https://cke.um.ac.ir/ *DOI: 10.22067/cke.2022.73356.1031*

Towards Leveraging Structure for Neural Predictor in NAS*

Research Article

Saeedeh Eslami[1] Reza Monsefi[2] Mohammad Akbari[3]

Abstract: Neural Architecture Search (NAS), which automatically designs a neural architecture for a specific task, has attracted much attention in recent years. Properly defining the search space is a key step in the success of NAS approaches, which allows us to reduce the required time for evaluation. Thus, late strategies for searching a NAS space is to leverage supervised learning models for ranking the potential neural models, i.e., surrogate predictive models. The predictive model takes the specification of an architecture (or its feature representation) and predicts the probable efficiency of the model ahead of training. Therefore, proper representation of a candidate architecture is an important factor for a predictor NAS approach. While several works have been devoted to training a good surrogate model, there exits limited research focusing on learning a good representation for these neural models. To address this problem, we investigate how to learn a representation with both structural and non-structural features of a network. In particular, we propose a tree structured encoding which permits to fully represent both networks' layers and their intra-connections. The encoding is easily extendable to larger or more complex structures. Extensive experiments on two NAS datasets, NasBench101 and NasBench201, demonstrate the effectiveness of the proposed method as compared with the state-of-the-art predictors.

Keywords: Neural Architecture Search (NAS), Search Space Pruning, Network Architecture, Representation Learning

1. Introduction

The success of various machine learning tasks heavily depends on effective design of a proper neural model. Extensive studies demonstrate that relying on expert experience often results in subjective sub-optimal solutions, needs huge time, and resource consumption. Generally, designing an optimal network architecture is a crucial and arduous step for every machine learning problem. Traditionally, an expert proposes a neural model for the given task and then the model is trained with various hyper-parameters to achieve the best design.

The process is repetitive, timely and error-prone. As such automating network design process, i.e., specifying network parameters, also known as Neural Architecture Search (NAS), has become an emerging topic in automatic machine learning (autoML). While NAS framework focuses on automatic ways to select hyper-parameters and design appropriate network architectures, designing an effective framework for NAS is still challenging and under-researched issue problem. The search space of NAS task is formed by all feasible combinations of various components of a network; an enormous space which can exhaust any search algorithm. Thus, a well-defined search space is essential to compensate the shortcomings of average search strategies like random sampling methods. To do so, retrospective studies in NAS have proposed various strategies for pruning the search space. For example, Reinforcement Learning (RL) and Evolutionary Algorithms seek the optimal solution in a discrete space [1, 2]. One-shot learning and stochastic neural search model the problem in a continuous space [3, 4]. However, these approaches often need a considerable amount of time (hours or days) merely to find the exact performance of candidate models and hence fully training and evaluating thousands of architectures. As a result, selecting the optimal architectures with the best performance on validation data is a matter of thousands of GPU days and the institute which adopts this approach should be able to afford such equipment, otherwise the search is actually impractical.

Recent studies in this regard show that a very efficient strategy for making the NAS feasible, is training a performance predicting model with the most influential network features to predict the proposed model's performance and thus keep the K −top promising modes for actual training [5, 6]. The predictor takes a network representation as input, i.e., input features, and estimates its final performance according to the previously seen models. The hypothesis behind this is that the predictor is trained with a few pair of fully trained networks and their performances. If the selected models are of high quality, training with this small set of models will help the predictor to reliably select a good model. While predictor's estimation makes the main algorithm needless of a complete training chore to assess other proposed models and thus saves time, training a meta-predictor is a nontrivial task due to following challenges. First, every network architecture is defined via two factors: the layer specifications and the connections between layers. Although critically important, the prior works have merely focused on search space and search strategy and have not considered a complete structure encoding for the networks presented to the predictor, i.e., their encoding whether ignores the layer specifications or the interconnections (misses some important aspects of a network) [1, 7]. But as we can see in another branch of NAS studies, named curve extrapolation, adding some structural information to the training reports [8, 9] improved the performance estimation and convergence time drastically. Since the introduction of additive (residual) networks like ResNet, DenseNet, etc., the connections between layers has become very complicated.

*Manuscript received: 02 November 2021; Revised, 2 February 2022, Accepted: 05 March 2022.

[1] PhD student, Department of Computer Engineering, Engineering Faculty of Ferdowsi University, Mashhad, Iran..

[2] Corresponding Author, Professor, Department of Computer Engineering, Engineering Faculty of Ferdowsi University, Mashhad, Iran. **Email**: Monsefi@um.ac.ir

[3] Assistant Professor, Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran.

The skip connections can make a network actually an ensemble structure as each way from the input layer to the output layer is a complete convolutional network (Figure 1). Simple encodings that were used in early works [10, 7] are totally inefficient to represent the possible options in each single element of the network. Due to their inflexibility, their rules of connectivity are also very strict and limiting. Second, a convolutional network is a discrete structure so a tabular (discrete) representation seems more suitable than continuous representations. Previous works mostly have employed a neural based predictor (RNN, CNN, and GCN) and continuous representations to build and train the predictor. Some discussions criticize using them saying neural networks are more designed to receive image/speech data and not much to handle tabular or discrete data which seems more compatible with tree structured models [11, 22]. Also, the training data insufficiency, which is very common in NAS, may affect the neural based predictors more than tree based ones. The predictor [11] that introduced as GBDT-NAS is a GBDT based regression model using a tabular, layer by layer encoding and gained the best performance till 2020 and best top 10 results on ImageNet and CIFAR10. However, we argue that their encoding has very limited power and cannot be extended easily. Encoding new generation additive networks is also a challenge for them. In this paper, we introduce a new tree structured representation for deep convolutional networks. The representation is a complete encoding of network structure including layer's full specification and the connection patterns inside the network. Each path is encoded independent of others, thus extending, reducing or any optimization of a path does not affect others. We trained the GBDT predictor with our encoding and tested it on ImageNet16 and CIFAR10.

The training and validation data are provided by NASBench101 [12] and [13]. We investigate the effect of data insufficiency on the predictor's performance. We beat the path-based encoding proposed by BANANAS and together with NAS-GBDT reached the best results on baseline datasets. The main contribution of this paper are as follows:

- We propose an approach to embed the structure of a neural model into the network encoding;
- We demonstrate how to learn a unified representation from both the structure information and layers' attributes of a neural model;
- We evaluate the proposed model in two widely used NAS datasets and show that a good representation is crucial for NAS predictor.

The formal definition of the problem is presented in section 2 for more clarification. Network representation is discussed in section 3. We also discuss different aspects of representation and our method of representation in its subsections A. and B. Then, we talk about the neural predictor in section 4. Finally, the results and the conclusion are presented in section 5 and section 6, respectively.

2. Statement of the problem

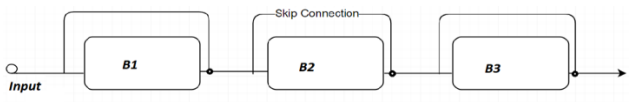
The aim is predicting the performance of a neural architecture before training. This can be modelled as a regression problem in machine learning where we aim at learning a regressor F to take the representation of an architecture $N_i \epsilon N$ and return its estimated performance as $\hat{y}_i = F(W_F, N_i)$, where W_f denotes the trainable function parameters. The function F is trained to minimize the following error:

$$\min_{W_f} ||\mathcal{F}(W_f, X_i) - y_i||_2^2 \quad (1)$$

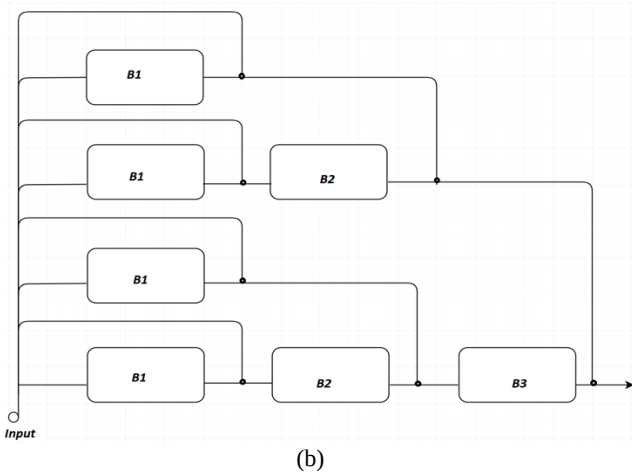
where y_i denotes the actual performance of the network as label data. The key to success of the predictor is to learn a representative encoding for the network N_i. In next section, representation issues are discussed and our method is presented to fully represent a deep convolutional network.

3. Network representation

Any deep neural architecture can be defined via a set of layers, their specifications (i.e., type of filters in each layer, order of filters and their set of hyperparameters, type of activation functions, etc.), and the interconnections among them. The CNN is a series of layers which receive the inputs from previous layers, transform them via some operations and pass them to the next layers. Thus, designing is a series of successive and dependent decisions about the layers' specifications and their interconnections. To leverage predictors for NAS, we need to extract discriminative features from both structural features of the network and non-structural attributes of its layers. Suppose we have a network N with L successive layers denoted by $(l_1, l_2 \ldots, l_L)$, where l_1 and l_L are the input and the output layers of the network, respectively. In the following sections, we explain how to build a representation for both structures of the network and attributes of its layers, respectively.



(a)



(b)

igure 1. A typical CNN model and its paths from input to output: (a). The typical representation of Residual networks, (b). The unraveled view of a 3-block residual network; adding a block doubles the number of paths

A. Modeling network structure

A major aspect of any neural architecture is the interconnections between layers which controls how features are extracted and passed into consecutive layers. In a typical convolutional neural network (CNN) layers are connected to

each other in a feed-forward manner. However, it is likely to have several paths from the input layer to the output layer, especially if skip connections are allowed [14], as shown in Figure 1. Different paths extract various type of features for the final layer. A good representation should denote all such paths so that the predictor can learn the process through the extracted features. Here, we try to model a network by a set of paths from input layer to output layer and to represent all existing paths in a tree structure. We say that the number of possible unique paths from input to output layer of a neural architecture is finite and can be pre-computed from the network architecture. Suppose N is a convolutional network with L layers denoted by $(l_1, l_2 \dots, l_L)$ and $A_{L\times L}$is its adjacency matrix and A_i is the $i-$th layer connectivity, then the total number of unique paths from the input layer to the output is

$$\begin{cases} \prod_{i=1}^{L-1} \deg(A_i) & if\ A_{1L} = 0 \\ ((\deg(A_1) - 1) \times \prod_{i=2}^{L-1} \deg(A_i)) + 1 & oth. \end{cases} \quad (2)$$

where $\deg(A_i)$ is the number of outgoing connections from the layer i. Assume an isolated tree root. It does not correspond to any layer of the original network and will only connect all the found paths as the root. So, the number of Root's first level children is the total number of paths which can be computed using the equation. These children nodes, represent the input/first network layer. The adjacency matrix is then traversed in a semi depth-first order beginning from A_1 connections and the paths are recognized one by one and attached to the previously built level of the tree. Each branch terminates by A_L as the tree leaf. There are possibly a number of repeated order of nodes but it doesn't matter.

Algorithm 1: The algorithm to form the tree of the network

Data: $Deg(A_i)$: Number of A_i's non-zero elements
Result: Full set of paths
$Visited \leftarrow zeros_{(1 \times L)}$;
$Num\_nodes \leftarrow 0$;
for $i \leftarrow 1$ **to** Num_paths **do**
  add l_1 to $tree_o$.children $j \leftarrow 2$;
  $parent \leftarrow 1$;
  $Num\_nodes \leftarrow Num\_nodes + 1$;
  while $j < L - 1$ **do**
    if l_{parent} *connects to* $l_j (i.e. A[parent, j] == 1)$ *AND* $Visited[j] < Deg(A[j])$ **then**
      Add $tree_{Num\_nodes}$ to $tree_{parent}.children$;
      $tree_{Num\_nodes} \leftarrow l_j$;
      $parent \leftarrow j$;
      $Visited \leftarrow Visited + 1$;
      $Num\_nodes \leftarrow Num\_nodes + 1$;
    end
    $j \leftarrow j + 1$;
  end
  if l_{parent} *connects to* l_L **then**
    Add $tree_{Num\_nodes}$ to $tree_{parent}.children$;
    $tree_{Num\_nodes} \leftarrow l_L$;
    $Num\_nodes \leftarrow Num\_nodes + 1$;
  end
end

We employ Depth First Search (DFS) algorithm to convert the network graph into a tree. This can be stated as follows. Suppose $\sigma(l_1, l_2 \dots, l_L)$ defines the set of paths of the tree composed of $l_1, l_2 \dots, l_L$ and rooted from l_1. If $\Delta(l_i)$ denotes the set of layers for them there is a direct connection from l_i in the network, then for all $1 \le i < j < k \le L$ if $l_i \in \Delta(l_k) \backslash \Delta(l_j)$ there exists a l_m $(i < m < j)$ such that $l_m \in \Delta(l_j)$.

Algorithm 1 shows the pseudo code of an algorithm for computing the paths and their representations.

B. Modeling layer attributes

So far, we only modelled the structure of a neural model. However, each layer can represent various operations. In this section, we explain how to model attributes of a single layer of a neural model. For the sake of simplicity, we only discuss the case that the network consists of convolution blocks; however, the extension to other neural models is intuitive. Different convolutional operations (convolution, pooling, etc.), basically convolve a kernel of arbitrary size with the input. So each operation can be represented by a numeric vector, as shown in Figure 2. The vector consists of four different values where the first one represents the type of operation via one-hot representation. The remaining elements represent all parameters associated to that operation. A convolutional operation needs kernel specification in terms of the width and height of the kernel and its number of input and output channels. We focus on square kernels and like [6] take number of output to input channels ratio to make it simple and efficient. The options for the ratio are not vast as the number of output channels is limited within a range of input channel number. The "skip-connection" is not a convolutional operation and it is necessary in residual networks to model skip connections. If the network is bound to be fully connected, we can add a semi-operation 'No-op' which means no transaction between the respective layers and thus lets us have freedom in establishing connections between layers. An operation is defined part by part by the integer values chosen from the set of valid ones. The operation set can be defined narrower or wider based on the task at hand. Some search spaces like NASBench-101 [12] define combined operations (layers) like $Conv - a \times a$ which is a $a \times a$ convolution followed by a Batchnorm and then a Relu.

See Table 1 for the full list of operations.

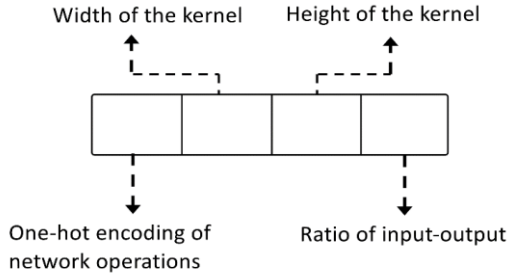


Figure 2. Vector representation of neural operations. Each operation can be fully represented by a vector of four values.

Table 1. The integer coding of various neural operations

Op. Name	Op. type	Kernel width	Kernel height	O/I channel ratio
Convolution	1	3,5	3,5	0.25,3
Max-pooling	2	3,5	3,5	1
Avg-Pooling	3	3,5	3,5	1
Relu	4	1	1	1
BatchNorm	5	1	1	1
Skip-connection	6	0	0	0
No-op	7	0	0	0

The sequence of operations should also be investigated. For example, to use 3x3 convolution at earlier levels rather than later produces a better network based on what [11] found in some tests with its method of assessment on a small set of convolutional operations. So, it's advantageous that the representation considers the order of operations as well.

Our strategy is to recognize different sequences after denoting existing paths and to keep counts of them. The smallest complete path is $'input - output'$ if there is a skip-connection between these two layers otherwise it is $'input - operation - output'$ and there would be no smaller path. The feature vector is of length D where D is the number of all possible $n-$length operation sequences. For a set of $m(m > 1)$ possible operations, the operation dataset contains m^n sequences from which $(m-1)^2 \times (m-2)^{n-2}$ are valid (patterns like operation-input-operation, output-operation-operation, etc. are considered invalid). An $m+1$ operation set has $(1 + {}^1/_m)^n$ times more operation sequences with the same length than the first set. While the feed forward network with $L + 1(L > 1)$ layers has $(L \times m)$ times more paths than a $L-$layered one. The maximum number of paths of the mentioned network is $m^{L-1} \times (L! \times m)$. We have:

$$\left(1 + {}^1/_m\right)^n \leq \left({}^3/_2\right)^n \leq \left({}^3/_2\right)^L \quad (3)$$

$$(m-1)^2 \times (m-2)^{n-2} \cong (m-2)^n \quad (4)$$

$$\forall n \leq L \left((m-1)^2 \times (m-2)^{n-2} \times \left(1 + {}^1/_m\right)^n\right) \leq m^L \times (L!) \quad (5)$$

Even at the extreme case when $n = L$, we can still favor the counter vector of $n-$length sequences to counter-vector of the full paths. When the number of layers and the pattern of connectivity is fixed in a search space, it is harmless and efficient to count full paths. The path-based feature vector which BANANAS proposed is well doing in NASBench-201 [13] where the connections are fixed and only the operations are varied but it's not applicable without truncation even in small cells when we have freedom in establishing connection among layers. To find better architectures, different number of layers and connections should be freely investigated. In these spaces, we propose that feature vectors represent $n-$length operation sequences. In section 4 our approach of constructing a performance predictor for receiving the mentioned encoding is discussed.

4. Neural predictor

Our objective is building an effective predictor for well estimating the performance of an architecture before training. This model takes a network architecture N and an epoch index t and produces a scalar value $F(N,t)$ as the prediction of the performance after exactly t epochs. Here, we incorporate both structural and non-structural information of the feature space to represent the network N. Further, inspiring from current estimation approaches [6], we consider the epoch index t another input to the model. The hypothesis behind this is that the validation accuracy generally changes as training proceeds. Therefore, when we predict performance, we have to be specific about time point of the prediction. This also helps us to better model the possible correlations between training samples. Inspired by [15], we use a three-step predictor to select promising models as follows.

4.1. Construction

To obtain a small training dataset, we train a random sample of architectures and construct training multivariates in (N_i, p_i, t_i) where N_i and p_i are in turn a network's architecture and validation accuracy at a certain training epoch plus the epoch number. Next, we use this small dataset to train a regression model F_i for predicting the accuracy of any architecture.

4.2. Ranking

The predictor model F_i is used to estimate the accuracy of a large number of random architectures. These architectures are then ranked based on their predicted accuracy and top K architectures are passed to the next step for final evaluation.

4.3. Evaluation

Here top K architectures, i.e., the promising ones, are trained and evaluated on real data to calculate their actual validation accuracy.

A. GBDT performance predictor

A single tree may not be powerful enough to capture complex relations in data. Gradient boosting decision trees (GBDT) boosts the prediction by leveraging multiple additive trees and thus different views on data:

$$\hat{y}_{GBDT}(x) = \sum_{s=1}^{S} \hat{y}_{DT_S}(x) \quad (6)$$

where S is the number of additive trees, and $\hat{y}_{DT_S}$ is the predictive model for the $s-$th tree. Each tree maps a feature vector to a weighted leaf node. The weights are adjusted according to the problem objective. Together GBDT extracts S rules to predict the target value of a given feature vector. Supposing the trees have enough diversity, each rule models different high-order feature interactions without human interference. Figure 3 illustrates our predictive model.

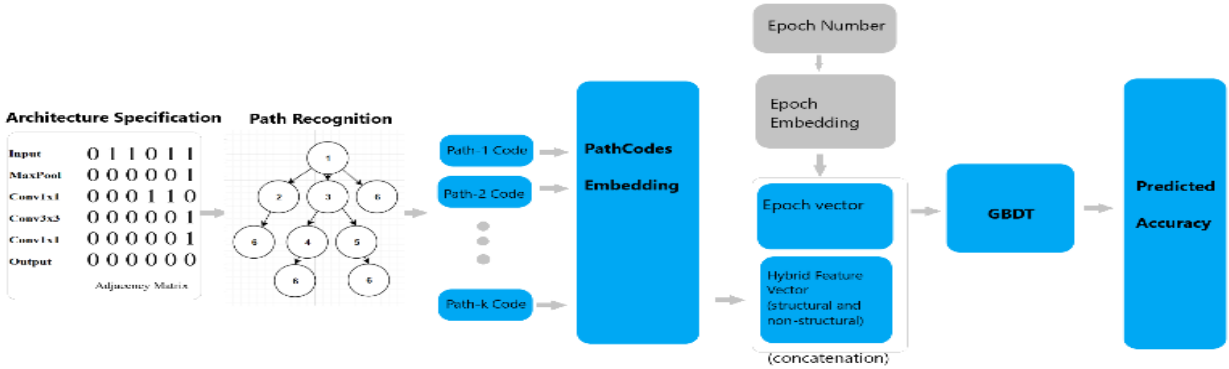


Figure 3. The schematic overview of the proposed framework

B. Sample efficiency

The predictor is trained by a M random architectures so as much as the M increases, the predictor becomes more accurate. Also, a large K makes the validation results very reliable. But there is a tradeoff between M and K as the whole training actually is done with $M + K$ models and the computation resources are limited. We have no formula to exactly determine best values and they are determined by repeated experiments.

C. Robustness to adversarial examples

Adversarial examples are the data deliberately perturbed by adversaries to mis-lead the classifiers. Network robustness is how the network resists or keeps performance against adversarial inputs. The problem is formulated as:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{x' \in S} \mathcal{L}(y, M(x'; \theta))\right] \quad (7)$$

where $S = \{x' : \|x - x'\|_p < \epsilon\}$ is the set of perturbed inputs, y is the true output, M is the model and $\mathcal{D}$ is the distribution. The model takes the architecture parameters as input and predicts its robustness [32]. To have a robust network, we need adversarial training and both accuracy and robustness should be taken into account. The objective function is generally presented as:

$$min: F(x) = \{f_1, f_2\} \quad (8)$$

Here $f_1$represents the error rate on clean data.

$$f_1 = 1 - \left(\frac{1}{n}\sum \mathbb{I}(\hat{y} == y)\right) \times 100\% \quad (9)$$

where n is the number of examples and $\mathbb{I}$ is an indicator function. While f_2 has the following general form:

$$f_2 = \frac{\left(1 - \left(\frac{1}{n}\sum \mathbb{I}(\hat{y}_a == y)\right) \times 100\%\right) - \mu}{\sigma} \quad (10)$$

the term inside parenthesis, is the error on adversarial examples generated from a random type of attack and μ and σ are the mean and standard deviation of the error rate of different training architectures for the attack [33].

The types of attacks are numerous. An architecture which resists against one type of attack may not do so against another one. This reveals the difficulty of designing a loss function for the problem. The related methods try to discover influential patterns or influential paths in robustness against certain adversarial attacks or how the parameters should be assigned to gain maximum robustness, and what is the reliable indicator of network robustness. The system should be trained with a huge amount of different architectures and various attacks to evaluate their robustness. The process of adversarial training for one type of attack is already very space and time consuming. So, to design a multi-objective loss function to consider a collection of attacks can be pursued in our future work independently.

5. Experiments

In this section, we conduct several experiments to evaluate the effectiveness of the proposed predictor. We evaluated our approach on two commonly used datasets for NAS, i.e., NASBench-101 and NASBench-201, which show superiority of our proposed approach over the state-of-the art baseline methods for NAS. While the architectures in NASBench datasets are limited in structures and types; the proposed representation paradigm can potentially encode various type of cells.

5-1. Experiments on NASBench 101

NASBench-101 search space is a dataset of more than 400‘000 pre-built and tested architectures on CIFAR-10. The search space is built upon NASnet principles [10]. The main stem of the architecture is composed of three times repetitions of a cell followed by a downsampling layer to manage the input dimensions. The architecture ends with a global average pooling layer and a dense softmax layer. Each cell is composed of up to 7 layers (two of them are reserved for input and output) and 5 valid operations. The operations include ‘CONV1x1’, ‘CONV3x3’, ‘MAXPOOL1x1’, plus ‘INPUT’ and ‘OUTPUT’ which refer to convolution with 1x1 kernel, 3x3 kernel, max pooling with 1x1 kernel, simple input and output resp. The convolution operations actually apply a series of [convolution-batchnorm-relu] operations. The cells are described with adjacency matrix and the list of applied operations. Each structure is trained, validated and tested three times and the results are averaged and reported. The dataset however contains some inconsistencies because the models with best validation errors do not necessarily report best test errors. Also, due to unstable models (like a model with only pooling operations), high variance is expected. The highest test accuracy which is reported by Regularized evolution [16] is 94.32% but running extensive search several times has led to a mean test accuracy of 94.1% and

a mean validation accuracy of 95.13% [15]. To better compare the results with [15, 11, 17], the current state of the art predictors, we validate the results on 2000 models and test them on the rest. So, next we will describe the predictor setup.

A. Experimental setting

NASBench-101 contains small cells with varied connectivity. As mentioned in section III our strategy is to recognize different sequences after denoting existing paths and to keep counts of them. We take the adjacency matrix and set of operations of a structure to extract all the paths from the input to the output layer and represent them in terms of applied operations. The order is important and different orders lead to different efficiency. Here we borrow the term n-gram to denote different n-length operation sequences. For example, a 3-gram can be [Conv1x1-MaxPool3x3-Output]. The feature vector to represent a network from the current dataset is the counter vector of all 48 introduced 3-grams. An addition to the operation set, adds a maximum of 52 more elements to the feature vector (about 2.08 times) while when a single layer is added to the once 7-layered structure, (even if assuming there is no operation set to choose from) it will make the primary 720-d path-indexing feature vector 7 times lengthier. Our encoding is independent of the number of paths and avoids this increase. For each training model, the predictor takes the vector and its accuracy. So, it knows how many of each n-gram exists in a structure and learns the goodness of such structure. We also tried other sequence lengths (bigrams and 4-grams) and observed that longer sequences do not have a very significant impact on the performance and can be omitted. A 4-length sequence can be represented by two interleaving 3-grams. We use a theorem from [17] which expresses longer sequences (paths) have a lower occurrence probability than shorter sequences and can be omitted. This theorem has come in the supplementary. We propose to use sequences up to length L/2 where L is the total number of layers. Also, the only important bigram (2-gram) is $'input - output'$ because the rest will appear in valid 3-grams.

B. Regression Results

At first, we test our encoding in a simple prediction task. A total sample of 2000 (1000+1000) architectures are taken randomly from the dataset to make the training and validation set. Similar to [11], we train a Gradient Boosting Decision Tree (GBDT) model with 100 trees and 31 leaves per tree for a single epoch. We compared our model with following baselines in NAS:

- Random Search: The simple and fast cell-based search method that combines random search with weight-sharing and is proposed in [18];
- NAO: A predictor-based NAS method which encodes the architecture cell in continuous space to optimize it there [7];
- RE: One of the first methods based on evolution and competes against RL in large spaces [16];
- Neural Predictor: It uses Graph Convolutional Networks to extract features and is proposed in [15];
- GBDT-NAS: The method proposes discrete encoding and uses GBDT as the predictor [11];
- BANANAS: The path-based predictor which proposes a one-hot feature vector to index the existing paths and Bayesian optimization [17];
- Weak NAS Predictor: A recent study which is proposed in [19] and focuses on using power of ensembles to improve the results rather than encoding.

The whole experiment is performed 50 times. For GBDT-NAS, Weak Predictors and BANANAS we used the authors' code. Also, we had valid codes for RE, RS and Neural Predictor but for NAO the values come from [15, 11]. Table 2 shows the results.

Table 2. Test and validation results of different NAS methods on CIFAR-10 using NASBench-101. The training and validation set, each consists of 1000 randomly sampled architectures.

Method	Test Acc(%)	Val Acc(%)	Test Regret
Random Search	93.7	94.5	0.62
NAO	93.90	94.1	0.42
RE	93.96	94.7	0.36
Neural Predictor	94.04	95.1	0.28
NAS-GBDT	94.14	94.5	0.18
BANANAS	93.9	94.5	0.42
Weak Predictors	94.23	94.9	0.09
Ours	94.21	94.9	0.11

We also tested several values for the best number of training samples (Figure 4.). We assumed an equal share for the number of training and validation samples. We varied the total number of samples up to 5000. After reaching 5000 samples, Regularized Evolution achieves 94.1% test and 94.8% validation accuracy; Random Search achieves 93.8% and 94.5% test and validation accuracies respectively. The weak predictors method reaches 94.2% and 95% test and validation accuracy. We observed after 2000 samples, the results do not improve significantly and there is a risk of overfitting. So, we chose 2000 samples with a fair share of 1000-1000 for the number of training and validation samples. Figure 4 shows the test accuracy results and Figure 5 presents the validation results.

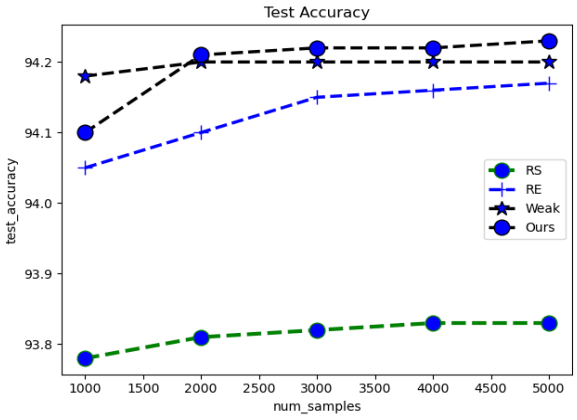


Figure 4. Test accuracy for a varied best number of samples

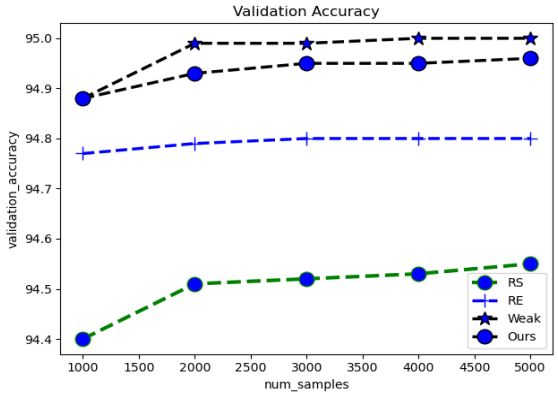


Figure 5. Validation results for a varied best number of samples

Table 3. The Kendall-Tau of using different encoders on the NAS-Bench-101 dataset. The encoders are trained with different proportions of the first 90% (381262) architectures tested with other 42362 ones.

Encoder	Proportions of 381262 training instances (%)							
	0.05 %	0.1 %	0.5 %	1%	5%	10 %	50 %	100 %
MLP	0.39	0.52	0.64	0.73	0.85	0.87	0.8	0.89
LSTM	0.55	0.59	0.71	0.77	0.84	0.85	0.88	0.89
GCN	0.55	0.57	0.79	0.82	0.86	0.87	0.89	0.89
GATES	0.76	0.77	0.84	0.85	0.88	0.89	0.90	0.90
Ours	0.80	0.85	0.86	0.87	0.88	0.88	0.89	0.90

3. Ranking results

A predictor evaluates an untrained model so it may underestimate a model's actual performance. If the predictor exhibits a uniform behavior towards each model, the results can be used to produce a meaningful ranking and the underestimation would not cause a problem. Thus, the relative ranking of the architectures is more explanatory than their exact performance values. Here, we adopt Kendall's Tau ranking correlation to evaluate several performance predictors. To keep up with [27]'s setting and use their reports, we used the first 90% (381262) architectures as the training data and the other 42362 architectures as the test data. Table 3 shows the Kendall's Tau correlation for several base predictors such as LSTM [28], MLP [28], GCN [29], and GATES [27] which is a new graph-based predictor. The proportions of training data is varied from 0.05% to 100%. The results related to our predictor on the small training sets (0.05% and 0.1%) proves its generalization ability. The data for the mentioned predictors come from [27].The other measure that we considered is the N@K which evaluates the true ranking among the top-k architectures. Table 4 shows the results.

The estimated average FLOPS of the found architectures with our method is 512M and the parameters are 2.8M. The found architectures by different cell-based NAS methods are compared in terms of number of parameters in Table 5. The methods include the RL-based Amoebanet [16], Progressive NAS (PNAS) [4], DARTS[30], ENAS [31] and GATES [27].

Table 4. N@K on NAS-Bench-101. The predictors are trained with 0.1% of the training data (381 architectures).

Encoder	N@5	N@10
MLP	1397 (3.30%)	552 (1.30%)
LSTM	1080 (2.54%)	312 (0.73%)
GCN	405 (0.97%)	405 (0.95%)
GATES	27 (0.05%)	27 (0.05%)
Ours	585 (1.38 %)	526 (1.24%)

Table 5. Comparison of discovered architectures on CIFAR-10

Method	Number of Parameters (M)	Number of Evaluated Archs.
Amoebanet[16]	2.8	20000
PNAS[4]	3.3	27000
DARTS[30]	3.2	1160
ENAS[31]	4.6	--
GATES[27]	4.1	800
Ours	2.8	500

The next step is to test the method on a more challenging dataset which was ImageNet in our case. Part *B* discusses the results we observed.

5.2. ImageNet Experiments

To evaluate our model on ImageNet, we used NASBench-201 which consists of validation and test results on about 16,000 architectures [13]. The results are reported on CIFAR-10, CIFAR-100, and ImageNet-16-120. The search space is similar to NASBench-101, however the cells are bound to be fully connected with four nodes but can choose from a variety of five operations per node. The main architecture is composed of a pre-convolution layer, followed by 3 times repetitions of the cell separated by a residual block of stride 2 and a final global average pooling layer. The lowest validation error and test error reported after 500 runs are 53.233% and 53.1556%, respectively which do not belong to the same architecture. The problem of inconsistency is more severe here which can lead to overfit to validation errors and increase in test error. We add the Input and Output to the operation set to form operational sequences. The feature vector length is 180. After 50 runs which took less than 1 TPU hour, we reached 54% validation error and 54.10% test error. BANANAS reported 54.6% and 54.7% for validation and test error after 200 trials which took over 40 TPU hours. We tested BANANAS with its proposed truncated path-based encoding and observed an average of 64% test error there.

We also calculated the Kendall-Tau correlation coefficient this time for NASBench-201 cells on ImageNet. Table 6 shows the results. The method is compared with MLP, LSTM, and GATES encoder. The encoder is trained using the first 50% (about 7800) samples and tested on the rest (about 7800 samples). The proportion of training data changes from 1% of the training set to 100%. As it can be

seen clearly in the table, our encoder reached much higher ranking correlations than other encoders. The first stage of experiment uses only 78 training samples and exhibits 91% correlation with the target values.

Table 6. The Kendall-Tau of different encoders on the NAS-Bench-201. The encoders are trained with different proportions of 50% (8713) of architectures and tested with other 7812 ones.

Encoder	Proportion of 7813 training instances				
	1%	5%	10%	50%	100%
MLP	0.0974	0.3959	0.5388	0.8299	0.8703
LSTM	0.5550	0.6407	0.7268	0.8791	0.9002
GATES	0.7401	0.8628	0.8802	0.9192	0.9259
Ours	0.9109	0.9125	0.9171	0.9227	0.9428

6. Related works

A proper encoding of the neural architectures is needed when sampling a new architecture, perturbing it in some way and training the predictor with it. There is not much study regarding the optimal encoding for each phase [20], thus, different encodings often mismatch. The architecture encoding can roughly be categorized into vector encoding and graph encoding [21]. Encoding via vectors is the serialization of the architecture mostly layer by layer (or block by block). For example, [6, 22, 23] had vector encodings composed of the layer type, kernel width (height), channel number, output to input ratio (layer information), number of consecutive convolution layers/blocks, pooling to convolution ratio (sub-network information), and some summary statistics. However, these features are not rigorous enough to encode all types of structures. The topological information also is not properly included. Besides, as long as it matters the predictor, the encoding should produce the same representation for isomorphic architectures [24]. The adjacency matrix provides a more powerful representation of the network. Each element of the binary matrix shows the existence of a connection between the two respective layers. Some other versions also represent the type of transformation/operation between the two layers [25]. An operation can be defined such that it can also show the layer type, kernel size, channel number, etc. (e.g., MaxPool 3x3). Encoding with adjacency matrix is not one to one, and different architectures may have similar encodings. To make it worse for a neural predictor, an architecture can have many different matrix representations. BANANAS [17] adopted a path-based encoding. They extracted all the input-to-output paths in the neural network (in terms of the operations) and considered a binary feature for each path. The corresponding features of the present paths are set to 1s. Although, not being suitable for macro-level search space, the study showed that path-based encoding significantly increases the performance of neural predictors and is the best choice for them. This is because the features do not depend on one another (each feature represents a unique path) and each neural architecture is mapped to only one encoding. Path-based encoding is also applied in [21, 25, 26].

Path-based encoding handles structural isomorphism but it may map different architectures to the same encoding as well. However, as it is discussed in [24] such architectures actually have a similar behavior. We use [20] as a benchmark to compare our results with. The Graph-based encodings apply Graph Convolutional Networks (GCN) to encode the neural architectures. The GCNs often take the directed acyclic graph of the network and embed them to fixed-length vector representations [15]. These encodings represent the topological information very well. The graphs need retraining each time a new dataset or space is added. However [17] claims that based on experimental results, the best-performing neural predictors are the feedforward network with the path encoding and the GCN with graph-based encoding. The feedforward networks had shorter runtime compared to the GCN and autoencoders. It is discussed in [11] that the representation data of an architecture is more tabular rather than continuous and is not well suited to the widely used neural predictors such as RNN, CNN and GCN. Their study proposes a tabular one-hot encoding such that bits of 0/1 describe a layer. Each bit defines a property of the layer in terms of existence/non-existence of a particular property in a specific layer: convolution: Yes/No, kernel size 3x3: Yes/No, skip connection with the layer i: Yes/No. This consecutive binary decisions build a tree structure in which each branch defines a whole unique structure. This tree structures representation is limited to 5 layers. The encoding is unable to properly show the interconnections and can support a very limited set of operations, thus it is not suitable for macro architectures or complicated cells. They applied a tree-based predictor (GBDT) to receive the encoding. A tree based predictor can very well handle discrete representations. Besides, the limited amount of training instances (which is very common in NAS) does not harm tree based predictors as much as it harms the neural predictors. The results showed significant enhancement in accuracy and elapsed time for simple cell-based architectures and is considered state of the art for 2020. In our study, we pursue path-based encoding and propose a tree-like encoding of the network. The feature vector counts the number of basic subtrees of arbitrary length. This is used as a kind of similarity between trees or different networks. Finally, we apply GBDT to take the encoding and do its task as the performance predictor.

7. Conclusion

Automatically designing neural architectures, i.e., NAS, is a promising path in machine learning. However, the main challenge for NAS algorithms is reducing the elapsed time on evaluating a proposed network. A recent strategy which attracted much attention has been using surrogate predictive models. The predictive models attempt to forecast the performance of a neural model ahead of training. In this study, we proposed to leverage both structural features of a deep network and attributes of each layer to learn an encoding for surrogate predictor. We proposed to represent each neural network via a tree structure and then extract features from the given tree. Extensive results on two widely used datasets of NAS task demonstrate the effectiveness of the proposed model. Many extensions of this work can be exploited. For example, structural features can be extracted using graph-based method like random

walks and graph convolutional networks.

Many existing deep models are vulnerable to adversarial attacks. Various methods have been proposed to design network architectures that are robust to one particular type of attacks or a random selection of them. We might pursue the idea of using proposed encoding in RAS (Robust Architecture Search) to test its ability against adversarial attacks and to observe on what kind of attack, it performs best. Training a robust model is quite challenging, so it needs to be fully investigated in an independent future work.

8. Appendix

8.1. Encoding of NASBench101

A simple cell is presented and encoded into vector in Figure 6.

8.2. Performance on isomorphic graphs

We provided a set of 10 architectures consisted of 5 pair of isomorphic graphs and tested the encoding to see whether or not it recognizes isomorphism. An instance of isomorphic cells found in NASBench-101 is presented Figure 7. The results are listed in Table 7.

8.3. Qualitative talk on best found cells

Here in Figure 8 we bring some of the 20-best performing NAS-Bench 101 cells together with the operation paths produced by the algorithm for a better view. Note that the results are reported on CIFAR-10 and are not to be generalized to other problems but can be a hint.

As it can be seen in Figure 9, the encoding vector is simple and interpretable. We observe that some paths are common in almost all these cells. For example, there exists a direct path/skip connection between the input layer and output layer in all the mentioned cells. More over the cells with more convolution layers act better than their similar networks where some layers are replaced with MaxPooling and 3x3 convolution seems a better choice than 1x1 but the key is to place them at the crowded layers (the layers with more connections). The cells can become more compact using mostly convolution layers and still have a high performance.

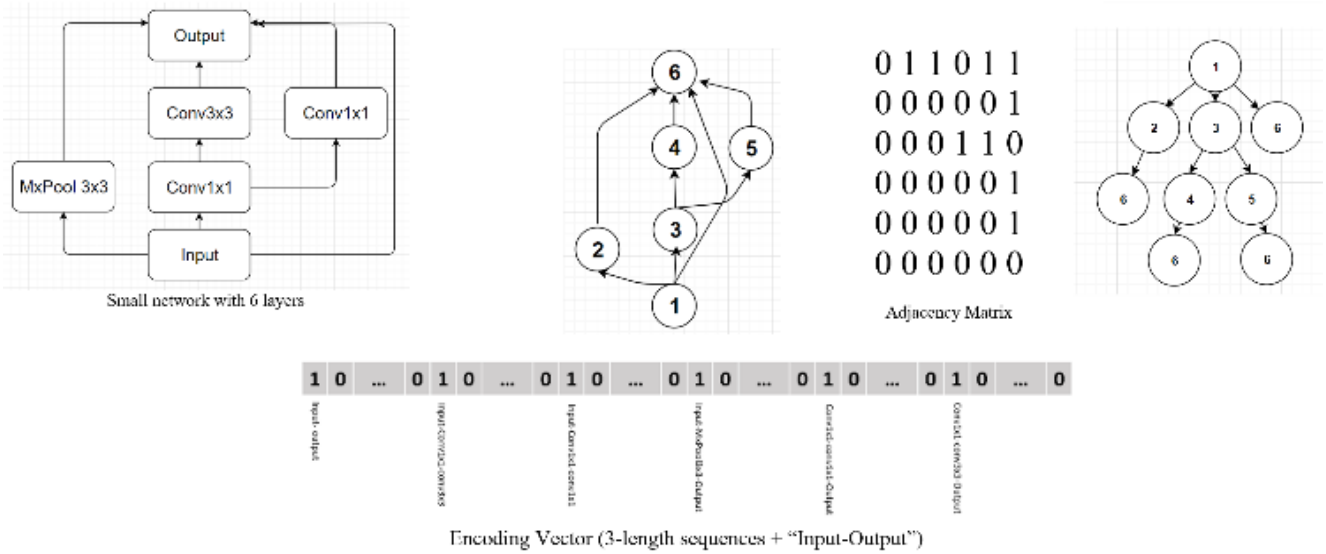


Figure 6. An illustration of graph and node representations. A neural network architecture with 5 possible operations per node defined in string format (Left). The paths are recognized (Top right) and the encoding vector is built up using both the path graph and the list of operations (Bottom).

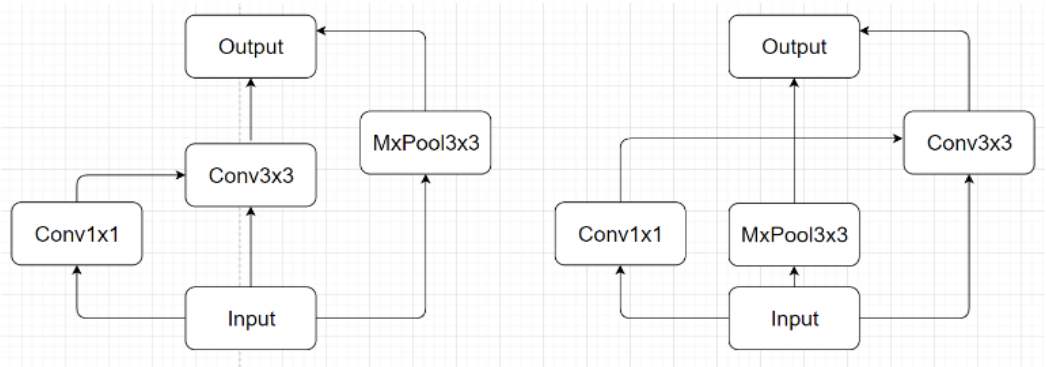


Figure 7. Isomorophic cells from NASBench-101

Table 7. Test accuracies on 5 NASBench-101 isomprphic pairs

	1	2	3	4	5
(a)	93.40	93.01	94.1	93.5	94.1
(b)	93.40	93.01	94.1	93.5	94.1

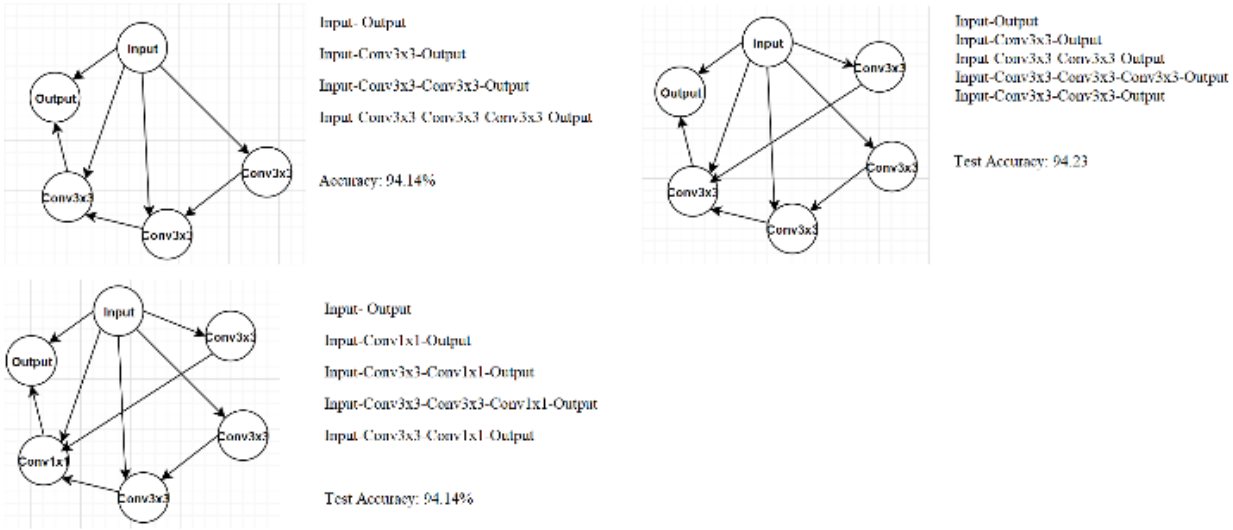


Figure 8. Best compact cells with extracted operation paths

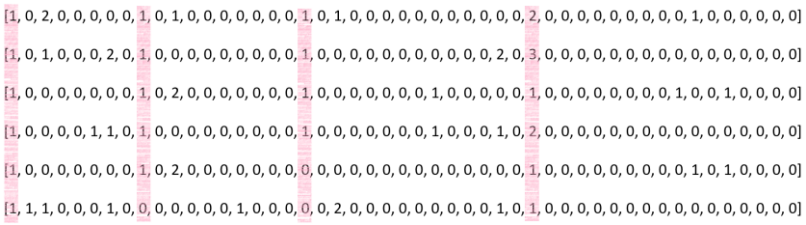


Figure 9. The 3-gram counter feature vector of 6 cells from the best performing set. The first element recognizes the 'input-output' sequence.

8.4. Selecting the length of sequences

We practically observed that for Bench-101 and 201 3-length sequences produce satisfying results. But we also used the following theorem to decide on the length of operation sequences to consider in the feature vector.

If $G_{n,k,r}$ denotes a graph with n nodes, r choices of operations on each node, and k expected number of edges the following theorem is applied.

Theorem 1. Given integers $r, c > 0$ there exists N such that $\forall n > N$, there exists a set of n paths $\mathcal{P}'$ and the probability that $G_{n,n+c,r}$ contains a path not in $\mathcal{P}'$ is less than $1/n^2$.

The theorem says that when $k = n + c$ and when n is large enough compared to c and r, the probability that random sampling method outputs a graph $G_{n,k,r}$ with a path outside of $\mathcal{P}'$ is very small. For the proof see [17].

References

[1] M. B. Zoph, Q. V. Le, Neural architecture search with reinforcement learning, arXiv preprint arXiv:1611.01578 (2016).

[2] B. Baker, O. Gupta, N. Naik, R. Raskar, Designing neural network architectures using reinforcement learning, arXiv preprint arXiv:1611.02167, (2016).

[3] A. Brock, T. Lim, J. M. Ritchie, N. Weston, Smash: one-shot model architecture search through hypernetworks, arXiv preprint arXiv:1708.05344 (2017).

[4] H. Liu, K. Simonyan, Y. Yang, Darts: Differentiable architecture search, arXiv preprint arXiv:1806.09055 (2018).

[5] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, R. Fonseca, Neural architecture search using deep neural networks and monte carlo tree search, Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, (2020)

[6] B. Deng, J. Yan, D. Lin, Peephole: Predicting network performance before training, arXiv preprint arXiv:1712.03351 (2017)

[7] R. Luo, F. Tian, T. Qin, E. Chen, T.-Y. Liu, Neural architecture optimization, arXiv preprint arXiv:1808.07233 (2018)

[8] B. Baker, O. Gupta, R. Raskar, N. Naik, Accelerating neural architecture search using performance prediction, arXiv preprint arXiv:1705.10823 435, (2017)

[9] Z. Zhong, J. Yan, W. Wu, J. Shao, C.-L. Liu, Practical block-wise neural network architecture generation, in:

Proceedings of the IEEE conference on computer vision and pattern recognition, (2018).

[10] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, (2018).

[11] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, T.-Y. Liu, Neural architecture search with gbdt, arXiv preprint arXiv:2007.04785 (2020)

[12] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, F. Hutter, Nas-bench-101: Towards reproducible neural architecture search, in: International Conference on Machine Learning, PMLR, (2019).

[13] X. Dong, Y. Yang, Nas-bench-201: Extending the scope of reproducible neural architecture search, arXiv preprint arXiv:2001.00326 (2020).

[14] A. Veit, M. J. Wilber, S. Belongie, Residual networks behave like ensembles of relatively shallow networks, Advances in neural information processing systems 29 (2016).

[15] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, P.-J. Kindermans, Neural predictor for neural architecture search, in: European Conference on Computer Vision, Springer, (2020)

[16] E. Real, A. Aggarwal, Y. Huang, Q. V. Le, Regularized evolution for image classifier architecture search, in: Proceedings of the aaai conference on artificial intelligence, Vol. 33, (2019)

[17] C. White, W. Neiswanger, Y. Savani, Bananas: Bayesian optimization with neural architectures for neural architecture search, arXiv preprint arXiv:1910.11858 1 (2) (2019).

[18] L. Li, A. Talwalkar, Random search and reproducibility for neural architecture search, in: arXiv preprint arXiv:1902.07638, (2019).

[19] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, L. Yuan, Weak nas predictors are all you need, arXiv preprint arXiv:2102.10490 (2021).

[20] C. White, W. Neiswanger, S. Nolen, Y. Savani, A study on encodings for neural architecture search, arXiv preprint arXiv:2007.04965 (2020)

[21] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, J. Liang, Npenas: Neural predictor guided evolution for neural architecture search, arXiv preprint arXiv:2003.12857 (2020).

[22] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, M. Zhang, Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor, IEEE Transactions on Evolutionary Computation 24 (2), (2019)

[23] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, A. C. I. Malossi, Tapas: Train-less accuracy predictor for architecture search, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, (2019)

[24] X. Ning, Y. Zheng, T. Zhao, Y. Wang, H. Yang, A generic graph-based neural architecture encoding scheme for predictor-based nas, in: Computer Vision–ECCV 2020: 16th European Conference, 28, 2020, Proceedings, Part XIII 16, Springer, (2020)

[25] E.-G. Talbi, Optimization of deep neural networks: a survey and unified taxonomy (2020).

[26] H. Cai, T. Chen, W. Zhang, Y. Yu, J. Wang, Efficient architecture search by network transformation, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, (2018)

[27] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. *arXiv preprint arXiv:2004.01899*, (2020)

[28] Wang, L., Zhao, Y., Jinnai, Y., Fonseca, R.: Alphax: exploring neural architec- tures with deep neural networks and monte carlo tree search. arXiv preprint arXiv:1805.07440 (2018)

[29] Shi, H., Pi, R., Xu, H., Li, Z., Kwok, J.T., Zhang, T.: Multi-objective neural architecture search via predictive network performance optimization. arXiv preprint arXiv:1911.09336 (2019)

[30] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)

[31] Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)

[32] Minghao Guo, Yuzhe Yang, Rui Xu, Ziwei Liu, Dahua Lin; When NAS Meets Robustness: In Search of Robust Architectures Against Adversarial Attacks, Proc. Of the IEEE/CVF conference on Computer Vision and Pattern Recognition (CVPR), (2020)

[33] Liu J., Jin Y., Multi-Objective search of robust neural architectures against multiple types of adversarial attacks, Neurocomputing Vol. 453 (2021).