NEURAL ARCHITECTURE SEARCH ON IMAGENET IN FOUR GPU HOURS: A THEORETICALLY INSPIRED PERSPECTIVE

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Architecture Search (NAS) has been explosively studied to automate the discovery of top-performer neural networks. Current works require heavy training of supernet or intensive architecture evaluations, thus suffering from heavy resource consumption and often incurring search bias due to truncated training or approximations. Can we select the best neural architectures without involving any training and eliminate a drastic portion of the search cost?

We provide an affirmative answer, by proposing a novel framework called *training-free neural architecture search* (**TE-NAS**). TE-NAS ranks architectures by analyzing the spectrum of the neural tangent kernel (NTK), and the number of linear regions in the input space. Both are motivated by recent theory advances in deep networks, and can be computed without any training. We show that: (1) these two measurements imply the *trainability* and *expressivity* of a neural network; and (2) they strongly correlate with the network's actual test accuracy. Further on, we design a pruning-based NAS mechanism to achieve a more flexible and superior trade-off between the trainability and expressivity during the search. In NAS-Bench-201 and DARTS search spaces, TE-NAS completes high-quality search but only costs **0.5** and **4** GPU hours with one 1080Ti on CIFAR-10 and ImageNet, respectively. We hope our work to inspire more attempts in bridging between the theoretic findings of deep networks and practical impacts in real NAS applications.

1 INTRODUCTION

The recent development of deep networks significantly contributes to the success of computer vision. Thanks to many efforts by human designers, the performance of deep networks have been significantly boosted (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; Szegedy et al., 2015; He et al., 2016; Xie et al., 2017). However, the manual creation of new network architectures not only costs enormous time and resources due to trial-and-error, but also depends on the design experience that does not always scale up. To reduce the human efforts and costs, neural architecture search (**NAS**) has recently amassed explosive interests, leading to principled and automated discovery for good architectures in a given search space of candidates (Zoph & Le, 2016; Brock et al., 2017; Pham et al., 2018; Liu et al., 2018a; Chen et al., 2018; Bender et al., 2018).

As an optimization problem, NAS faces two core questions: 1) "**what to select**", i.e. the objective function that defines what are good architectures we want; 2) "**how to select**", i.e. by what means we could effectively optimize the objective function. These two questions are entangled and highly non-trivial, since the search spaces are of extremely high dimension, and the generalization ability of architectures cannot be easily inferred (Dong & Yang, 2020; Dong et al., 2020). Existing NAS methods mainly leverage the validation set and conduct accuracy-driven architecture optimization. They either formulate the search space as a super-network ("supernet") and make the training loss differentiable through the architecture parameters (Liu et al., 2018b), or treat the architecture selection as a sequential decision making process (Zoph & Le, 2016) or evolution of genetics (Real et al., 2019). However, these NAS algorithms suffer from heavy consumption of both time and GPU resources. Training a supernet till convergence is extremely slow, even with many effective heuristics for sampling or channel approximations (Dong & Yang, 2019; Xu et al., 2019). Approximated proxy inference such as truncated training/early stopping can accelerate the search, but is well known to

introduce search bias to the inaccurate results obtained (Pham et al., 2018; Liang et al., 2019; Tan et al., 2020). The heavy search cost not only slows down the discovery of novel architectures, but also blocks us from more meaningfully understanding the NAS behavors.

On the other hand, the analysis of neural network's trainability (how effective a network can be optimized via gradient descent) and expressivity (how complex the function a network can represent) has witnessed exciting development recently in the deep learning theory fields. By formulating neural networks as a Gaussian Process (no training involved), the gradient descent training dynamics can be characterized by the Neural Tangent Kernel (NTK) of infinite (Lee et al., 2019) or finite (Yang, 2019) width networks, from which several useful measures can be derived to depict the network trainability at the initialization. Hanin & Rolnick (2019a;b); Xiong et al. (2020) describe another measure of network expressivity, also without any training, by counting the number of unique linear regions that a neural network can divide in its input space. We are therefore inspired to ask:

- ***How to select*** *neural architectures at initialization without involving any training, thus significantly eliminating a heavy portion of the search cost?*
- *Can we define* ***what to select*** *in NAS by analyzing the trainability and expressivity of architectures, and further benefit our understanding of the search process?*

Our answers are **yes** to both questions. In this work, we propose TE-NAS, a framework for training-free neural architecture search. We leverage *two indicators*, the condition number of NTK and the number of linear regions, that can decouple and effectively characterize the trainability and expressivity of architectures respectively in the complex NAS search space. Most importantly, these two indicators can be measured in a training-free manner, thus largely accelerates the NAS search process and benefits the understanding of discovered architectures. To our best knowledge, TE-NAS makes the first attempt to bridge between the theoretical findings of deep neural networks and the real-world NAS applications. While we intend not to claim that the two indicators we use are the only nor the best options, we hope our work to open a door to theoretically-inspired NAS and inspire the discovery of more deep network indicators. Our contributions are summarized as below:

- We identify and investigate two training-free indicators to rank the quality of deep architectures: the spectrum of their NTKs, and the number of linear regions in their input space. Our study finds that they reliably indicate the trainability and expressivity of a deep network respectively, and are strongly correlated with the network's test accuracy.
- We leverage the above two theoretically-inspired indicators to establish a training-free AS framework, **TE-NAS**, therefore eliminating a drastic portion of the search cost. We further introduce a pruning-based mechanism, to boost search efficiency and to more flexibly trade-off between trainability and expressivity.
- In NAS-Bench-201/DARTS search spaces, **TE-NAS** discovers architectures with a strong performance at remarkably lower search costs, compared to previous efforts. With just one 1080Ti, it only costs 0.5 GPU hours to search on CIFAR10, and 4 GPU hours on ImageNet, respectively, setting the new record for ultra-efficient yet high-quality NAS.

2 Related Works

Neural architecture search (NAS) is recently proposed to accelerate the principled and automated discovery of high-performance networks. However, most works suffer from heavy search cost, for both weight-sharing based methods (Liu et al., 2018b; Dong & Yang, 2019; Liu et al., 2019; Yu et al., 2020a) and single-path sampling-based methods (Pham et al., 2018; Guo et al., 2019; Real et al., 2019; Tan et al., 2020). A one-shot super network can share its parameters to sampled sub-networks and accelerate the architecture evaluations, but it is very heavy and hard to optimize and sufferes from poor correlation between its accuracy and those of the sub-networks (Yu et al., 2020c). Sampling-based methods achieve more accurate architecture evaluations, but their truncated training still impose bias to the performance ranking since this is based on the results of early training stages.

Instead of estimating architecture performance by direct training, people also try to predict network's accuracy (or ranking), called *predictor based NAS* methods (Liu et al., 2018a; Luo et al., 2018; Dai et al., 2019; Luo et al., 2020). Graph neural network (GNN) is a popular choice as the predictor

model (Wen et al., 2019; Chen et al., 2020). Siems et al. (2020) even propose the first large scale surrogate benchmark, where most of the architectures' accuracies are predicted by a pretrained GNN predictor. The learned predictor can achieve highly accurate performance evaluation. However, the data collection step - sampling representative architectures and train them till converge - still requires extremely high cost. People have to sample and train 2,000 to 50,000 architectures to serve as the training data for the predictor. Moreover, none of these works can demonstrate the cross-space transferability of their predictors. This means one has to repeat the data collection and predictor training whenever facing an unseen search space, which is highly nonscalable.

The heavy cost of architecture evaluation hinders the *understanding* of the NAS search process. Recent pioneer works like Shu et al. (2019) observed that DARTS and ENAS tend to favor architectures with wide and shallow cell structures due to their smooth loss landscape. Siems et al. (2020) studied the distribution of test error for different cell depths and numbers of parameter-free operators. Chen & Hsieh (2020) for the first time regularizes the Hessian norm of the validation loss, and visualizes the smoother loss landscape of the supernet. Still, these analyses cannot be directly leveraged to guide the design of network architectures.

Mellor et al. (2020) recently proposed a NAS framework that does not involve training, which shares the same motivation with us towards training-free architecture search at initialization. They empirically find that the correlation between sample-wise input-output Jacobian can indicate the architecture's test performance. However, why does the Jacobian work is not well explained and demonstrated. Their search performance on NAs-Bench-201 is still left behind by the state-of-the-art NAS works, and they did not extend to DARTs space.

Meanwhile, we see the evolving development of **deep learning theory** on neural networks. NTK (neural tangent kernel) is proposed to characterize the gradient descent training dynamics of infinite wide (Jacot et al., 2018b) or finite wide deep networks (Hanin & Nica, 2019). Wide networks are also proved to evolve as linear models under gradient descent (Lee et al., 2019). This is further leveraged to decouple the trainability and generalization of networks (Xiao et al., 2019). Besides, a natural measure of network's expressivity is the number of linear regions it can separate in its input space (Hanin & Rolnick, 2019a;b; Xiong et al., 2020). In our work, we for the first time discover two important indicators that can effectively rank architectures, thus bridging the theoretic findings and real-world NAS applications. Instead of claiming the two indicators we discover are the best, we believe there are more meaningful properties of deep networks that can benefit the architecture search process. We leave them as open question and encourage the community to study.

3 METHODS

The core motivation of our TE-NAS framework is to achieve architecture evaluation without involving any training, to significantly accelerate the NAS search process and reduce the search cost. In section 3.1 we present our study on two important indicators that reflect the trainability and expressivity of a neural network, and in section 3.2 we design a novel pruning-based method that can achieve a superior trade-off between the two indicators.

3.1 ANALYZING TRAINABILITY AND EXPRESSIVITY OF DEEP NETWORKS

Trainability and expressivity are distinct notions regarding a neural network (Xiao et al., 2019). A network can achieve high generalization performance only if the function it can represent is complex enough and at the same time it can be effectively trained by gradient descent.

3.1.1 TRAINABILITY BY CONDITION NUMBER OF NTK

The trainability of a neural network indicates how effective it can be optimized using gradient descent (Burkholz & Dubatovka, 2019; Hayou et al., 2019; Shin & Karniadakis, 2020). Although some heavy networks can theoretically represent complex functions, they not necessarily can be effectively trained by gradient descent. One typical example is that, even with much more number of parameters, Vgg networks (Simonyan & Zisserman, 2014) usually perform worse and require more special engineering tricks compared with ResNet family (He et al., 2016), whose superior trainability property is studied by Yang & Schoenholz (2017).

Recent work (Jacot et al., 2018a; Lee et al., 2019; Chizat et al., 2019) studied the gradient descent training of neural networks using a quantity called the neural tangent kernel (NTK). The finite width NTK is defined by $\hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}') = J(\boldsymbol{x})J(\boldsymbol{x}')^T$, where $J_{i\alpha}(\boldsymbol{x}) = \partial_{\theta_\alpha} z_i^L(\boldsymbol{x})$ is the Jacobian evaluated at a point $\boldsymbol{x}$ for parameter θ_α, and z_i^L is the output of the i-th neuron in the last output layer L.

Lee et al. (2019) further proves that wide neural networks evolve as linear models using gradient descent, and their training dynamics is controlled by ODEs that can be solved as

$$\mu_t(X_{\text{train}}) = (\mathbf{I} - e^{-\eta \hat{\Theta}_{\text{train}} t}) Y_{\text{train}} \tag{1}$$

for training data. Here $\mu_t(x) = \mathbb{E}[z_i^L(x)]$ is the expected outputs of an infinitely wide network. $\hat{\Theta}_{\text{train}}$ denotes the NTK between the training inputs, and X_{train} and Y_{train} are drawn from the training set $\mathcal{D}_{\text{train}}$. As the training step t tends to infinity we can see that Eq. 1 reduce to $\mu(X_{\text{train}}) = Y_{\text{train}}$.

The relationship between the conditioning of $\hat{\Theta}$ and the trainability of networks is studied by Xiao et al. (2019), and we brief the conclusion as below. We can write Eq. 1 in terms of the spectrum of Θ:

$$\mu_t(X)_{\text{train},i} = (\mathbf{I} - e^{-\eta \lambda_i t}) Y_{\text{train},i}, \tag{2}$$

where λ_i are the eigenvalues of $\hat{\Theta}_{\text{train}}$ and we order the eigenvalues $\lambda_0 \geq \cdots \geq \lambda_m$. As it has been hypothesized by Lee et al. (2019) that the maximum feasible learning rate scales as $\eta \sim 2/\lambda_0$, plugging this scaling for η into Eq. 2 we see that the λ_m will converge exponentially at a rate given by $1/\kappa$, where $\kappa = \lambda_0/\lambda_m$ is the condition number. Then we can conclude that if the κ of the NTK associated with a neural network diverges then it will become untrainable, so we use κ as a metric for trainability:

$$\kappa_{\mathcal{N}} = \frac{\lambda_0}{\lambda_m}. \tag{3}$$

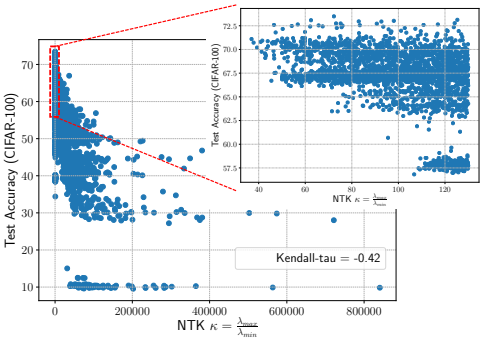


Figure 1: Condition number of NTK $\kappa_{\mathcal{N}}$ exhibits negative correlation with the test accuracy of architectures in NAS-Bench201 (Dong & Yang, 2020).

Figure 1 demonstrates that the $\kappa_{\mathcal{N}}$ is negatively correlated with the architecture's test accuracy, with the Kendall-tau correlation as -0.42. Therefore, minimizing the $\kappa_{\mathcal{N}}$ during the search will encourage the discovery of architectures with high performance.

3.1.2 Expressivity by Number of Linear Regions

The expressivity of a neural network indicates how complex the function it can represent (Hornik et al., 1989; Giryes et al., 2016). For ReLU networks, each ReLU function defines a linear boundary and divides its input space into two regions. Since the composition of piecewise linear functions is still piecewise linear, every ReLU network can be seen as a piecewise linear function. The input space of a ReLU network can be partitioned into distinct pieces (i.e. linear regions) (Figure 2), each of which is associated with a set of affine parameters, and the function represented by the network is affine when restricted to each piece. Therefore, it is natural to measure the expressivity of a ReLU network with the number of linear regions it can separate.

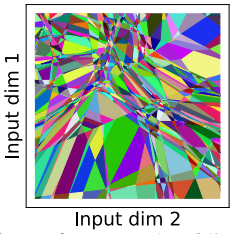


Figure 2: Example of linear regions divided by a ReLU network[1]

Following Hanin & Rolnick (2019a;b); Xiong et al. (2020), we first introduce the following definition of activation patterns and linear regions for ReLU CNNs.

Definition 1 of Xiong et al. (2020) (Activation Patterns and Linear Regions) *Let $\mathcal{N}$ be a ReLU CNN. An activation pattern of $\mathcal{N}$ is a function $\boldsymbol{P}$ from the set of neurons to $\{1, -1\}$, i.e., for each neuron z in $\mathcal{N}$, we have $\boldsymbol{P}(z) \in \{1, -1\}$. Let θ be a fixed set of parameters (weights and biases) in $\mathcal{N}$, and $\boldsymbol{P}$ be an activation pattern. The region corresponding to $\boldsymbol{P}$ and θ is*

$$\boldsymbol{R}(\boldsymbol{P}; \theta) := \{\boldsymbol{x}^0 \in \mathbb{R}^{C \times H \times W} : z(\boldsymbol{x}^0; \theta) \cdot \boldsymbol{P}(z) > 0, \quad \forall z \in \mathcal{N}\}, \tag{4}$$

[1]Plot is generated by us with the same method described by Hanin & Rolnick (2019a).

where $z(\boldsymbol{x}^0;\theta)$ is the pre-activation of a neuron z. Let $R_{\mathcal{N},\theta}$ denote the number of linear regions of $\mathcal{N}$ at θ, i.e., $R_{\mathcal{N},\theta} := \#\{\boldsymbol{R}(\boldsymbol{P};\theta) : \boldsymbol{R}(\boldsymbol{P};\theta) \neq \emptyset$ for some activation pattern $\boldsymbol{P}\}$.

Eq. 4 tells us that a linear region in the input space is a set of input data $\boldsymbol{x}^0$ that satisfies a certain fixed activation pattern $\boldsymbol{P}(z)$, and therefore the number of linear regions $R_{\mathcal{N},\theta}$ measures how many unique activation patterns that can be divided by the network.

In our work, we draw network parameters from the standard gaussian distribution $\theta \sim N(0,1)$, repeat the measurement of the number of linear regions, and calculate the average as the approximation to its expectation:

$$\hat{R}_{\mathcal{N}} \simeq \mathbb{E}_{\theta\sim N(0,1)} R_{\mathcal{N},\theta} \tag{5}$$

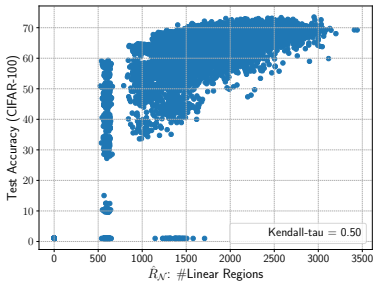


Figure 3: Number of linear regions $\hat{R}_{\mathcal{N}}$ of architectures in NAS-Bench201 exhibits positive correlation with test accuracies.

We iterate through all architectures in NAS-Bench-201 (Dong & Yang, 2020), and calculate their numbers of linear regions. Figure 3 demonstrates that the number of linear regions is positively correlated with the architecture's test accuracy, with the Kendall-tau correlation as 0.5. Therefore, maximizing the number of linear regions during the search will also encourage the discovery of architectures with high performance.

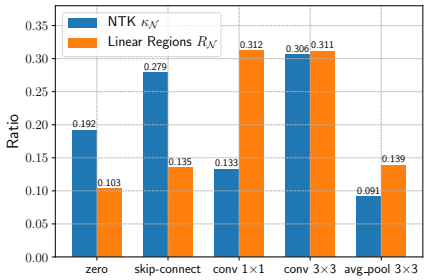


Figure 4: $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ prefer different operators in NAS-Bench201.

Finally, in Figure 4 we analyze the operator composition of top 10% architecture by maximizing $R_{\mathcal{N}}$ and minimizing $\kappa_{\mathcal{N}}$, respectively. We can clearly see that $R_{\mathcal{N}}$ and $\kappa_{\mathcal{N}}$ have different preferences to choosing operators. They both choose a large ratio of conv3 $\times$ 3 for high generalization performance. But meanwhile, $R_{\mathcal{N}}$ heavily selects conv1 $\times$ 1, and $\kappa_{\mathcal{N}}$ leads to skip-connect, favoring the gradient flow.

3.2 Pruning-by-importance Architecture Search

Given the strong correlation between the architecture's test accuracy and its $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$, how to build an efficient NAS framework on top of them? We motivate this section by addressing two questions:

1. How to combine $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ together, with good explicit trade-off?

We first need to turn the two measurements $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ into one combined function, based on which we can rank architectures. As seen in Figure 1 and 3, the magnitudes of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ differ much. To avoid one overwhelming the other numerically, one possible remedy is normalization; but we cannot pre-know the ranges nor the value distributions of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$, before computing them over a search space. In order to make our combined function *well defined before search* and *agnostic to the search space*, instead of using the numerical values of $\kappa_{\mathcal{N}}/R_{\mathcal{N}}$, we could refer to their relative rankings. Specifically, each time by comparing the sampled set of architectures peer-to-peer, we can directly sum up the two relative rankings of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ as the selection criterion. The equal-weight summation treats trainability and expressivity with the same importance conceptually[1] and delivers the best empirical result: we thus choose it as our default combined function. We also tried some other means to combine the two, and the ablation studies can be found in Appendix D.2.

2. How to search more efficiently?

Sampling-based methods like reinforcement learning or evolution can use rankings as the reward or filtering metric. However, they are inefficient, especially for complex cell-based search space. Consider an network stacked by repeated cells (directed acyclic graphs) (Zoph et al., 2018; Liu et al., 2018b). Each cell has E edges, and on each edge we only select one operator out of $|\mathcal{O}|$ ($\mathcal{O}$ is the set of operator candidates). There are $|\mathcal{O}|^E$ unique cells, and for sampling-based methods, $\gamma \cdot |\mathcal{O}|^E$ networks have to be sampled during the search. The ratio γ can be interpreted as the sampling efficiency: a method with small γ can find good architectures faster. However, the search time cost of sampling-based methods still scales up with the size of the search space, i.e., $|\mathcal{O}|^E$.

[1] We tried some weighted summations of the two, and find their equal-weight summation to perform the best.

Inspired by recent works on pruning-from-scratch (Lee et al., 2018; Wang et al., 2020), we propose a pruning-by-importance NAS mechanism to quickly shrink the search possibilities and boost the efficiency further, reducing the cost from $|\mathcal{O}|^E$ to $|\mathcal{O}| \cdot E$. Specifically, we start the search with a super-network $\mathcal{N}_0$ composed of all possible operators and edges. In the outer loop, for every round we prune one operator on each edge. The outer-loop stops when the current supernet $\mathcal{N}_t$ is a single-path network[2], i.e. the algorithm will return us the final searched architecture. For the inner-loop, we measure the change of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ before and after pruning each individual operator, and assess its importance using the sum of two ranks. We order all currently available operators in terms of their importance, and prune the lowest-importance operator on each edge.

The whole pruning process is extremely fast. As we will demonstrate later, our approach is principled and can be applied to different spaces without making any modifications. This pruning-by-importance mechanism may also be extended to indicators beyond $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$. We summarize our training-free and pruning-based NAS framework, TE-NAS, in Algorithm 1.

Algorithm 1: TE-NAS: Training-free Pruning-based NAS via Ranking of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$.

Input: supernet $\mathcal{N}_0$ stacked by cells, each cell has E edges, each edge has $|\mathcal{O}|$ operators, step $t = 0$.
while $\mathcal{N}_t$ *is not a single-path network* **do**
  for *each operator* o_j *in* $\mathcal{N}_t$ **do**
    $\Delta\kappa_{t,o_j} = \kappa_{\mathcal{N}_t} - \kappa_{\mathcal{N}_t \backslash o_j}$ ▷ the higher $\Delta\kappa_{t,o_j}$ the more likely we will prune o_i
    $\Delta R_{t,o_j} = R_{\mathcal{N}_t} - R_{\mathcal{N}_t \backslash o_j}$ ▷ the lower $\Delta R_{t,o_j}$ the more likely we will prune o_j
  Get importance by $\kappa_{\mathcal{N}}$: $s_\kappa(o_j) =$ index of o_i in descendingly sorted list $[\Delta\kappa_{t,o_1}, ..., \Delta\kappa_{t,o_{|\mathcal{N}_t|}}]$
  Get importance by $R_{\mathcal{N}}$: $s_R(o_j) =$ index of o_j in ascendingly sorted list $[\Delta R_{t,o_1}, ..., \Delta R_{t,o_{|\mathcal{N}_t|}}]$
  Get importance $s(o_j) = s_\kappa(o_j) + s_R(o_j)$
  $\mathcal{N}_{t+1} = \mathcal{N}_t$
  for *each edge* e_i, $i = 1, ..., E$ **do**
    $j^* = \arg\min_j \{s(o_j) : o_j \in e_i\}$ ▷ find the operator with least importance on each edge.
    $\mathcal{N}_{t+1} = \mathcal{N}_{t+1} \backslash o_{j^*}$
  $t = t + 1$
return *Pruned single-path network* $\mathcal{N}_t$.

3.2.1 Visualization of Search Process

TE-NAS benefits us towards a better understanding of the search process. We can analyze the trajectory of $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ during the search. It is worth noting that our starting point $\mathcal{N}_0$, the un-pruned supernet, is assumed to be of the highest expressivity (as it is composed of all operators in the search space and has the largest number of parameters and ReLU functions). However, it has poor trainability, as people find many engineering techniques are required to effectively training the supernet (Yu et al., 2020a;b). Therefore, during pruning we are expecting to strengthen the trainability of the supernet, while retaining its expressivity as much as possible.

As we observe in Figure 5, the supernet $\mathcal{N}$ is first pruned by quickly reducing $\kappa_{\mathcal{N}}$, i.e., increasing the network's trainability. After that, as the improvement of $\kappa_{\mathcal{N}}$ is almost plateaued, the method carefully fine-tunes the architecture without sacrificing too much expressivity $R_{\mathcal{N}}$.

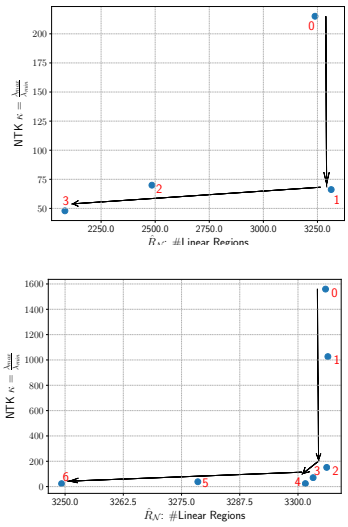


Figure 5: Pruning trajectory on NAS-Bench-201 (top) and DARTs search space (bottom). Number "0" indicates the supernet $\mathcal{N}_0$ before any pruning, which is of high expressivity but poor trainability.

[2]Different search space may have different criteria for single-path network. In NAS-Bench201 (Dong & Yang, 2020) each edge only keeps one operator at the end of the search, while in DARTS space (Liu et al., 2018b) there are two operators on each edge in the searched network.

4 EXPERIMENTS

In this section, we evaluate our TE-NAS on two search spaces: NAS-Bench-201 (Dong & Yang, 2020) and DARTS (Liu et al., 2018b). Search and training protocols are summarized in Appendix A. All our codes and pre-trained models will be released upon acceptance.

4.1 RESULTS ON NAS-BENCH-201

NAS-Bench-201 (Dong & Yang, 2020) provides a standard cell-based search space (containing 15,625 architectures) and a database of architecture's performance evaluated under a unified protocol. The network's test accuracy can be directly obtained by querying the database, which facilitates people to focus on studying NAS algorithms without network evaluation. NAS-Bench-201 supports three datasets (CIFAR-10, CIFAR-100, ImageNet-16-120 (Chrabaszcz et al., 2017)). The operation space contains *none* (*zero*), *skip connection*, *conv*1×1, *conv*3×3 *convolution*, and *average pooling* 3×3. We refer to their paper for details of the space. Our search is dataset-specific, i.e. the search and evaluation are conducted on the same dataset.

Table 1: Comparison with state-of-the-art NAS methods on NAS-Bench-201. Test accuracy with mean and deviation are reported. "optimal" indicates the best test accuracy achievable in NAS-Bench-201 search space.

Architecture	**CIFAR-10**	**CIFAR-100**	**ImageNet-16-120**	**Search Cost (GPU sec.)**	**Search Method**
ResNet (He et al., 2016)	93.97	70.86	43.63	-	-
RSPS (Li & Talwalkar, 2020)	$87.66(1.69)$	$58.33(4.34)$	$31.14(3.88)$	8007.13	random
ENAS (Pham et al., 2018)	$54.30(0.00)$	$15.61(0.00)$	$16.32(0.00)$	13314.51	RL
DARTS (1st) (Liu et al., 2018b)	$54.30(0.00)$	$15.61(0.00)$	$16.32(0.00)$	10889.87	gradient
DARTS (2nd) (Liu et al., 2018b)	$54.30(0.00)$	$15.61(0.00)$	$16.32(0.00)$	29901.67	gradient
GDAS (Dong & Yang, 2019)	$93.61(0.09)$	$70.70(0.30)$	$41.84(0.90)$	28925.91	gradient
NAS w.o. Training (Mellor et al., 2020)	$91.78(1.45)$	$67.05(2.89)$	$37.07(6.39)$	4.8	training-free
TE-NAS (ours)	$\mathbf{93.9(0.47)}$	$\mathbf{71.24(0.56)}$	$\mathbf{42.38(0.46)}$	1558	training-free
Optimal	94.37	73.51	47.31	-	-

We run TE-NAS for four independent times with different random seeds, and report the mean and standard deviation in Table 1. We can see that TE-NAS achieves the best accuracy on all three datasets, and largely reduces the search cost ($5\times \sim 19\times$ reduction). Although Mellor et al. (2020) requires even less search time (by only sampling 25 architectures), they suffer from much inferior accuracy performance, with notably larger deviations across different search rounds.

4.2 RESULTS ON CIFAR-10 WITH DARTS SEARCH SPACE

Architecture Space The DARTs operation space $\mathcal{O}$ contains eight choices: *none* (*zero*), *skip connection*, *separable convolution* 3×3 and 5×5, *dilated separable convolution* 3×3 and 5×5, *max pooling* 3×3, *average pooling* 3×3. Following previous works (Liu et al., 2018b; Chen et al.,

Table 2: Comparison with state-of-the-art NAS methods on CIFAR-10.

Architecture	**Test Error (%)**	**Params (M)**	**Search Cost (GPU days)**	**Search Method**
AmoebaNet-A (Real et al., 2019)	$3.34(0.06)$	3.2	3150	evolution
PNAS (Liu et al., 2018a)*	$3.41(0.09)$	3.2	225	SMBO
ENAS (Pham et al., 2018)	2.89	4.6	0.5	RL
NASNet-A (Zoph et al., 2018)	2.65	3.3	2000	RL
DARTS (1st) (Liu et al., 2018b)	$3.00(0.14)$	3.3	0.4	gradient
DARTS (2nd) (Liu et al., 2018b)	$2.76(0.09)$	3.3	1.0	gradient
SNAS (Xie et al., 2018)	$2.85(0.02)$	2.8	1.5	gradient
GDAS (Dong & Yang, 2019)	2.82	2.5	0.17	gradient
BayesNAS (Zhou et al., 2019)	$2.81(0.04)$	3.4	0.2	gradient
ProxylessNAS (Cai et al., 2018)†	2.08	5.7	4.0	gradient
P-DARTS (Chen et al., 2019)	2.50	3.4	0.3	gradient
PC-DARTS (Xu et al., 2019)	$2.57(0.07)$	3.6	0.1	gradient
SDARTS-ADV (Chen & Hsieh, 2020)	$2.61(0.02)$	3.3	1.3	gradient
TE-NAS (ours)	$2.63(0.064)$	3.8	$0.02^\ddagger$	training-free

* No cutout augmentation.
† Different space: PyramidNet (Han et al., 2017) as the backbone.
‡ Recorded on a single GTX 1080Ti GPU.

2019; Xu et al., 2019), for both search and evaluation phases, we stack 20 cells to compose the network and set the initial channel number as 36. We place the reduction cells at the 1/3 and 2/3 of the network and each cell consists of six nodes.

Results We run TE-NAS for four independent times with different random seeds, and report the mean and standard deviation. Table 2 summarizes the performance of TE-NAS compared with other popular NAS methods. TE-NAS achieves a test error of 2.63%, ranking among the top of recent NAS results, but meanwhile largely reduces the search cost to only 30 minutes. ProxylessNAS achieves lowest test error, but it searches on a different space with a much longer search time and has a larger model size. Besides, Mellor et al. (2020) did not extend to their Jacobian-based framework to DARTs search space for CIFAR-10 or ImageNet classification.

4.3 Results on ImageNet with DARTs Search Space

Architecture Space Following previous works (Xu et al., 2019; Chen et al., 2019), the architecture for ImageNet is slightly different from that for CIFAR-10. The network is stacked with 14 cells with the initial channel number set to 48. During retraining evaluation, we follow the mobile setting to control the FLOPs not exceed 600 MB by adjusting the channel number. The spatial resolution is downscaled from 224×224 to 28×28 with the first three convolution layers of stride 2.

Results As shown in Table 3, we achieve a top-1/5 test error of 26.0%/8.3%, achieving competitive performance with recent state-of-the-art works in the ImageNet mobile setting. However, Te-NAS only cost four GPU hours with only one 1080Ti. Searching on ImageNet takes longer time than on CIFAR-10 due to the larger input size and more network parameters.

Table 3: Comparison with state-of-the-art NAS methods on ImageNet under the mobile setting.

Architecture	**Test Error(%)**		**Params**	**Search Cost**	**Search**
	top-1	top-5	**(M)**	**(GPU days)**	**Method**
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	2000	RL
AmoebaNet-C (Real et al., 2019)	24.3	7.6	6.4	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	225	SMBO
MnasNet-92 (Tan et al., 2019)	25.2	8.0	4.4	-	RL
DARTS (2nd) (Liu et al., 2018b)	26.7	8.7	4.7	4.0	gradient
SNAS (mild) (Xie et al., 2018)	27.3	9.2	4.3	1.5	gradient
GDAS (Dong & Yang, 2019)	26.0	8.5	5.3	0.21	gradient
BayesNAS (Zhou et al., 2019)	26.5	8.9	3.9	0.2	gradient
ProxylessNAS (GPU) (Cai et al., 2018)†	24.9	7.5	7.1	8.3	gradient
P-DARTS (CIFAR-10) (Chen et al., 2019)	24.4	7.4	4.9	0.3	gradient
P-DARTS (CIFAR-100) (Chen et al., 2019)	24.7	7.5	5.1	0.3	gradient
PC-DARTS (CIFAR-10) (Xu et al., 2019)	25.1	7.8	5.3	0.1	gradient
PC-DARTS (ImageNet) (Xu et al., 2019)†	24.2	7.3	5.3	3.8	gradient
TE-NAS (ours)	26.2	8.3	6.3	0.02	training-free
TE-NAS (ours)†	25.9	8.3	6.5	0.17	training-free

† The architecture is searched on ImageNet, otherwise it is searched on CIFAR-10 or CIFAR-100.

5 Conclusion

The key questions in Neural Architecture Search (NAS) are "what are good architectures" and "how to find them". Validation loss or accuracy are possible answers but not enough, due to their search bias and heavy evaluation cost. Our work demonstrates that two theoretically inspired indicators, the spectrum of NTK and number of linear regions, no only strongly correlate with the network's performance, but also benefit the reduced search cost and decoupled analysis of the network's trainability and expressivity. Without involving any training, our TE-NAS achieve competitive NAS performance with minimum search time. We for the first time bridge the gap between the theoretic findings of deep neural networks and the real-world NAS applications, and we encourage the community to further explore more meaningful network properties so that we will have a better understanding of good architectures and how to search them.

REFERENCES

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pp. 549–558, 2018.

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

Rebekka Burkholz and Alina Dubatovka. Initialization of relus for dynamical isometry. In *Advances in Neural Information Processing Systems*, pp. 2385–2395, 2019.

Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pp. 8699–8710, 2018.

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. *arXiv preprint arXiv:2002.05283*, 2020.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1294–1303, 2019.

Xin Chen, Lingxi Xie, Jun Wu, Longhui Wei, Yuhui Xu, and Qi Tian. Fitting the search space of weight-sharing nas with graph convolutional networks. *arXiv preprint arXiv:2004.08423*, 2020.

Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. 2019.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets, 2017.

Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pp. 11398–11407, 2019.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on computer vision and pattern recognition*, pp. 1761–1770, 2019.

Xuanyi Dong and Yi Yang. Nas-bench-102: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. Nats-bench: Benchmarking nas algorithms for architecture topology and size. *arXiv preprint arXiv:2009.00437*, 2020.

Raja Giryes, Guillermo Sapiro, and Alex M Bronstein. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 64(13): 3444–3457, 2016.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv preprint arXiv:1904.00420*, 2019.

Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5927–5935, 2017.

Boris Hanin and Mihai Nica. Finite depth and width corrections to the neural tangent kernel. *arXiv preprint arXiv:1909.05989*, 2019.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. *arXiv preprint arXiv:1901.09021*, 2019a.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In *Advances in Neural Information Processing Systems*, pp. 361–370, 2019b.

Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. *arXiv preprint arXiv:1902.06853*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*. 2018a.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018b.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, pp. 8572–8583, 2019.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 367–377. PMLR, 2020.

Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035*, 2019.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 19–34, 2018a.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 82–92, 2019.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018b.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.

Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *arXiv preprint arXiv:2002.10389*, 2020.

Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. *arXiv preprint arXiv:2006.04647*, 2020.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

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, pp. 4780–4789, 2019.

Yeonjong Shin and George Em Karniadakis. Trainability of relu networks and data-dependent initialization. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020.

Yao Shu, Wei Wang, and Shaofeng Cai. Understanding architectures learnt by cell-based neural architecture search. In *International Conference on Learning Representations*, 2019.

Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10781–10790, 2020.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

Wei Wen, Hanxiao Liu, Hai Li, Yiran Chen, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. *arXiv preprint arXiv:1912.00848*, 2019.

Lechao Xiao, Jeffrey Pennington, and Samuel S Schoenholz. Disentangling trainability and generalization in deep learning. *arXiv preprint arXiv:1912.13053*, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

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

Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. On the number of linear regions of convolutional neural networks. *arXiv preprint arXiv:2006.00978*, 2020.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2019.

Ge Yang and Samuel Schoenholz. Mean field residual networks: On the edge of chaos. In *Advances in neural information processing systems*, pp. 7103–7114, 2017.

Greg Yang. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760*, 2019.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. *arXiv preprint arXiv:2003.11142*, 2020a.

Kaicheng Yu, Rene Ranftl, and Mathieu Salzmann. How to train your super-net: An analysis of training heuristics in weight-sharing nas. *arXiv preprint arXiv:2003.04276*, 2020b.

Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. In *ICLR*, 2020c.

Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. *arXiv preprint arXiv:1905.04919*, 2019.

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

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

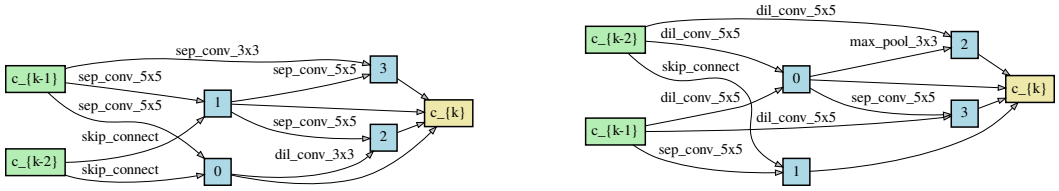
A IMPLEMENTATION DETAILS

For $\kappa_{\mathcal{N}}$ we sample one mini-batch of size 32 from the training set, and calculate $\hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}') = J(\boldsymbol{x})J(\boldsymbol{x}')^T$. For $R_{\mathcal{N}}$ we sample 5000 images, forward them through the network, and collect the activation patterns from all ReLU layers. The calculation of both $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ are repeated for three times in all experiments, where each time the network weights are randomly drawn from standard Guassian $N(0, 1)$ without involving any training (network weights are fixed).

Our retraining settings (after search) follow previous works (Xu et al., 2019; Chen et al., 2019; Chen & Hsieh, 2020). On CIFAR-10, we train the searched network with cutout regularization of length 16, drop-path (Zoph et al., 2018) with probability as 0.3, and an auxiliary tower of weight 0.4. On ImageNet, we also use label smoothing during training. On both CIFAR-10 and ImageNet, the network is optimized by an SGD optimizer with cosine annealing, with learning rate initialized as 0.025 and 0.5, respectively.

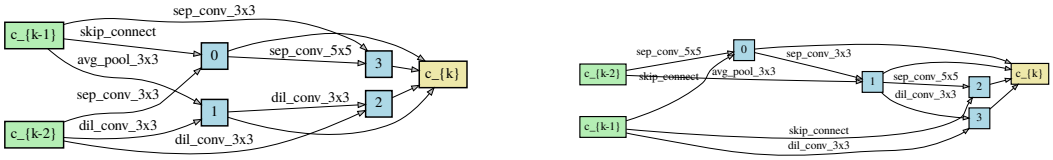
B SEARCHED ARCHITECTURE

We visualize the searched normal and reduction cells in figure 6 and 7, which is directly searched on CIFAR-10 and ImageNet respectively.



(a) Normal Cell (b) Reduction Cell

Figure 6: Normal and Reduction cells discovered by TE-NAS on CIFAR-10.

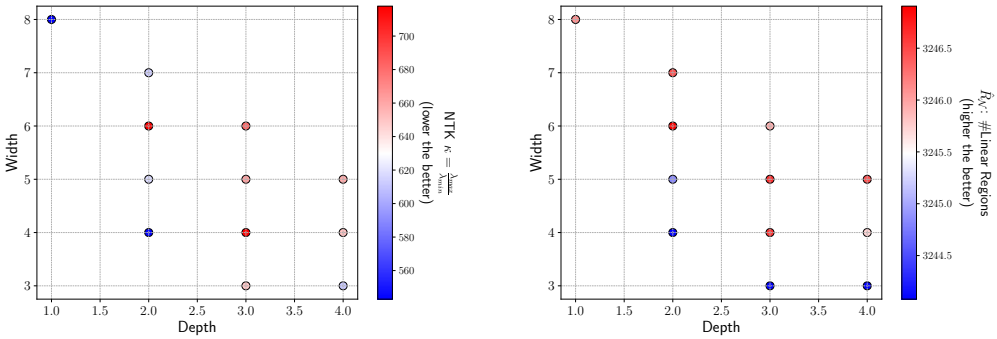


(a) Normal Cell (b) Reduction Cell

Figure 7: Normal and Reduction cells discovered by TE-NAS on imageNet.

C DEPTH AND WIDTH PREFERENCE OF $\kappa_{\mathcal{N}}$ AND $R_{\mathcal{N}}$ IN DARTS SPACE

To analysis the impact of different architectures on trainability and expressivity in DARTs search space, we visualize $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ with different depths and width. Following Shu et al. (2019), the depth of a cell is defined as the number of connections on the longest path from input nodes to the output node, and the width of a cell is the summation of the edges of the intermediate nodes that are connected to the input nodes. We randomly sample 20,000 architectures in DARTs space, and plot the visualizations in Figure 8. Good architectures should exhibit low $\kappa_{\mathcal{N}}$ (good trainability, blue dots in Figure 8(a)) and high $R_{\mathcal{N}}$ (powerful expressivity, red dots in Figure 8(b)). Therefore, $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ tell us that in DARTs space shallow but wide cells are preferred to favor both trainability and expressivity. This conclusion matches the findings by Shu et al. (2019): existing NAS algorithms tend to favor architectures with wide and shallow cell structures, which enjoy fast convergence with smooth loss landscape and accurate gradient information.



(a) $\kappa_{\mathcal{N}}$ (b) $R_{\mathcal{N}}$

Figure 8: Depth and width preference of (a) $\kappa_{\mathcal{N}}$ and (b) $R_{\mathcal{N}}$ on DARTs Search Space.

D More Ablation Studies

D.1 Search with only $\kappa_{\mathcal{N}}$ or $R_{\mathcal{N}}$

As we observed in Table 4, searching with only $\kappa_{\mathcal{N}}$ or $R_{\mathcal{N}}$ leads to inferior performance, which indicates the importance of maintaining both trainability and expressivity during the search.

Table 4: Search with only $\kappa_{\mathcal{N}}$ or $R_{\mathcal{N}}$ on CIFAR-100 in NAS-Bench-201 space.

Methods	CIFAR-100 Test Accuracy
Prune with only $\kappa_{\mathcal{N}}$	69.25 (1.29)
Prune with only $R_{\mathcal{N}}$	70.48 (0.29)
TE-NAS	**71.24** (0.56)

D.2 Different combination options for $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$

Pruning by $s(o_j) = s_\kappa(o_j) + s_R(o_j)$ is not the only option (see Algorithm 1). Here in this study we consider more:

1) pruning by $s(o_j) = \min(s_\kappa(o_j), s_R(o_j))$, i.e., pruning by the worst case.
2) Pruninng by $s(o_j) = \max(s_\kappa(o_j), s_R(o_j))$, i.e., pruning by the best case.
3) pruning by summation of changes $\Delta\kappa_{t,o_j} + \Delta R_{t,o_j}$, i.e., directly use the numerical values of the changes.

As we observed in Table 5, our TE-NAS stands out of all options. This means a good trade-off between $\kappa_{\mathcal{N}}$ and $R_{\mathcal{N}}$ are important, and also the ranking strategy is better than directly using numerical values.

Table 5: Search with only $\kappa_{\mathcal{N}}$ or $R_{\mathcal{N}}$ on CIFAR-100 in NAS-Bench-201 space.

Methods	CIFAR-100 Test Accuracy
Prune by $s(o_j) = \min(s_\kappa(o_j), s_R(o_j))$	70.75 (0.73)
Prune by $s(o_j) = \max(s_\kappa(o_j), s_R(o_j))$	70.33 (1.09)
Prune by $\Delta\kappa_{t,o_j} + \Delta R_{t,o_j}$	70.47 (0.68)
Prune by $s(o_j) = s_\kappa(o_j) + s_R(o_j)$ (TE-NAS)	**71.24** (0.56)