Annotation

In our research, we investigate a novel method for sampling ensembles of deep learning models using a hypernetwork. The hypernetwork is a neural network that controls the diversity of the models by translating a parameter representing ensemble diversity into a neural network architecture. Architectures are obtained in a one-shot manner by perturbing from the base architecture. We evaluate the performance of the proposed algorithm by conducting experiments on the CIFAR-100 dataset, validating the method and comparing the resulting ensembles with those sampled by other search algorithms.

Keywords: differentiable search · neural ensembles · hypernetwork · diversity control

Content

Introduction

Nowadays methods of neural architecture search (NAS) are well-explored and proved to be an effective way of creating more efficient neural networks [1–4]. On the other hand, neural ensemble search (NES) is modern and not as well investigated problem as NAS, although it is well-established that ensembles of deep learning models exhibit superior performance and possess greater robustness compared to individual models [5–7].

A straightforward approach to constructing neural network ensembles involves generating multiple random initializations of the architecture search algorithm. Ensembles created through this way (DeepEns) are capable of achieving higher predictive capabilities compared to methods for searching single model architectures [8; 9]. Subsequently, the field progressed, and various more efficient approaches to constructing DeepEns emerged, taking into account model diversity and striving to search for them more effectively [10–12]. However, the sequential training of models is fraught with substantial computational costs, as even a single run of the architecture search process is a computationally intensive task [13].

Contemporary researchers are interested in constructing ensembles of neural networks in a one-shot manner [5;13]. This approach aims to circumvent the computational overhead associated with sequential model training by generating an ensemble of architectures simultaneously. The premise behind one-shot ensemble generation lies in the efficient exploration of the architecture space, seeking for diverse and well-performing architectures.

As Neural Architecture Search (NAS) methods are more established and well-investigated, while Neural Ensemble Search (NES) represents a more recent yet closely related domain, NES techniques frequently build upon the investigations conducted within the NAS field. Our method is no exception and is grounded in the DARTS [14] methodology.

The distinctive feature of the DARTS algorithm is the smoothing of the architecture space. The continuous space facilitates a transition from discrete optimization to continuous optimization, in which solutions are significantly easier to find using gradient-based methods [15]. This approach inspired researchers to develop extensions of the DARTS solution, with proposed improvements to balance the search and evaluation processes [16; 17], optimize the solution search from a memory perspective [18], and control the complexity of the final solution [19]. Our paper leverages the ideas introduced by DARTS to search for ensembles of neural networks.

The idea behind our method is illustrated in Figure 1. Our ensemble construction method builds upon a known optimal architecture, sampling models similar to it in terms of shared edges. To control diversity, we introduce a diversity parameter λ. We posit that the farther an architecture is from the optimal one in this space, the worse is its performance. Under these conditions, we strive to find an ensemble that strikes a balance between exploring the architecture space and maintaining high predictive capability for each individual architecture.

To sample architectures we employ the concept of a hypernetwork [20]. A hypernetwork is a neural network that generates parameters for another neural network, referred to as the target network. In our case, the hyperparameter of the target network is its architecture. In prior research, hypernetworks have been utilized to modulate different characteristics, such as architectural complexity [19] or parameter configurations of the target model [21], across several contemporary studies. Hypernetworks have also been used to search for the architecture of a single network [22], and our method employs a similar approach. In our work, the hypernetwork is employed to modulate the diversity of the target models.

The hypernetwork uses the introduced understanding of the difference, namely, the number of distinct connections with the optimal network. Consequently, the architectures obtained from it vary in terms of connections, which results in a diversity of responses [13]. To train the hypernetwork, we employ a regularizer based on the smoothed count of shared connections with the optimal architecture, which enables the hypernetwork to learn to sample the desired network architectures for different diversity parameters λ. After training the hypernetwork, one can sample a set of architectures from it, whose diversity can be controlled, and each of which shows high performance on the validation dataset.

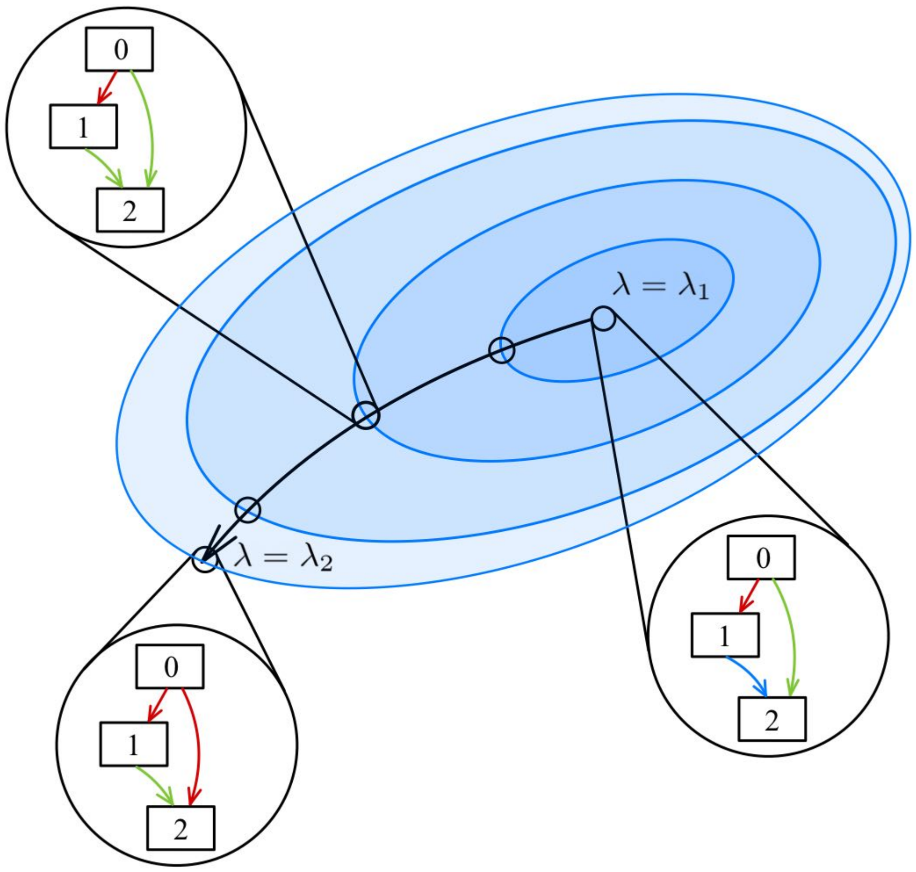

Figure 1 — The architecture space is endowed with a distance metric induced by the difference in edges. Architectures vary in terms of the diversity parameter λ. The blue ellipses represent equidistant surfaces in the metric of this space. The optimal architecture corresponds to $\lambda = \lambda_1$, and as the diversity parameter changes from λ_1 to λ_2, the architecture undergoes a complete transformation.

Thus, our method is capable of generating ensembles of deep learning models in a one-shot manner, which places it in line with the works [5; 13]. At the same time, it contrasts with the works [23; 24], where ensembles are generated sequentially, taking more time for training.

We conducted extensive experiments with our method on the CIFAR-100 dataset, pursuing several objectives. Firstly, we aimed to validate the hypernetwork in terms of its ability to generate architectures that can yield effective models with controlled diversity. Secondly, we sought to verify the efficiency gains of our accelerated method. Lastly, we investigated the effectiveness of ensembling the obtained networks. Regarding parameter sharing, our approach leverages this technique to enable efficient generation of diverse architectures within the hypernetwork framework.

Notation. We use bold lowercase letters $\mathbf{y}$ to denote vectors, and bold uppercase letters $\mathbf{X}$ to denote matrices. In our paper, we address the problem of supervised learning. We assume that the dataset $\mathcal{D} = (\mathbf{X}, \mathbf{y})$ is given, where $\mathbf{X}$ represents the feature matrix and $\mathbf{y}$ represents the target vector. The dataset is divided into training and validation subsets, for which the loss functions $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ are specified, respectively. In general, these functions depend on the target vector and the predicted outputs of the training model. However, in our paper, we will not explicitly state their dependence on the dataset, assuming it implicitly. Therefore, we will state that the loss functions depend only on the predicted outputs of the model. Moreover, when appropriate, we shall assume that the model outputs, and hence the loss function, depends on the model architecture and its parameters.

1. Problem Statement

1.1 Neural Architecture Search

Let $\mathcal{V} = \{1, \dots, N\}$ be a set of vertices, where N is a number of vertices, and $\mathcal{E} = \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i < j\}$ a set of edges between them. A set of possible operations in architecture $\mathcal{O}$ usually contains pooling, convolutions, etc. For each edge there is an operation $o \in \mathcal{O}$ that transits information from one node to another. Thus, the task of neural architecture search is reduced to the task of finding operations $o^{(i,j)}$ for each edge (i, j). Considering $\boldsymbol{\alpha} \in \mathcal{A}$ as the vector of parameters indicating the operations within each edge, the NAS problem can be formally written in the following way:

$$\begin{aligned} &\min_{\boldsymbol{\alpha} \in \mathcal{A}} \mathcal{L}_{val}(\boldsymbol{w}^*_{\boldsymbol{\alpha}}, \boldsymbol{\alpha}), \\ s.t. \quad &\boldsymbol{w}^*_{\boldsymbol{\alpha}} = \arg \min_{\boldsymbol{w} \in \mathcal{W}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha}), \end{aligned} \tag{1.1}$$

where $\mathcal{W}$ is the set of all possible operation parameters between all possible edges in the architecture. In this problem, the challenge lies in the fact that the architecture space is exponentially large with respect to the number of vertices.

1.2 Neural Ensemble Search

Contrary to the selection of one single architecture in conventional NAS algorithm [14;22;25], this paper focuses on the problem of selecting a well-performing neural network ensemble with diverse architectures from the NAS search space, i.e. neural ensemble search (NES).

In order to facilitate our study, several key terms are defined. The architecture of a model, i.e. a set of node operations, is denoted by $\boldsymbol{\alpha}$. For a fixed architecture, optimal parameters are denoted by $\boldsymbol{w}^*_{\boldsymbol{\alpha}}$. The output of an architecture $\boldsymbol{\alpha}$, given its corresponding model parameters $\boldsymbol{w}_{\boldsymbol{\alpha}}$, is denoted by $f(\boldsymbol{w}_{\boldsymbol{\alpha}}, \boldsymbol{\alpha})$. Finally, the set of architectures included in the ensemble is denoted by $S \in \mathcal{S}^n$, where $\mathcal{S}^n$ is a set of all ensembles of size n.

Given the ensemble scheme, NES can be formally framed as

$$\begin{aligned} &\min_{S \in \mathcal{S}^n} \mathcal{L}_{val} \left(\frac{1}{|S|} \sum_{\boldsymbol{\alpha} \in S} f(\boldsymbol{w}^*_{\boldsymbol{\alpha}}, \boldsymbol{\alpha}) \right), \\ s.t. \quad &\forall \boldsymbol{\alpha} \in S \quad \boldsymbol{w}^*_{\boldsymbol{\alpha}} = \arg\min_{\boldsymbol{w} \in \mathcal{W}} \mathcal{L}_{train}(f(\boldsymbol{w}, \boldsymbol{\alpha})). \end{aligned} \tag{1.2}$$

In this task, there is an issue of an exponentially large architecture space inherited from NAS. Additionally, there is a problem of an exponentially large number of architecture combinations in the resulting ensemble S. To address these challenges, a method of architecture space smoothing has been devised.

1.3 Architectural Space

NAS algorithms search for optimal architecture. As it was mentioned below, architecture of neural network is a set of operations between nodes. In differentiable NAS methods architecture is a vector constructed by following rules. For each edge $(i, j) \in \mathcal{E}$, $\boldsymbol{\alpha}^{(i,j)} \in \mathbb{R}^{|\mathcal{O}|}$ is a vector, which assigns impact of each operation [14].

In our paper, we substantially employ the edge-normalization approach [18], which has demonstrated promising results in enhancing architecture search. In this approach, the magnitude of contribution varies not only for operations within edges but also for the edges themselves. Thus, the algorithm modification assists in selecting not only operations within edges but also the presence of edges themselves. This enables the resulting architectures to be more compact and uncluttered. For each vertex $i \in \mathcal{V}$, $\boldsymbol{\beta}^{(i)}$ is a vector that represents the contribution of edges leading to a given node.

Concatenation of all structural parameters vectors $\boldsymbol{\alpha}^{(i,j)}$ and $\boldsymbol{\beta}^{(i)}$ is denoted as $\boldsymbol{\alpha} \in \mathcal{A}$, where are $\mathcal{A} = \mathbb{R}^s$. This vector contains all information about smoothed architecture. From this vector, the resulting discrete architecture can be obtained through a straightforward application of one-hot choice.

2. Method

2.1 Shared Connections Regularizer

The proposed method constructs an ensemble, starting from the optimal architecture, which is obtained by solving the optimization problem (1.1). Subsequently, the method searches for architectures that differ from the initial one by selecting alternative connections. In doing so, the method seeks the most optimal way to replace a certain number of connections. To accomplish this step, we employ a regularizer in our work, which facilitates obtaining the desired architectures during the optimization process.

Transitioning to a more formal description of the method, let $\boldsymbol{\alpha}^*$ denote the optimal architecture obtained by solving the optimization problem (1.1). To quantify architectural diversity, we define λ, a parameter ranging from 0 to Λ, where Λ is the predefined maximum number of edges in the resulting architectures.

To compute the number of shared edges with the optimal architecture $\boldsymbol{\alpha}^*$, we employ the dot product of the parameter vectors of these architectures. For architecture smoothing, we apply the Gumbel-softmax operation [26]. It incorporates a temperature parameter t that governs the degree of discretization of the resulting architecture, thus allowing control over the regularizer's stringency on par with the regularizer weight parameter.

Given the parameter λ we reformulate problem (1.1) in the following way

$$\begin{gathered}\min_{\boldsymbol{\alpha}} \mathcal{L}_{val}(\boldsymbol{w}^*, \boldsymbol{\alpha}) + c(\lambda - \langle \boldsymbol{\alpha}^*, GS(\boldsymbol{\alpha}) \rangle)^2, \\ s.t. \quad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha}),\end{gathered} \tag{2.1}$$

where c is weight of the regularizer. In addressing the given task, one can obtain architectures that differ from the original by a certain number of edges, controlled by the diversity parameter λ. At the same time, the discovered architectures will be optimal for the specified number of edges. The latter statement has been validated through computational experimentation.

2.2 Diversity Control via Hypernetwork

In the previous section, a method for controlling diversity for two architectures was discussed. However, to construct an ensemble, a means of controlling the diversity of a set of architectures is required. To address this task, we employ the concept of hypernetwork [20]. A hypernetwork in our research is a parametric mapping from $[0, \Lambda]$ to the set of model structural parameters [19]

$$\boldsymbol{\alpha} : [0, \Lambda] \times \mathbb{R}^u \to \mathbb{R}^s,$$

where $\mathbb{R}^u$ is a hypernetwork parametric space and $\mathbb{R}^s$ is space of model structural parameters. In this terms $\boldsymbol{\alpha}$ can be redefined using hypernetwork

$$\boldsymbol{\alpha} = \boldsymbol{\alpha}(\lambda, \boldsymbol{a}),$$

where $\boldsymbol{a}$ is a parameter vector of the hypernetwork. In this paper $\boldsymbol{\alpha}$ is a piecewise-linear function, i.e.

$$\boldsymbol{\alpha}(\lambda, \boldsymbol{a}) = \sum_{k=0}^{K-1} \left(\frac{\lambda - r_k}{r_{k+1} - r_k} \boldsymbol{a_k} + \left(1 - \frac{\lambda - r_k}{r_{k+1} - r_k}\right) \boldsymbol{a_{k+1}} \right) I[\lambda \in [r_k, r_{k+1}]], \tag{2.2}$$

where $\boldsymbol{a_k}$ and r_k are trainable parameters for each k and K is predefined amount of pivots. The training of the hypernet is a key task for ensemble construction in our work. Let us formulate the training objective as

$$\begin{aligned} &\min_{\boldsymbol{a} \in \mathcal{A}} \mathbb{E}_{\lambda \sim p(\Lambda)} [\mathcal{L}_{val}(\boldsymbol{w}^*, \boldsymbol{\alpha}(\lambda, \boldsymbol{a})) + c(\lambda - \langle \boldsymbol{\alpha}^*, GS(\boldsymbol{\alpha}(\lambda, \boldsymbol{a}))\rangle)^2], \\ &\quad s.t. \quad \boldsymbol{w}^* = \arg\min_{\boldsymbol{w} \in \mathcal{W}} \mathbb{E}_{\lambda \sim p(\Lambda)} [\mathcal{L}_{train}(\boldsymbol{w}, \boldsymbol{\alpha}(\lambda, \boldsymbol{a}))], \end{aligned} \tag{2.3}$$

where $p(\Lambda)$ is a predefined distribution over the set $[0, \Lambda]$. Instead of using the standard softmax function as in DARTS [14], we utilize its improvement, the Gumbel-softmax [19], denoted as GS in (2.3). The Gumbel-softmax is a parametric relaxation of categorical distribution over the descrete set. For our task, it is more effective than the standard softmax, as it allows for controlling the discretization of the distribution through a temperature parameter t, and adds randomization. In our work it is used for distribution over the set of possible operations between nodes and captures the objective of our method: selecting the optimal operation. During optimization process we sample λ from $p(\Lambda)$ and perform optimization steps in two stages.

Due to the structure of the regularizer described in Section 2.1, during the iterative optimization of the network, architectures with many overlapping edges will be sampled. This accelerates the optimization process and enhances the algorithm's efficiency [27;28]. During optimization, the diversity parameter λ is sampled, representing the number of common edges between the optimal architecture and the one being trained in the current iteration. In the process of selecting architectures for the final ensemble, λ is also sampled, and architectures are drawn from the hypernetwork with the corresponding number of edges from the optimal architecture. Consequently, at each iteration, the parameters for the edges that will be included in the final ensemble are updated. To numerically evaluate the number of shared parameters and, consequently, the speed of the optimization process, we formulate Theorem 1.

Theorem 1. *Let us consider the problem of finding an ensemble of size N. Suppose the maximum number of edges in the final architectures is denoted by Λ. Let $p(\lambda) = U(\overline{1, \Lambda})$. Then, the expectation of sampled edges for parameter updates, which will participate in the final ensemble, is estimated by*

$$\mathbb{E}[m] = N\frac{(\Lambda+1)^2}{4\Lambda}. \tag{2.4}$$

Proof. First of all, let us consider that μ is the number of edges from optimal architecture that were chosen on an optimization iteration. Let ν be the number of edges from the optimal architecture sampled during process of ensemble formation. Then the average size of intersection can be calculated as

$$\mathbb{E}[m_i|\mu,\nu] = \nu\mathbb{P}\{A\} = \frac{\mu\nu}{\Lambda},$$

where A is the event that a randomly selected edge was used during the optimization step, $\mathbb{P}\{A\} = \frac{\mu}{\Lambda}$. Secondly, let us calculate unconditional mathematical obedience

$$\begin{aligned}\mathbb{E}[m_n] =&\mathbb{E}_\mu\mathbb{E}_\nu\mathbb{E}[m|\mu,\nu] = \sum_{i=1}^{\Lambda}\sum_{i=1}^{\Lambda}\mathbb{P}\{\mu=i\}\mathbb{P}\{\nu=j\}\frac{ij}{\Lambda} = \\ =&\frac{1}{\Lambda^3}\sum_{i=1}^{\Lambda}\sum_{j=1}^{\Lambda} ij = \frac{1}{\Lambda^3}\frac{(\Lambda+1)\Lambda}{2}\sum_{i=1}^{\Lambda} i = \frac{(\Lambda+1)^2}{4\Lambda}\end{aligned}$$

On every iteration μ and ν for every architecture in the final ensemble is sampled independently, so we can conclude that the average number of trained edges from the final ensemble will be equal to the sum across architectures, i.e.

$$\mathbb{E}[m] = \sum_{n=1}^{N}\mathbb{E}[m_n] = N\frac{(\Lambda+1)^2}{4\Lambda}.$$

□

The Theorem 1 merits further discussion. For comparison, consider the DARTS ensembling algorithm, which involves ensembling multiple runs with different initializations. In this method, the maximum number of edges for which parameters will be updated can be estimated as Λ, since the algorithm can only update parameters for one run at a time. Therefore, we can estimate the relative speed of the DartsEns and proposed hypernet training using expression (2.4) as

$$\frac{m_H}{m_{DE}} \approx N\frac{(\Lambda+1)^2}{4\Lambda^2}, \tag{2.5}$$

where m_H and m_{DE} represent the average number of edges from the final ensemble for which the network parameters are updated every iteration while training hypernetwork and DartsEns respectively.

However, it is necessary to make a few remarks regarding the applicability of the Theorem 1 towards our method. If during the optimization and the sampling of the final ensemble, the realization of $p(\lambda)$ coincides, i.e. $\mu=\nu$, then the parameters for all Λ edges from the final ensemble will be effectively updated, this scenario is not accounted for in the theorem. Additionally, the potential overlap with the remaining edges not included in the optimal architecture is not considered. Based on these arguments, one can conclude that the expression presented in Theorem 1 provides a lower bound on the number of edge intersections sampled during the optimization process and when constructing the ensemble.

During the training of the hypernetwork, it is crucial to strike a balance in the weight of the regularizer. An excessively small weight will prevent the regularizer from serving as a source of diversity, making it impossible to control the diversity of architectures. Conversely, an excessively large weight will inhibit the algorithm's ability to escape local minima induced by the regularizer, hindering its capacity to identify well-performing architectures. The same can be said about the magnitude of discretization, that is, the temperature parameter t in the Gumbel-softmax distribution. An appropriate balance must be maintained for this parameter as well. An overly small value of t would lead to premature discretization, limiting the exploration of the architecture space and potentially trapping the algorithm in sub-optimal solutions. On the other hand, an excessively large value of t would result in slow convergence and inefficient utilization of computational resources, as the search process would remain diffuse and unfocused for an extended period.

2.3 Ensembling Strategy

Summarizing the above, we present an Algorithm 1 to solve the problem (1.2) using differentiable search with controlled diversity of the models. In the first step of the algorithm, an optimal network is obtained by solving the problem defined in Equation (1.1). Subsequently, a hypernetwork (2.3) is trained, where the output from the previous step is used as the optimal architecture. This results in a piecewise-linear hypernetwork capable of generating architectures for the ensemble. The hypernetwork is then applied to generate candidate architectures for the ensemble. In the final stage, n architectures are selected, representing the best-performing ensemble on the validation dataset.

We should notice that after selecting the candidate architectures for the ensemble, it is necessary to retrain them from scratch for subsequent validation. The retraining process is much cheaper than the architecture search process. Thus, it will not significantly impact the overall runtime. Additionally, it is

Algorithm 1 EdgeNES

```
Initialize: $n \in \mathbb{N}$, $\mathcal{S}' = \varnothing, N \in \mathbb{N}$
$\boldsymbol{\alpha}^* \leftarrow$ result of 1.1
Train hypernetwork $\boldsymbol{\alpha}$ using 2.3 and $\boldsymbol{\alpha}^*$
for $i = 1, \dots, N$ do
    Sample $\lambda \sim p(\Lambda)$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{\boldsymbol{\alpha}(\lambda, \boldsymbol{\alpha}^*)\}$
end for
$\mathcal{S} \leftarrow$ best performing ensemble of size $n$ from $\mathcal{S}'$
Return: $\mathcal{S}$ as a resulting ensemble
```

essential to remark on the final stage of the algorithm. To find the optimal ensemble, we propose enumerating combinations of candidate architectures and evaluating their performance on the validation dataset. The number of such combinations will be exponentially large with respect to the ensemble size.

This approach is similar to random search [29], but the key difference lies in the fact that the original search space of DARTS contains $\sim 10^{25}$ architectures, whereas the number of candidate architectures in Algorithm 1 equals N. In other words, we are controllably narrowing the search space.

By controllably restricting the search space, this approach alleviates the computational burden associated with exploring an extremely vast search space, which can be prohibitively expensive. Moreover, the candidate architectures for the ensemble are known to perform well on the validation dataset and exhibit differences in terms of their distinct edges, which provides diversity in their predictions.

As a result, we obtain an ensemble of neural networks, each of which show high scores on the validation data. The diversity of architectures lies in the different connections employed in the selected architectures, which ensures a difference in the responses [13].

3. Preliminary results

3.1 Experimental setup

The entirety of the experimental code can be found at github. Here we provide some details about the experiments conducted in the article. We outline the pipeline of the DARTS method. This algorithm allows for the search of the architecture of a single layer in a network, after which the architecture of this layer is replicated and concatenated to form the final network. Subsequently, the entire final network is trained from scratch, during which the weights in the selected edges are optimized. In our experiments, a single search cell was trained, and then two such cells were taken and trained from scratch. In our experiments, the architecture was trained for 50 epochs, and the retraining also occurred for 50 epochs. The training was performed on a 3070TI GPU. The training of a single epoch of the architecture search stage took approximately one minute.

The network weights were updated with a standard step of 0.025, as taken from the original paper. The architectural parameters were trained with a step of 0.02. The optimizer used in our study was Adam, with parameters $\beta = (0.5, 0.9)$ for both training stages. The seeds utilized during the method's execution have been recorded, for instance, the seeds for generating the ensemble of random architectures were 120-124 for the different experiments.

The hypernetwork was trained for 150 epochs, using 5 pivots, i.e., r_i in the formula 2.2. In this work, the two pivots: 0 and Λ were set as constants and did not change during the training. Hypernet's training was performed in accordance with the formulation given in (2.3). The distribution $p(\lambda)$ was taken to be uniform from 0 to Λ. It is worth noting that, in order for the hypernetwork to be trained properly, λ was sampled from a uniform distribution on $[-\varepsilon, \Lambda+\varepsilon]$, where $\varepsilon = 0.2$. So that the hypernetwork could better learn the distributions for 0 and Λ. Experiments have shown that without this correction, the network is trained incorrectly, and it does not sample architectures containing 0 and Λ edges.

The rationale behind this approach is that by extending the sampling range of λ beyond the $[0, \Lambda]$ interval, the hypernetwork can better capture the boundary conditions at 0 and Λ. This is crucial, as the trained hypernetwork needs to accurately represent the architectural search space, including the extremes of the search space. If the hypernetwork fails to properly model the edges at 0 and Λ, it may not be able to effectively explore the entire architecture space, leading to suboptimal performance. The proposed modification to the sampling distribution ensures that the hypernetwork can robustly learn the desired architectural representations, even at the boundaries of the search space.

3.2 Method validation

In this subsection, we present the preliminary results of hypernetwork training, where we verify the correctness of its operation in several experiments.

The Figure 3.1 shows the dependence of the number of common edges with the optimal architecture on the λ parameter, which is set by the hypernetwork for sampling the architecture.

In the first experiment, we verify the correctness of architecture sampling in terms of the common edges with the optimal architecture. The graph in the Figure 3.1 shows that the hypernetwork samples architectures with the corresponding number of edges, which confirms the correctness of the regularizer and the proper tuning of hyperparameters such as the regularizer weight C and the distribution $p(\lambda)$.

Furthermore, the observed relationship between the λ parameter and the number of common edges suggests that the hypernetwork has learned to effectively navigate the architecture space, prioritizing the exploration of regions that are more likely to contain high-performing models. This is a desirable property, as it indicates that the hypernetwork has developed a meaningful internal representation of the architecture space and can leverage this knowledge to guide the search process.

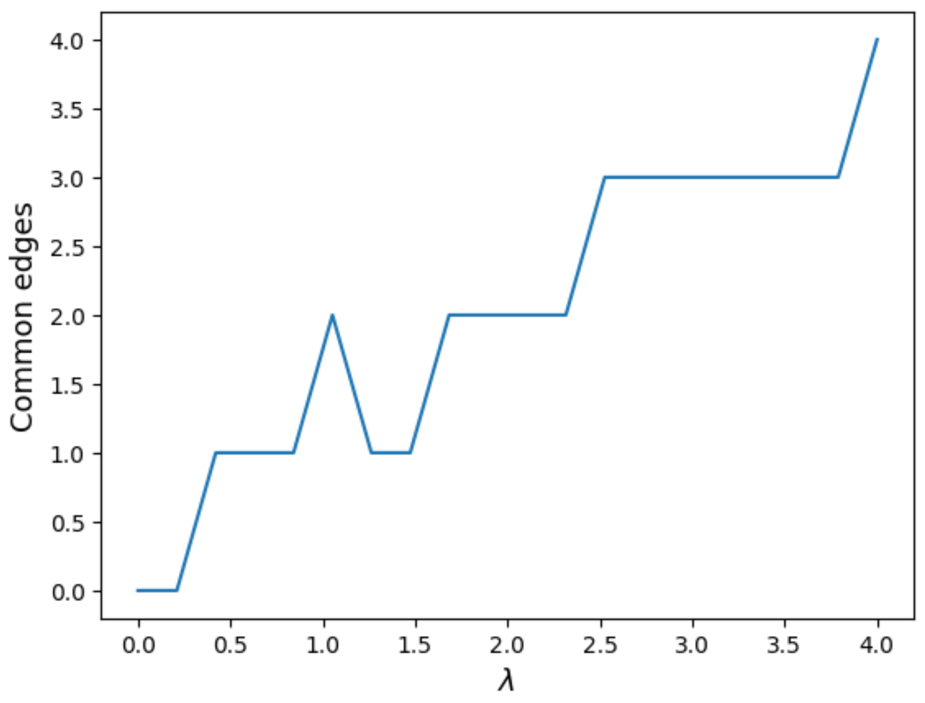

Figure 3.1 — The graph shows the dependence of the number of common edges with the optimal architecture on the λ parameter, which is set by the hypernetwork for sampling the architecture.

The second experiment verifies the hypothesis made in the introduction, that the further the architecture is from the optimal one, the worse it performs. The graph in Figure 3.2 shows that this hypothesis holds true, and the predictive ability of the architectures changes in accordance with the introduced assumption.

3.3 Warming significance

As described in the article, the warm-up technique was used not only with respect to decreasing the temperature, but also with respect to increasing the weight of the regularizer. This approach allows the algorithm to learn the parameters of all the edges more effectively, so that it can then more efficiently select architectures for the ensemble. To verify the effectiveness of this approach, a series of experiments were conducted to train the network with the selection of constant and linearly varying regularizer weights and temperatures

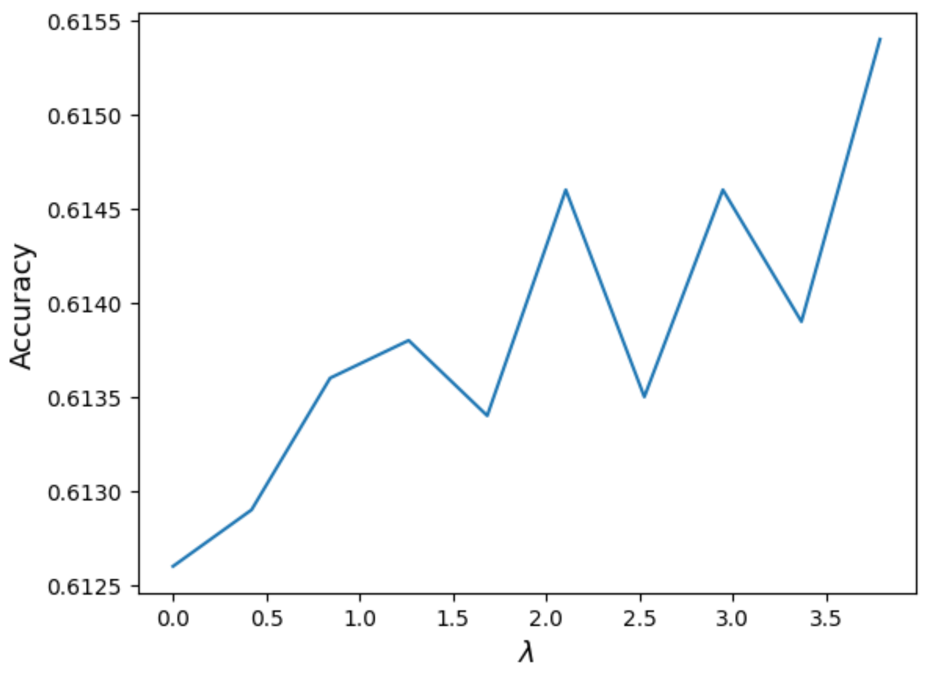

Figure 3.2 — The dependence of the prediction accuracy of the architecture on the deviation from the optimal architecture in terms of the diversity parameter λ.

in the Gumbel-Softmax distribution. The results in Table 1 show that this approach improves the efficiency of the method.

Common edges	**On**	**Off**
1	59.8 ± 1.8	57.3 ± 2.8
2	62.4 ± 2.0	57.7 ± 2.1
3	61.6 ± 0.5	60.2 ± 1.7

Table 1 — Comparison of the effectiveness of the Method with and without the warm-up technique. The table presents the performance accuracy of the method on the CIFAR-10 dataset, comparing the results when the warm-up technique is enabled versus disabled.

The rationale behind this warm-up strategy is that it helps the algorithm overcome the initial challenges in the architectural search process. By gradually increasing the regularizer weight and the temperature of the Gumbel-Softmax distribution, the network is encouraged to explore a broader range of architectural configurations in the early stages of training. This exploration allows the network to learn more robust representations of the edge parameters, which can then be leveraged to more effectively select the final ensemble of architectures.

The experiments demonstrate that this warm-up procedure leads to superior performance compared to using constant values for the regularizer weight and temperature. This suggests that the gradual transition in the search process is crucial for the algorithm to converge to high-performing architectures. The reported results in Table 1 provide empirical evidence for the benefits of this warm-up technique in the context of the studied method.

4. Computational Experiments

In this study, we conduct experiments on the CIFAR-100 dataset [30] using a convolutional neural network (CNN) [3; 31] model to search ensemble in the DARTS [14] search space. Specifically, we utilize a one-layer architecture with six nodes and seven possible operations for each edge. Our optimizer is Adam [32] for both stages of optimization, and we employ a cosine annealing learning rate scheduler. The loss criterion utilized in this study is CrossEntropyLoss.

In our work, instead of setting constant parameters, we employ an annealing technique. Specifically, we linearly increase the weight of the regularizer c and linearly decrease the temperature t in the Gumbel-softmax distribution during the optimization process, i.e.

$$c_i = c_{start} + (c_{end} - c_{start})\frac{e_{current}}{e_{total}} \quad \text{and} \quad t_i = t_{start} + (t_{end} - t_{start})\frac{e_{current}}{e_{total}},$$

where c_{start}, t_{start} are initial values of the weight and temperature, c_{end}, t_{end} are values in the end of the optimization process, $e_{current}, e_{total}$ are current epoch of process and total amount of epochs respectively. By selecting $c_{start} < c_{end}$ and $t_{start} > t_{end}$, we can obtain architectures that exhibit controlled diversity while simultaneously demonstrating robust performance.

In this setup, we conduct a series of experiments to validate the proposed method, investigate speed of learning and evaluate its efficiency.

4.1 Hypernetwrok Validation

In this experiment, we validate results of hypernetwork to assess the predictive capability of the obtained architectures. We employ DARTS to obtain an optimal architecture, and subsequently, we train the hypernetwork using (2.3). To evaluate the effectiveness of the gained hypernetwork, we introduce a controlled perturbation by randomly replacing a certain number of edges in the optimal architecture. This controlled modification allows us to assess the hypernetwork's ability to sample effective architectures that deviate from the optimal configuration. The derived architectures are retrained and evaluated for accuracy performance on a validation dataset.

The results of the seven experimental runs are presented in Table 2, providing an overview of the architectures predictive capabilities. Each cell in the table reports the mean and variance values of accuracy in percents on a validation dataset, enabling a detailed analysis of the prediction accuracy and consistency across different architectural modifications. The top row of the table indicates the number of shared edges with the optimal architecture, facilitating the interpretation of the results in relation to the degree of perturbation introduced.

λ	0	1	2	3
Hypernetwork	46.2 ± 2.5	45.9 ± 2.7	$\mathbf{46.7 \pm 2.8}$	$\mathbf{46.6 \pm 2.1}$
Random Deviation	43.9 ± 7.7	46.1 ± 5.4	45.6 ± 4.9	45.9 ± 2.5

Table 2 — The accuracy performance in percents achieved by the architectures gained from random deviation and from a hypernetwork. The values presented in the table represent the accuracy performance exhibited by the retrained architectures when evaluated on the validation dataset.

The values from the experiment presented in the table allow us to draw several important conclusions. Firstly, the architectures obtained through the use of the hypernet demonstrate statistically significant superiority in accuracy compared to the architectures derived from the random deviation approach. This finding suggests that the hypernet is capable of discovering more optimal architectural configurations tailored to the given task.

Secondly, when examining the architectures obtained through random deviation, an expected trend emerges: as the number of randomly selected edges increases, the dispersion of accuracy also increases, while the mean accuracy decreases. This can be attributed to the fact that random changes in the architecture are more likely to degrade performance than to improve it.

Thirdly, in analyzing the architectures generated by the hypernetwork, a similar pattern can be observed: the further the obtained architecture deviates from the optimal configuration in terms of common connections, the lower its accuracy becomes. This result correlates with the Figure 1. And reinforces the method from a theoretical perspective.

Furthermore, it is noteworthy that the hypernetwork's ability to maintain high performance while exploring new configurations plays a crucial role in creating ensembles of the deep learning models. By navigating the vast search space of possible architectures, the hypernetwork can strike a balance between exploiting the strengths of the original architecture and exploring deviations from it. Consequently, the conducted experiments highlight the advantages of employing a hypernetwork for discovering optimal deviations from the optimal network.

4.2 Convergence Speed

In the following experiment, we compare the performance of the algorithm in terms of accuracy and speed with a baseline. The baseline in our case is the ensembling of DARTS [14]. In other words, multiple runs of the DARTS algorithm are performed, and the obtained architectures are ensembled to produce collective outputs. Despite the simplicity of this method, it serves as a strong baseline in the field of neural network ensembling [33].

A comparative plot is constructed in Figure 4.1, illustrating the relationship between ensembling accuracy and the number of training iterations, which can be equated to training time. According to the graph, the advantage of our proposed method is evident: an efficient ensemble is discovered more rapidly compared to training multiple models individually, corroborating the theoretical estimates presented in Section 2.2. Substituting the data from our experiment into the Theorem 1, we obtain that the method should theoretically run faster than DartsEns.

To confirm the statistical significance of the results, we will conduct a t-test comparison of the areas under the curves (AUCs) for the different methods, which hepls to avoid multiple testing problem. Based on the graph, the methods up to 80 iterations do not provide architectures that perform better

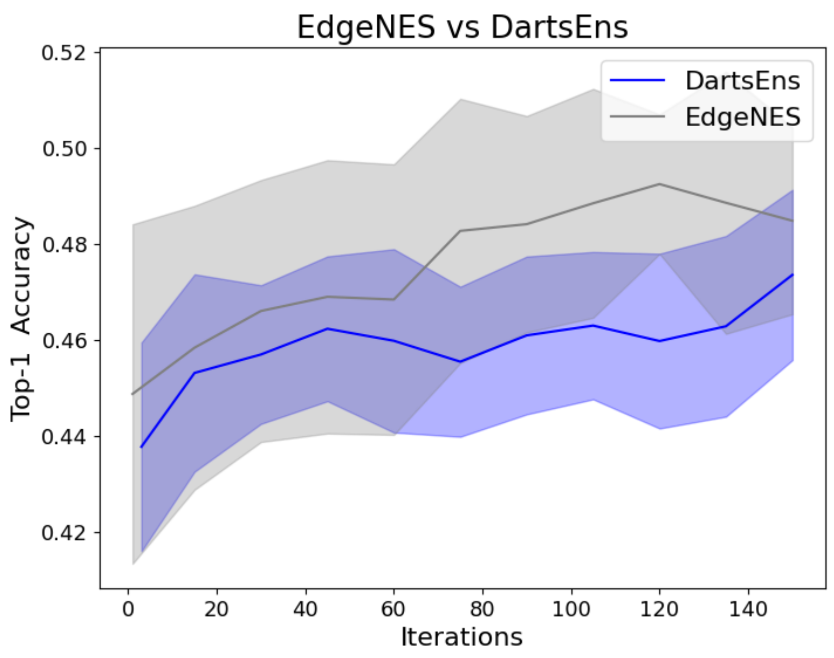

Figure 4.1 — The graphical representation delineates the evolution of predictive accuracy on the test dataset for ensemble configurations, as a function of the training iterations, contrasting the performance trajectories of proposed EdgeNES 1 and the DARTS ensemble approach.

than random. Therefore, the comparison will be made using the graphs starting from iteration 80. The p-value of t-test obtained for the AUCs is equal to 0.0287, which allows us to reject the null hypothesis of equal average AUCs. This indicates that our method yields a statistically significantly better result.

4.3 Ensembling Results

In the final experiment, we conduct a comparative evaluation of the ultimate predictive accuracy achieved on the test dataset and the computational runtime across the various methods under consideration. The results, tabulated in Table 3, encompass both single-model approaches and ensemble-based techniques.

From the values given in Table 3 it is evident that ensemble-based approaches exhibit the potential to achieve higher predictive accuracy with en-

Method	**Test Accuracy, %**	**Search cost (GPU hours)**
DARTS	46.8 ± 1.7	~ 1.2
Random	44.6 ± 6.9	0
DartsEns	$\mathbf{48.9 \pm 1.2}$	~ 3.6
Random-NES	46.9 ± 3.2	0
RandomD-NES	47.7 ± 2.0	~ 1.2
EdgeNES	$\mathbf{48.1 \pm 1.7}$	$\sim \mathbf{2.1}$

Table 3 — A comparative evaluation of the results obtained from diverse methodological approaches, assessing their predictive accuracy on held-out test data and associated computational runtimes.

hanced robustness, thereby corroborating the efficacy of model ensembling techniques in deep learning paradigms.

Furthermore, the proposed EdgeNES method demonstrates a clear superiority over the other methods included in the analysis. Compared to the ensemble-based DARTS approach, EdgeNES operates with greater computational efficiency without compromising predictive accuracy. Additionally, when contrasted with randomly generated architectures, EdgeNES excels in achieving higher accuracy levels and exhibits greater robustness.

From the empirical findings, it can be inferred that the EdgeNES methodology represents a promising avenue for leveraging the benefits of ensemble-based optimization while mitigating the associated computational overhead. Its ability to strike an optimal balance between predictive performance and computational efficiency renders it a compelling choice for practical applications with stringent resource constraints or real-time performance requirements.

Conclusion

In this paper, we proposed a novel method for sampling ensembles of deep learning models with diversity control. Our method utilizes a hypernetwork to generate diverse architectures by perturbing a base architecture in terms of common connections divergence. The diversity of the ensemble is controlled by a penalty term added to the loss function, which encourages the ensemble members to be diverse We conducted extensive experiments on the CIFAR-100 dataset and demonstrated that our method performs compatible results in terms of accuracy. Overall, proposed method shows potential for practical applications in deep learning ensembling.

References

1. *Zoph Barret, Le Quoc V.* Neural architecture search with reinforcement learning // *arXiv preprint arXiv:1611.01578.* — 2016.

2. XNAS: Neural Architecture Search with Expert Advice / Niv Nayman, Asaf Noy, Tal Ridnik et al. // Proc. NeurIPS. — 2019. — Pp. 1975–1985.

3. Learning transferable architectures for scalable image recognition / Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V Le // Proceedings of the IEEE conference on computer vision and pattern recognition. — 2018. — Pp. 8697–8710.

4. Designing neural network architectures using reinforcement learning / Bowen Baker, Otkrist Gupta, Nikhil Naik, Ramesh Raskar // *arXiv preprint arXiv:1611.02167.* — 2016.

5. Multi-headed neural ensemble search / Ashwin Raaghav Narayanan, Arber Zela, Tonmoy Saikia et al. // *arXiv preprint arXiv:2107.04369.* — 2021.

6. Ensemble methods as a defense to adversarial perturbations against deep neural networks / Thilo Strauss, Markus Hanselmann, Andrej Junginger, Holger Ulmer // *arXiv preprint arXiv:1709.03423.* — 2017.

7. *Dietterich Thomas G.* Ensemble methods in machine learning // International workshop on multiple classifier systems / Springer. — 2000. — Pp. 1–15.

8. *Krogh Anders, Vedelsby Jesper.* Neural Network Ensembles, Cross Validation, and Active Learning // Proc. NIPS. — 1994. — Pp. 231–238.

9. *Hansen Lars Kai, Salamon Peter.* Neural Network Ensembles // *IEEE Trans. Pattern Anal. Mach. Intell.* — 1990. — Vol. 12. — Pp. 993–1001.

10. *Fort Stanislav, Hu Huiyi, Lakshminarayanan Balaji.* Deep ensembles: A loss landscape perspective // *arXiv preprint arXiv:1912.02757.* — 2019.

11. *Lakshminarayanan Balaji, Pritzel Alexander, Blundell Charles.* Simple and scalable predictive uncertainty estimation using deep ensembles // *Advances in neural information processing systems.* — 2017. — Vol. 30.

12. Hyperparameter ensembles for robustness and uncertainty quantification / Florian Wenzel, Jasper Snoek, Dustin Tran, Rodolphe Jenatton // *Advances in Neural Information Processing Systems.* — 2020. — Vol. 33. — Pp. 6514–6527.

13. Neural ensemble search via Bayesian sampling / Yao Shu, Yizhou Chen, Zhongxiang Dai, Bryan Kian Hsiang Low // Uncertainty in Artificial Intelligence / PMLR. — 2022. — Pp. 1803–1812.

14. *Liu Hanxiao, Simonyan Karen, Yang Yiming.* Darts: Differentiable architecture search // *arXiv preprint arXiv:1806.09055.* — 2018.

15. *Amari Shun-ichi.* Backpropagation and stochastic gradient descent method // *Neurocomputing.* — 1993. — Vol. 5, no. 4-5. — Pp. 185–196.

16. Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation / Xin Chen, Lingxi Xie, Jun Wu, Qi Tian // Proc. ICCV. — 2019. — Pp. 1294–1303.

17. arXiv:2009.01027: / Xiangxiang Chu, Xiaoxing Wang, Bo Zhang et al.: 2020.

18. Pc-darts: Partial channel connections for memory-efficient architecture search / Yuhui Xu, Lingxi Xie, Xiaopeng Zhang et al. // *arXiv preprint arXiv:1907.05737.* — 2019.

19. Neural Architecture Search with Structure Complexity Control / Konstantin Yakovlev, Olga Grebenkova, Oleg Bakhteev, Vadim Strijov. — EasyChair Preprint no. 7973, EasyChair, 2022.

20. *Ha David, Dai Andrew, Le Quoc V.* Hypernetworks // *arXiv preprint arXiv:1609.09106.* — 2016.

21. Continual learning with hypernetworks / Johannes Von Oswald, Christian Henning, Benjamin F Grewe, João Sacramento // *arXiv preprint arXiv:1906.00695.* — 2019.

22. Smash: one-shot model architecture search through hypernetworks / Andrew Brock, Theodore Lim, James M Ritchie, Nick Weston // *arXiv preprint arXiv:1708.05344.* — 2017.

23. Neural ensemble search for uncertainty estimation and dataset shift / Sheheryar Zaidi, Arber Zela, Thomas Elsken et al. // *Advances in Neural Information Processing Systems.* — 2021. — Vol. 34. — Pp. 7898–7911.

24. *Rahaman Rahul et al.* Uncertainty quantification and deep ensembles // *Advances in neural information processing systems.* — 2021. — Vol. 34. — Pp. 20063–20075.

25. Accelerating neural architecture search using performance prediction / Bowen Baker, Otkrist Gupta, Ramesh Raskar, Nikhil Naik // *arXiv preprint arXiv:1705.10823.* — 2017.

26. *Jang Eric, Gu Shixiang, Poole Ben.* Categorical reparameterization with gumbel-softmax // *arXiv preprint arXiv:1611.01144.* — 2016.

27. Efficient neural architecture search via parameters sharing / Hieu Pham, Melody Guan, Barret Zoph et al. // International conference on machine learning / PMLR. — 2018. — Pp. 4095–4104.

28. SNAS: stochastic neural architecture search / Sirui Xie, Hehui Zheng, Chunxiao Liu, Liang Lin // *arXiv preprint arXiv:1812.09926.* — 2018.

29. *Li Liam, Talwalkar Ameet.* Random search and reproducibility for neural architecture search // Uncertainty in artificial intelligence / PMLR. — 2020. — Pp. 367–377.

30. *Krizhevsky Alex, Hinton Geoffrey et al.* Learning multiple layers of features from tiny images. — 2009.

31. *O'shea Keiron, Nash Ryan.* An introduction to convolutional neural networks // *arXiv preprint arXiv:1511.08458.* — 2015.

32. *Kingma Diederik P., Ba Jimmy.* Adam: A Method for Stochastic Optimization // Proc. ICLR. — 2015.

33. *Tanveer Muhammad Suhaib, Khan Muhammad Umar Karim, Kyung Chong-Min*. Fine-tuning darts for image classification // 2020 25th International Conference on Pattern Recognition (ICPR) / IEEE. — 2021. — Pp. 4789–4796.