$

where $\mathcal{V}$ denotes the set of n latent nodes, $\mathcal{E}$ represents the m directed edges, and $\mathcal{O}$ is the predefined operation space.

The full architecture space $\mathcal{A}$ consists of pairs of such DAGs (denoted as normal and reduction cells):

$$\mathcal{A} = \{(G_1, \mathbf{o_1}) \times (G_2, \mathbf{o_2}) \mid G_1, G_2 \text{ are DAGs}, |\mathcal{V}_1| = |\mathcal{V}_2| = n, |\mathcal{E}| = m, \mathbf{o}_1, \mathbf{o}_2 \in \mathcal{O}^m\}. \tag{4}$$

To train the surrogate model, we generate N architectures, evaluate their performance on a fixed validation dataset, and compile the training dataset:

$$\mathcal{D}_{\text{train}} = \{(\alpha_i, \mathbf{y}_i, \text{acc}_i)\}_{i=1}^{N}, \tag{5}$$

where $\mathbf{y}_i$ is the vector of predictions from architecture α_i on the validation set, and acc_i is its validation accuracy.

Unlike DARTS, which employs continuous relaxation and gradient-based optimization, our method relies on discrete sampling to explore the search space. This strategy enables broader diversity in architecture generation at the expense of computational efficiency.

Implementation Details
Each cell contains $n = 5$ nodes, with two outgoing edges per node assigned random operations from $\mathcal{O}$, yielding $m = 10$ edges per cell. The operation set $\mathcal{O}$ includes:
- Separable convolutions (kernel sizes: 3×3, 5×5)
- Dilated separable convolutions (kernel sizes: 3×3, 5×5)
- 3×3 max pooling
- 3×3 average pooling
- Identity operation

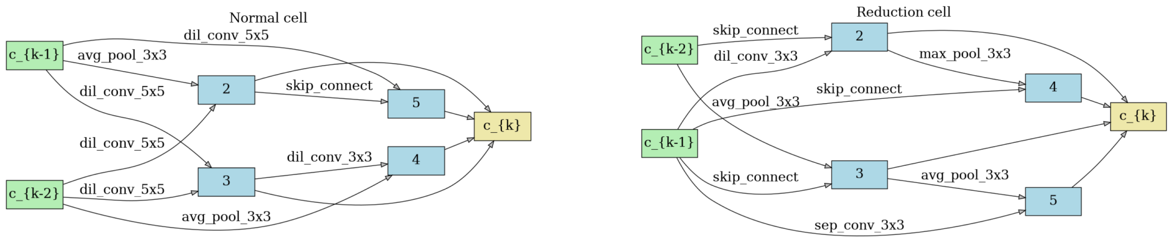

Fig. 1: Illustration of normal and reduction cell structures used in our architecture search space. The normal cell (left) preserves spatial dimensions while the reduction cell (right) halves them.

Full models are constructed by stacking normal and reduction cells in a 2:1 ratio, starting from a predefined number of initial channels. Each model combines one normal and one reduction cell sampled independently from $\mathcal{A}$.

The total number of unique cell architectures is computed as:

$$|\mathcal{A}_{\text{cell}}| = \binom{2}{2}\binom{3}{2}\binom{4}{2}\binom{5}{2} \cdot |\mathcal{O}|^8, \quad |\mathcal{O}| = 7, \tag{6}$$

resulting in a complete architecture space of:

$$|\mathcal{A}| = |\mathcal{A}_{\text{cell}}|^2 \approx 10^{18}. \tag{7}$$

This vast combinatorial space renders exhaustive search computationally intractable, necessitating efficient surrogate-guided exploration.

3.2 Surrogate Function

In order to construct the ensemble described in Section 3.3, we need to be able to predict both the performance and the similarity of the models. However, due to the enormous number of architectures, directly obtaining these characteristics is unfeasible because of the excessive time requirements.

To address this challenge, we propose to employ a surrogate function — a model that, given an architectural representation, can predict key characteristics of neural network models, such as accuracy or their latent embeddings. Formally surrogate function defined as:

$$f_{acc} : \mathcal{A} \to [0, 1] \quad f_{sim} : \mathcal{A} \to \mathbb{R}^l$$

where l is dimension of latent space

Currently, each architecture in the dataset is represented as a directed acyclic graph (DAG), where the nodes correspond to latent representations and the edges to operations, similar to the format presented in NAS-Bench-201 [23]. For example, the DARTS architectures consist of connected normal and reduction cells (in a 2:1 ratio), as shown in Figure 2.

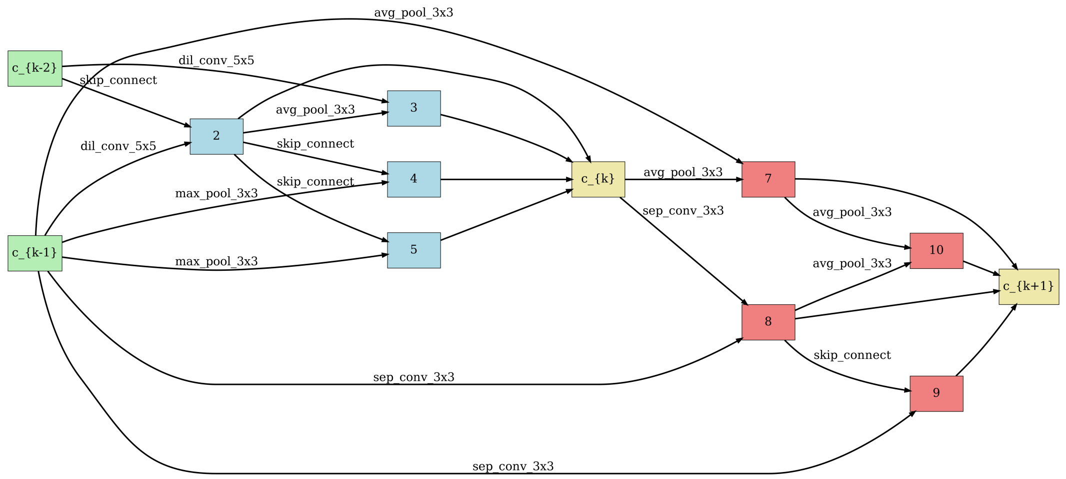

Fig. 2: Combined normal and reduced cells. The red vertices belong to the reduction cell; the blue vertices belong to the normal cell.

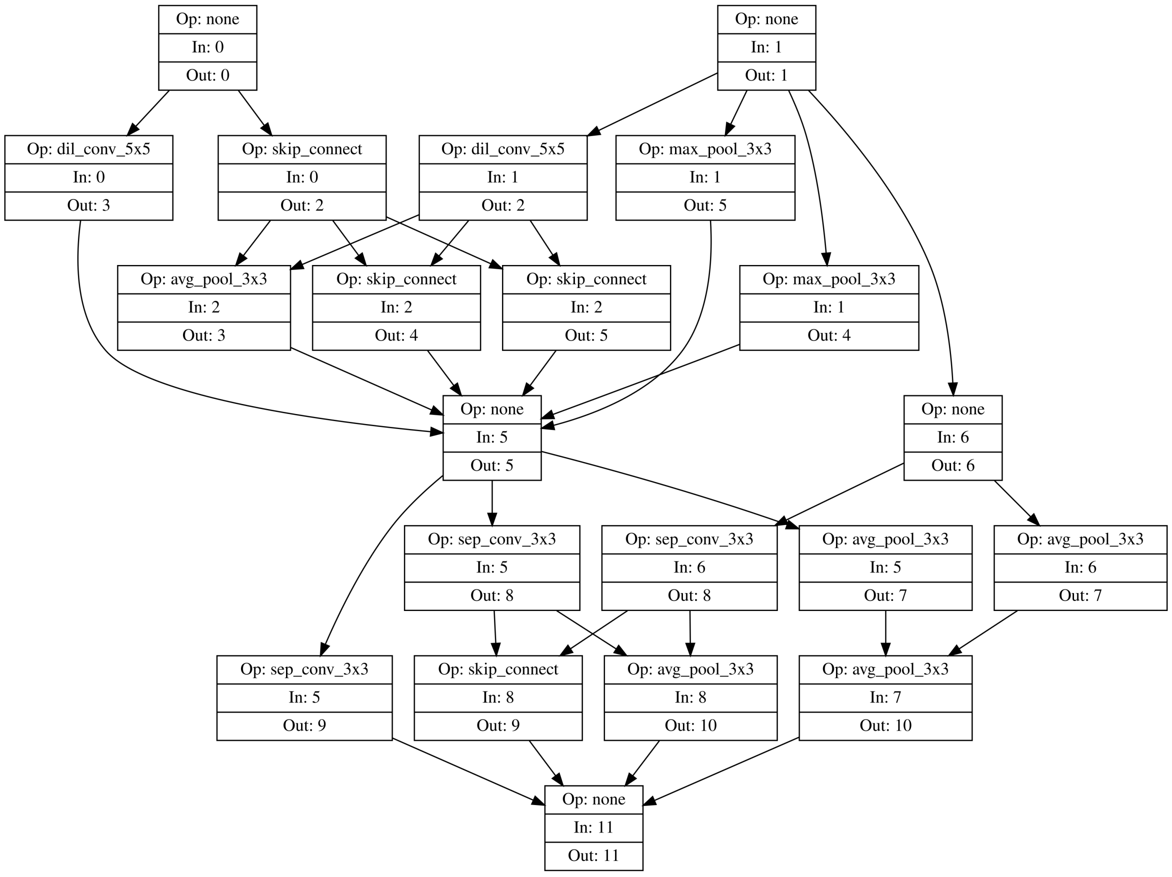
Op: none In: 0 Out: 0
Op: none In: 1 Out: 1
Op: dil_conv_5x5 In: 0 Out: 3
Op: skip_connect In: 0 Out: 2
Op: dil_conv_5x5 In: 1 Out: 2
Op: max_pool_3x3 In: 1 Out: 5
Op: avg_pool_3x3 In: 2 Out: 3
Op: skip_connect In: 2 Out: 4
Op: skip_connect In: 2 Out: 5
Op: max_pool_3x3 In: 1 Out: 4
Op: none In: 5 Out: 5
Op: none In: 6 Out: 6
Op: sep_conv_3x3 In: 5 Out: 8
Op: sep_conv_3x3 In: 6 Out: 8
Op: avg_pool_3x3 In: 5 Out: 7
Op: avg_pool_3x3 In: 6 Out: 7
Op: sep_conv_3x3 In: 5 Out: 9
Op: skip_connect In: 8 Out: 9
Op: avg_pool_3x3 In: 8 Out: 10
Op: avg_pool_3x3 In: 7 Out: 10
Op: none In: 11 Out: 11

Fig. 3: Conversion of an architecture to the NAS-Bench-101 format.

Subsequently, it is more convenient to adopt an alternative graph representation: the *nodes* represent operations and the *edges* their corresponding latent embeddings. In this way, the architecture is transformed into the NAS-Bench-101 format [24] (see Figure 3).

The conversion is carried out as follows:

1. Each edge of the original graph is transformed into a node in the new graph; the label of this node indicates the operation present on the corresponding edge in the original graph.
2. An oriented edge is drawn from the node with label o_i to the node with label o_j if, in the original graph, the operation o_i acts from a node x to a node y, and the operation o_j acts from node y to a node z (with x, y, and z being arbitrary nodes).

We encode the operations as one-hot vectors. Since our dataset consists of graphs (architectural encodings), we adopt a Graph Attention Network (GAT) as the surrogate function, as in [25].

For predicting accuracy, supervised learning can be employed; however, to train the model to predict similarity, we utilize the Triplet Loss [21]. As in [18], Triplet Loss is employed to learn the latent representations of architectures. Similarity between two models can be defined, for instance, by comparing the fraction of identical predictions or by computing the average distance between their output distributions using a suitable metric for multivariate distributions, such as a divergence or distance function in the probability space (e.g., Jensen–Shannon divergence or Hellinger distance). This allows us to construct a similarity matrix of size $N \times N$.

In this article, we construct a similarity matrix based on model responses on the validation dataset. The matrix is constructed as follows:
1. For each of the N models, predictions are computed on a fixed validation dataset consisting of K examples. Denote the predictions of model M_i by the vector
$$\mathbf{y}^{(i)} = \left(y_1^{(i)}, \dots, y_K^{(i)}\right).$$
2. For every pair of models (M_i, M_j), where $i, j = 1, \dots, N$, the fraction of matching predictions is computed:
$$s_{ij} = \frac{1}{K} \sum_{k=1}^{K} \mathbb{I}\left(y_k^{(i)} = y_k^{(j)}\right),$$
where $\mathbb{I}$ is the indicator function.
3. The values s_{ij} form the symmetric similarity matrix $\mathbf{S} \in [0,1]^{N \times N}$, with each entry $\mathbf{S}_{ij}$ representing the degree of similarity between models M_i and M_j.
4. To discretize the similarity for the purposes of Triplet Loss, the matrix $\mathbf{S}$ is converted into a discrete matrix $\mathbf{M} \in \{-1, 0, 1\}^{N \times N}$ according to the following rule:
$$\mathbf{M}_{ij} = \begin{cases} 1, & \text{if } s_{ij} > q_p, \\ -1, & \text{if } s_{ij} < q_n, \\ 0, & \text{otherwise,} \end{cases}$$
where q_p and q_n are predetermined thresholds corresponding to the upper and lower quantiles of the distribution of s_{ij} values.

The training algorithm for surrogate similarity function is as follows:

Algorithm 1 Training the surrogate model for architecture diversity

Input: f_{sim}: an untrained surrogate model; $\mathcal{B}$: a set of architectures of pretrained models; N: the number of architectures; $\mathbf{M}$: a discrete similarity matrix of size $N \times N$; n: the number of training epochs; `optimizer`: the optimization algorithm; m: the margin parameter for the Triplet Loss.

Output: Trained surrogate model f_{sim}

```
1  for $i \leftarrow 1$ to $n$ do
2      for $j \leftarrow 1$ to $N$ do
3          $\mathcal{P}_j \leftarrow \{k \mid \mathbf{M}[j,k] = 1\}$ // Sample a positive example
4          $k_p \leftarrow \text{UniformSample}(\mathcal{P}_j)$
5          $\mathcal{N}_j \leftarrow \{k \mid \mathbf{M}[j,k] = -1\}$ // Sample a negative example
6          $k_n \leftarrow \text{UniformSample}(\mathcal{N}_j)$
           // Compute embeddings
7          $\mathbf{e}_a \leftarrow f_{sim}(\mathcal{B}[j])$
8          $\mathbf{e}_p \leftarrow f_{sim}(\mathcal{B}[k_p])$
9          $\mathbf{e}_n \leftarrow f_{sim}(\mathcal{B}[k_n])$
           // Compute the triplet loss
10         $\mathcal{L} \leftarrow \max\left(0, ||\mathbf{e}_a - \mathbf{e}_p||_2^2 - ||\mathbf{e}_a - \mathbf{e}_n||_2^2 + m\right)$
           // Optimization step
11         optimizer.zero_grad()
12         $\mathcal{L}$.backward()
13         optimizer.step()
14     end
15 end
16 return $f_{sim}$
```

3.3 Ensemble construction

Our basic algorithm is shown in the diagram 4

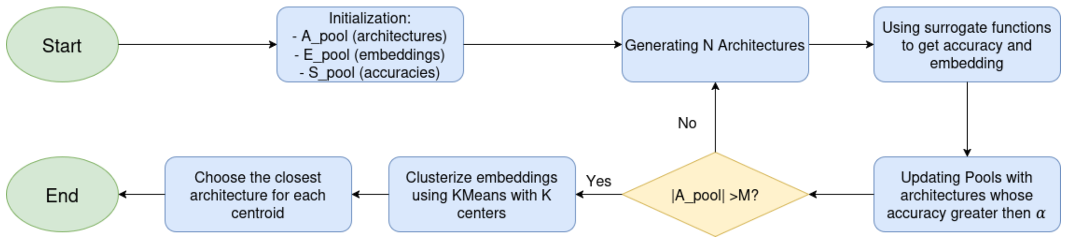
Start
Initialization:
- A_pool (architectures)
- E_pool (embeddings)
- S_pool (accuracies)
Generating N Architectures
Using surrogate functions to get accuracy and embedding
Updating Pools with architectures whose accuracy greater then α
|A_pool| >M?
No
Yes
Clusterize embeddings using KMeans with K centers
Choose the closest architecture for each centroid
End

Fig. 4: Ensemble construction algorithm.

The algorithm 2 begins by initializing three sets: $\mathcal{A}_{pool}$, which stores candidate architectures whose predicted accuracy exceeds a predefined threshold α; $\mathcal{E}_{pool}$, which holds the corresponding diversity embeddings; and $\mathcal{S}_{pool}$, which contains the predicted accuracy scores obtained from surrogate models.

In each iteration, a batch of N candidate architectures, denoted by $\mathcal{A}_{cand}$, is generated. For each architecture, both the predicted accuracy and a representation in the latent diversity space are computed using the surrogate functions f_{acc} and f_{div}, respectively. Only those architectures with predicted accuracy above the threshold α are added to the pool.

Once the number of architectures in $\mathcal{A}_{pool}$ reaches or exceeds M, the diversity embeddings $\mathcal{E}_{pool}$ are clustered into K groups using the KMeans algorithm. For each cluster center, the architecture whose embedding lies closest to the center (in Euclidean distance) is selected. The resulting K architectures form the final ensemble, aiming to balance both predictive performance and diversity.

Algorithm 2 Ensemble Construction

```
Input: $K$: ensemble size; $M$: minimum pool size ($M \geq K$); $N$: number of architectures generated per
       iteration; $\alpha$: accuracy threshold; $f_{\text{acc}}$: surrogate model for accuracy prediction; $f_{\text{div}}$: surrogate model
       for diversity embeddings;
Output: $\mathcal{A}_{\text{best}}$: set of $K$ selected architectures
1  $\mathcal{A}_{\text{pool}} \leftarrow \emptyset$ // Initialize architecture pool
2  $\mathcal{E}_{\text{pool}} \leftarrow \emptyset$ // Initialize diversity embedding pool
3  $\mathcal{S}_{\text{pool}} \leftarrow \emptyset$ // Initialize pool of predicted accuracy scores
4  while $|\mathcal{A}_{pool}| < M$ do
5      $\mathcal{A}_{\text{cand}} \leftarrow \text{GenerateArchs}(N)$ // Generate N random candidate architectures
6      $\mathbf{s} \leftarrow f_{\text{acc}}(\mathcal{A}_{\text{cand}})$ // Predict accuracy for candidates
7      $\mathbf{e} \leftarrow f_{\text{div}}(\mathcal{A}_{\text{cand}})$ // Predict diversity embeddings for candidates
8      for $(a, s, e) \in zip(\mathcal{A}_{cand}, \mathbf{s}, \mathbf{e})$ do
9          if $s \geq \alpha$ then
               // Updating pools
10             $\mathcal{A}_{\text{pool}} \leftarrow \mathcal{A}_{\text{pool}} \cup \{a\}$
11             $\mathcal{E}_{\text{pool}} \leftarrow \mathcal{E}_{\text{pool}} \cup \{e\}$
12             $\mathcal{S}_{\text{pool}} \leftarrow \mathcal{S}_{\text{pool}} \cup \{s\}$
13         end
14     end
15 end
16 $\mathcal{C} \leftarrow \text{KMeans}(\mathcal{E}_{\text{pool}}, K)$ // Cluster diversity embeddings into K groups
17 $\mathcal{A}_{\text{best}} \leftarrow \emptyset$ // Initialize set of best architectures
18 foreach $c \in \mathcal{C}$ do
19     $\mu_c \leftarrow c.\text{center}$
20     $i^* \leftarrow \arg\min_i \|\mathcal{E}_{\text{pool}}[i] - \mu_c\|_2$ // Find index of embedding closest to cluster center
21     $\mathcal{A}_{\text{best}} \leftarrow \mathcal{A}_{\text{best}} \cup \{\mathcal{A}_{\text{pool}}[i^*]\}$ // Add selected architecture to best set
22     Remove index $i^*$ from $\mathcal{A}_{\text{pool}}$, $\mathcal{E}_{\text{pool}}$, and $\mathcal{S}_{\text{pool}}$ // Remove selected index from pools
23 end
24 return $\mathcal{A}_{best}$
```

4 Computational Experiment

In this section we present the experimental results as well as the metrics used for comparison. In Section 4.1, we describe the dataset collection procedure. Section 4.2 details the training setup for the surrogate functions. Finally, in Section 4.3, we compare our proposed method against DeepEnsemble and Random Search.

4.1 Dataset Construction

The models in our dataset were generated according to the following algorithm:

1. Split the CIFAR-10 dataset into training and validation subsets with an 80%/20% ratio.
2. Sample "normal" and "reduction" cells from the DARTS search space, ensuring each node has exactly two incoming edges from previous nodes.

3. Assemble a DARTS architecture consisting of two normal cells followed by one reduction cell.
4. Train each sampled architecture on the training subset.
5. For each trained model, record:
   - its architectural description,
   - its predictions on the validation set,
   - its validation accuracy.

Table 1 summarizes the training hyperparameters for all sampled models:

Tab. 1: Training Hyperparameters for Dataset Models

Hyperparameter	**Value**
Number of epochs	**80**
Optimizer	**Adam**
Learning rate	**0.001**
Training set size	**48 000**
Validation set size	**12 000**
Batch size	**256**

The resulting pool of models exhibited the following distributions:

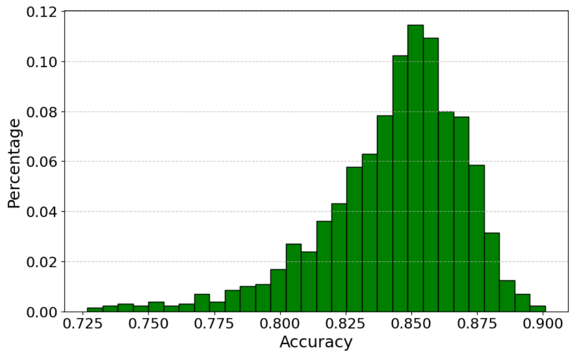

Fig. 5: Distribution of model accuracies

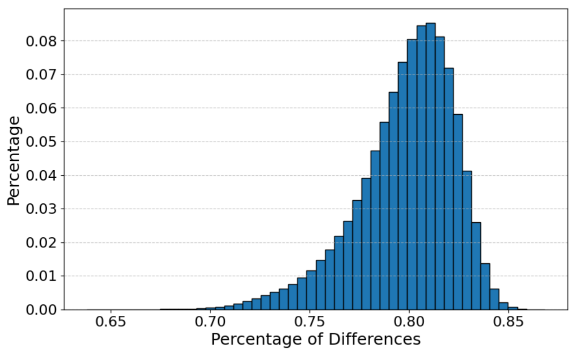

Fig. 6: Distribution of inter-model diversity

4.2 Training of the Surrogate Functions

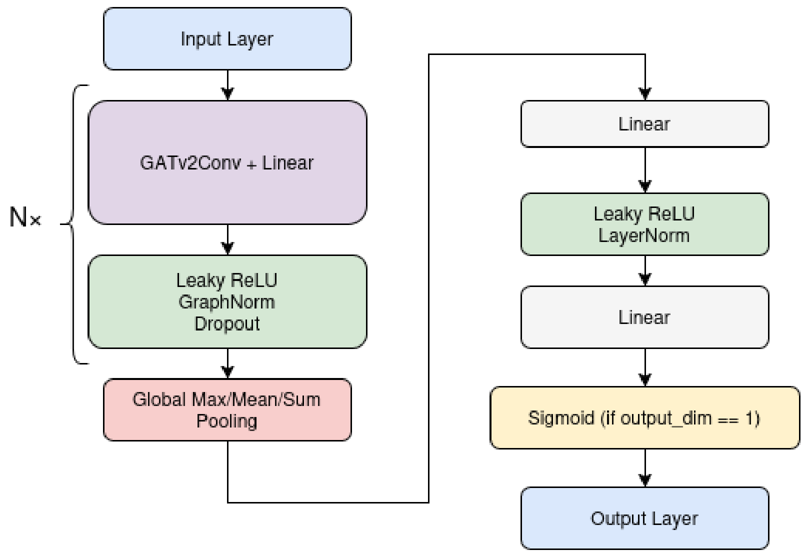

Fig. 7: Architecture of the surrogate function (GAT with 4 convolutional layers and 2 fully connected output layers)

We employed a graph attention network (GAT) with four convolutional layers and two fully connected layers at the output, i.e. N=4 (see Figure 7). In both surrogate training regimes, the learning rate was halved every 5 epochs.

Tab. 2: Training Hyperparameters for Diversity Surrogate

Hyperparameter	Value
Number of epochs	30
Optimizer	Adam
Learning rate	0.001
Dropout	0.1
Heads	4
Training set size	1040
Validation set size	260

In the diversity regime, we train the GAT surrogate with an output dimensionality of 128. The learning rate is halved every 5 epochs, and dropout is kept low (10%) to encourage stable attention weights. The model attends with 4 heads over the graph structure, using 30 epochs of Adam optimization on 1040 training examples, with validation on 260 instances.

Tab. 3: Training Hyperparameters for Accuracy Surrogate

Hyperparameter	Value
Number of epochs	50
Optimizer	Adam
Learning rate	0.01
Dropout	0.4
Heads	16
Training set size	1040
Validation set size	260

For accuracy prediction, the surrogate outputs a single scalar (via sigmoid activation) estimating architecture performance. We use a higher dropout (40%) and more attention heads (16) to regularize and diversify feature aggregation. The model is trained for 10 epochs on the same dataset split.

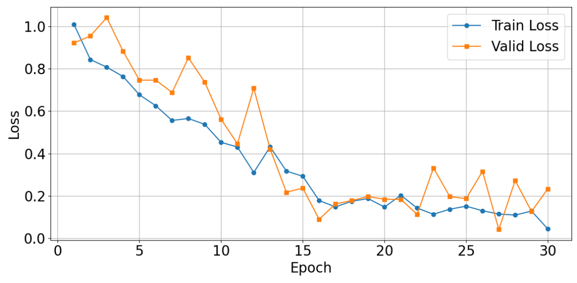

Fig. 8: Training curve of the surrogate model for diversity

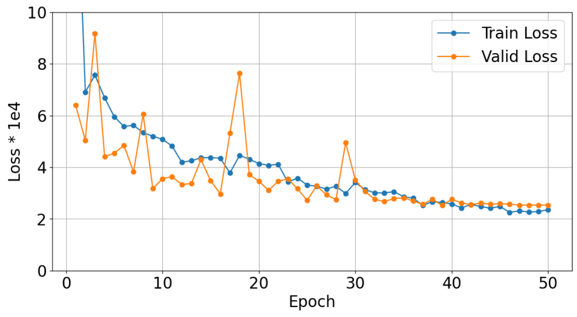

Fig. 9: Training curve of the surrogate model for accuracy

4.3 Experimental Results on Ensemble Construction

We compare our surrogate-assisted NES ensemble construction method against DARTS [26] and a pure Random Search baseline [26]. Our ensemble comprises six models. Training was performed for 100 epochs with a batch size of 96 images, using the Adam optimizer (learning rate 0.025,cosine learning scheduler, weight decay 10^{-4}). The total training time was 11 hours on NVIDIA P100 GPU.

Table 4 presents the test accuracies of the compared methods.

Tab. 4: Comparison of ensemble test accuracies.

Model	Test Accuracy (%)
Surrogate-assisted NES (ours)	**94.2**
DeepEns (DARTS) [26]	**97.38**
Random Search [26]	**97.29**

References

[1] P.N. Suganthan Ye Ren, Le Zhang. Ensemble classification and regression-recent developments, applications and future directions [review article]. *IEEE Computational Intelligence Magazine*, 11(1):41–53, Feb 2016. ISSN 1556-603X. 10.1109/mci.2015.2471235. URL https://doi.org/10.1109/mci.2015.2471235.

[2] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(10): 993–1001, 1990. 10.1109/34.58871.

[3] Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris C. Holmes, Frank Hutter, and Yee Whye Teh. Neural ensemble search for uncertainty estimation and dataset shift. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 7898–7911, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/41a6fd31aa2e75c3c6d427db3d17ea80-Abstract.html.

[4] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International conference on machine learning*, pages 2902–2911. PMLR, 2017.

[5] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[6] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

[7] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[8] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G. Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE Trans. Neural Networks Learn. Syst.*, 34(2):550–570, 2023. 10.1109/TNNLS.2021.3100554.

[9] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1946–1956, 2019.

[10] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. *Advances in neural information processing systems*, 31, 2018.

[11] Ankit Kumar Saroj Kumar Pandey Neeraj varshney Teekam Singh Chetan Swarup, Kamred Udham Singh. Brain tumor detection using cnn, alexnet amp;amp; googlenet ensembling learning approaches. *Electronic Research Archive*, 31(5): 2900–2924, 2023. ISSN 2688-1594. 10.3934/era.2023146. URL https://doi.org/10.3934/era.2023146.

[12] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[13] Yao Shu, Yizhou Chen, Zhongxiang Dai, and Bryan Kian Hsiang Low. Neural ensemble search via bayesian sampling. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 1803–1812. PMLR, 01–05 Aug 2022. URL https://proceedings.mlr.press/v180/shu22a.html.

[14] Haibin Ling Minghao Chen, Jianlong Fu. One-shot neural ensemble architecture search by diversity-guided search space shrinking. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16525–16534, Jun 2021. 10.1109/cvpr46437.2021.01626. URL https://doi.org/10.1109/cvpr46437.2021.01626.

[15] Zhichao Lu, Ran Cheng, Shihua Huang, Haoming Zhang, Changxiao Qiu, and Fan Yang. Surrogate-assisted multi-objective neural architecture search for real-time semantic segmentation. *CoRR*, abs/2208.06820, 2022. 10.48550/ARXIV.2208.06820.

[16] Zhichao Lu, Kalyanmoy Deb, Erik D. Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I*, volume 12346 of *Lecture Notes in Computer Science*, pages 35–51. Springer, 2020. 10.1007/978-3-030-58452-8_3.

[17] Maria G. Baldeon Calisto and Susana K. Lai-Yuen. Emonas-net: Efficient multiobjective neural architecture search using surrogate-assisted evolutionary algorithm for 3d medical image segmentation. *Artif. Intell. Medicine*, 119:102154, 2021. 10.1016/J.ARTMED.2021.102154.

[18] Ferrante Neri Yu Xue, Zhenman Zhang. Similarity surrogate-assisted evolutionary neural architecture search with dual encoding strategy. *Electronic Research Archive*, 32(2):1017–1043, 2024. ISSN 2688-1594. 10.3934/era.2024050. URL https://doi.org/10.3934/era.2024050.

[19] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *CoRR*, abs/1710.10903, 2017. 10.48550/arxiv.1710.10903. URL http://arxiv.org/abs/1710.10903.

[20] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European conference on computer vision*, pages 660–676. Springer, 2020.

[21] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018. 10.48550/arxiv.1806.09055. URL http://arxiv.org/abs/1806.09055.

[23] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

[24] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *CoRR*, abs/1902.09635, 2019. 10.48550/arxiv.1902.09635. URL http://arxiv.org/abs/1902.09635.

[25] Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *CoRR*, abs/1912.00848, 2019. 10.48550/arxiv.1912.00848. URL http://arxiv.org/abs/1912.00848.

[26] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR, 2020.