IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 July 2022, accepted 14 September 2022, date of publication 22 September 2022, date of current version 3 October 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3208591

RESEARCH ARTICLE

Learning a Unified Latent Space for NAS: Toward Leveraging Structural and Symbolic Information

SAEEDEH ESLAMI[1], REZA MONSEFI[1], (Member, IEEE), AND MOHAMMAD AKBARI[2]
[1]Computer Department, Engineering Faculty, Ferdowsi University of Mashhad, Mashhad 9177948974, Iran
[2]Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran 15875-4413, Iran

Corresponding author: Reza Monsefi (monsefi@um.ac.ir)

ABSTRACT Automatically designing neural architectures, i.e., NAS (Neural Architecture Search), is a promising path in machine learning. However, the main challenge for NAS algorithms is to reduce the considerable elapsed time to evaluate a proposed network. A recent strategy which attracted much attention is to use surrogate predictive models. The predictive models attempt to forecast the performance of a neural model ahead of training, exploiting only their architectural features. However, preparing the training data for predictive models is laborious and resource demanding. Thus, improving the model's sample efficiency is of high value. For the best performance, the predictive model should be given a representative encoding of the network architecture. Still, the potential of a proper architecture encoding in pruning and filtering out the unwanted architectures is often overlooked in previous studies. Here, we discuss how to build a proper representation of network architecture that preserves explicit or implicit information inside the architecture. To perform the experiments, two standard NAS benchmarks, NASbench 101 and NASbench 201 are used. Extensive experiments on the mentioned spaces, demonstrate the effectiveness of the proposed method as compared with the state-of-the-art predictors.

INDEX TERMS Neural architecture search, search space pruning, network architecture, representation learning.

I. INTRODUCTION

Deep neural networks have been successfully used and addressed various challenging tasks, including computer vision, speech recognition, machine translation, and medical diagnosis, in recent decades. This success has sprung from well-designed network architectures and powerful computing machines. The manual process of designing a new neural model for a specific problem is time and labor consuming. Besides, relying on expert experience often results in subjective sub-optimal solutions. This has resulted in an emerging branch in machine learning which attempts to automatically find the optimal neural model for a specific task at hand, namely Neural Architecture Search (NAS).

The optimal answer, there, consists of three parts: the optimal structure of the model, the optimal parameters and the optimal hyperparameters. Traditional optimization methods address the parameter optimization problem. The challenge, however, lies in simultaneous optimization of the structure and hyperparameters of a neural model due to interdepencency among the hyperparameters and the structure of a network. Thus, separate optimization of the architecture or the hyperparameters, may yield a sub-optimal solution [1]. However, per deep structure, vast selection of hyperparameter exist which form the pool of candidate models. A combination of possible structures and their corresponding hyperparameters, regarding a certain task, builds an enormous space of options, called NAS search space. Such a space would contain at least a few hundred thousand different models. The challenge would be much severe in real-world tasks. Conventional search methods [2], [3], [4], often fully train and evaluate a large part of the space to converge; which is very costly. Thus efficient methods for searching these huge search spaces is highly desirable.

Retrospective studies in literature have proposed useful strategies for addressing such a big search space including:

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

proxy estimates, developing warm starts, early stopping via curve analysis, one-shot learning and using surrogate predictive models. While leveraging proxy estimates produced by simplified search spaces, is fast, several studies demonstrates its unreliable results [5], [6], [7]. Curve extrapolation methods [8] seek to stop training for presumably week models using the elapsed learning curve and require heavy computations. One-shot methods [9], [10] train only one large super-graph and share the attained weights on common edges among the sampled sub-graphs. So they have a fast convergence but less efficient results and suffer from a large memory consumption.

Lately, an efficient strategy based on predictive models has been adopted in which candidate architectures are evaluated with no training [7], [11]. The predictor is a learner; trained with a few pair of fully trained networks and their performance values. The hypothesis behind the predictor approach is to build a supervised learning model that can catch the most influential network features from training set. If the selected networks are of high quality, training with this small set will help the predictor to reliably select a good model. However, training a meta-predictor is nontrivial due to following challenges. (1) **Lack of training data**. To train a predictor model, we need an annotated dataset in which each architecture is labeled with its actual performance. As such, each model in the training set should first be fully trained and then evaluated on the problem data. For instance, to build a set with 1000 samples, although this number is quite small as compared to the size of the search spaces, we need several experimental runs with considerable GPU time. (2) **Proper representation of symbolic data**. Proper representation of the input data is a vital requirement for the success of any learning model. The network architecture is a symbolic data composed of both topological information, i.e. the structure of the network architecture, and numerical information, i.e. the hyper-parameters of the networks. To feed such data into a learning model, a proper representation should be constructed, i.e. the encoding of the architecture. A representative encoding should preserve the dependency of the layers in each path of the network from input to the output layer, which transforms and aggregates the data.

The representation of the architectures for the predictor, has rarely been studied. Existing methods either fix or limit the network topology to be able to encode the hyperparameters [12], or use global features or layer-by-layer features [5], [13], [14] which overlook the topology information. It is worth noting that graph-based encoders that recently achieved best results, however, can encode the network as a whole [11], [15]. These encoders, although, preserve the topology, they need lots of training data which is rarely available in NAS. Further, they can overfit on cell-based networks as the complexity of the predictor exceeds the complexity of the network cells. These complex predictors, indeed, tend to memorize good architectures rather than learning appropriate features.

There are certain sequences of (network) processes that lead to superior performances. We try to discover decisive patterns inside a network and use it as a measure to estimate the performance. In this study, we propose a method to encode and integrate the structural and non structural features of the network as a whole via extracting substructures as local attributes. To extract structural features, we use tree kernels and for non structural features we propose an operation coding method. The proposed method can model nested and multi-connection networks in polynomial time.

The basic of the method is to convert a deep convolutional model into tree kernels using a small set of samples and then running kernel-level matching. So it relies on two concepts: *finding the kernels* and *effective sampling of the search space*

To perform the experiments, we use two standard benchmarks, NASbench 101 and NASbench 201, trained on the CIFAR-10 and ImageNet datasets. For each architecture, the datasets also include the respective training and evaluation statistics. These information have been collected in standard and reliable conditions and are now a basis for fair comparison of neural network architecture search algorithms. Experiments with the proposed method show a significant improvement in sample efficiency of the predictor and its performance estimation. The experiments also show that the knowledge of the predictor can be transferred to other similar problems to find strong architectures. The main contributions of this paper are as follows:

- The importance of structure in NAS is investigated and we introduce a method to embed the structure in network encoding.
- For the first time in NAS we propose to use tree kernels in finding structural similarities between two networks.
- We investigate the effect of lack of sufficient data on the performance of the predictive model. At present, our method has the best sample efficiency compared to a wide range of methods that had the best performance in 2020 and 2021.

II. RELATED WORK

In different steps of the architecture search process (sampling, modifying and training), a proper encoding of the neural architectures is needed. Although there is not much study regarding the proper encoding, a general classification for network encoding methods is as follows.

A. DIRECT ENCODING

The direct encoding is the common sequential or layer-by-layer network encoding for flat feed-forward networks with no extra branch. The primary network can be reproduced directly from the network code. Topology-related parameters such as the number of hidden layers, the number of neurons, the layers' types, kernel sizes and a set of general characteristics such as learning rates and biases can be encoded this way [13], [14], [16]. Due to the presence of dependent variables, a network encoding can have variable length. For

example, topology depends on the number of layers and hyper-parameters depend on the type of operation. There are several issues about using direct encoding in NAS: **(1):** These features are not capable enough to display branched or hierarchical architectures and many more. **(2):**Structural information such as layer dependency and sequence of operations, can only be implicitly modeled. **(3):** Vector distances can produce different representations for an architecture, which can be confusing for a predictive model. The reason is that they are affected by the traversal order of nodes. The same issue exists with word-based coding methods such as TAPAS [14], QBlockQNN [17], E2EPP [13], SMASH [9], and a number of others. So, they cannot guarantee the mapping of isomorphic architectures to one representation [18]. Path-based encoding [19] however, has solved this using the traversal order of the nodes in an input to output path. **(4):**Direct encoding often faces immeasurable challenges; by increasing network depth and search space size, the length of the coding string increases. For example, the encoding schema proposed by [19] produces encoding vectors of length m^n (minimum) for a network with n layers and m internal operations.

B. INDIRECT ENCODING

The indirect encoding methods need a decoder to reproduce a network from its code. Indirect encoding allows for more compact representation because it does not describe the network in detail. The most well-known indirect methods are based on LSTM and graph networks.

The Graph-based encodings apply Graph Convolutional Networks (GCN) to take the directed acyclic graph of the network and embed them to fixed-length vector representations [20], [21]. These encodings represent the topological information and graph structured data very well. There are two levels to do so: *graph level* and *node-level*. For graph level, the whole graph is considered as an individual to be labeled. For node-level, the inputs are considered vertices of a graph and their relation is reflected in the connections. The task is to have GCN, predict their labels using this relation graph [20]. Several layers of GCN can be stacked together to reach a stronger representation.

Recently, the use of GCN [22] as a predictive model has led to superior results in NAS [20], [23], [24]. There are also considerations regarding graph convolutioanl networks as NAS predictors: **(1):**It is discussed in [12] that the architecture representation data is more tabular rather than continuous and is not well suited to the widely used neural predictors such as RNN, CNN and GCN. **(2):** Despite the high accuracy results, the use of graphical convolutional networks leads to unnecessary calculations and the possibility of over-fitting [22], [24]. **(3):** The graphs fit to and memorize good architectures rather than extracting design rules. They also need retraining each time a new dataset or space is added [18].

Instead of memorizing the graphs, our method learns the decisive substructures in the form of tree kernels to preserve the. It exploits the structural similarity measured based on the common kernels to encode graphs in the latent space.

III. PROBLEM STATEMENT

The task at hand is to predict 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 $\mathcal{P}$ to take the representation of an architecture $N_i \in \mathcal{N}$ as $e(N_i)$ and return its estimated performance as $\hat{y}_i = \mathcal{P}(\mathbf{W}_p, e(N_i))$, where $\mathbf{W}_p$ denotes the trainable function parameters. The function $e(.)$ takes an architecture and embeds it into a real valued vector. The function $\mathcal{P}$ is trained to solve the following minimization problem,

$$\min_{\mathbf{W}_p} \|\mathcal{P}(\mathbf{W}_p, e(N_i)) - y_i\|_2^2, \tag{1}$$

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 so that the regressor, i.e. $\mathcal{P}$, can accurately estimate the network performance.

IV. PRELIMINARIES

Before diving deep into technical details, we first describe basic definitions related to our work.

A. KERNEL-BASED STRUCTURAL LEARNING

A neural network can be described by a labelled directional graph $G = (V_G, E_G)$, where V_G denotes its set of nodes and $E_G \subset V_G \times V_G$ shows the set of edges. To learn and find the similarities in graph-structured data, kernels can be applied. Although the kernels differ in the way they are calculated and the types of features they extract, the idea of using kernels to compare two graphs is to break them down into substructures (nodes or sub-graphs) and calculate the kernel for each pair of substructures. Let $\mathcal{G}$ be a graph space; the kernels compare the $G, H \in \mathcal{G}$ by calculating the difference between their components.

Assume $\mathcal{F} = F_1 \times \ldots \times F_k$ is the space of components that make up $G \in \mathcal{G}$. Also suppose $F : \mathcal{F} \rightarrow \mathcal{G}$ is a mapping of the components to the graph, so that $F(g) = G$ if and only if $g \in \mathcal{F}$ components, build the graph G. If $F^{-1}(G) = \{g \in \mathcal{F} : F(g) = G\}$, then: $k(G, H) = \sum g \in F^{-1}(G) \sum h \in F^{-1}(H) \prod_{i=1}^{d} k_i(g_i, h_i)$. Here the function k_i, is the kernel on F_i and $F^{-1}(G)$, is the inverted mapping and the set of all components of the graph G [25]. Each pattern represents a set of homogeneous, identical graphs. The function $\phi(G)_{\sigma_i}(1 \leq i \leq N)$ counts the occurrence of sub-graphs of type σ_i in graph G and the kernel counts the simultaneous occurrence of sub-graph patterns in respective graphs. Thus, we have:

$$\begin{cases} k_{subgraph}(G, H) =< \phi_{subgraph}(G), \phi_{subgraph}(H) > \\ \phi_{subgraph}(G) = ((G)_{\sigma_1}, \ldots, (G)_{\sigma_N}) \end{cases} \tag{2}$$

Each $\phi_i(G)_{\sigma_j^r}$ counts the occurrence of nodes labeled $\sigma_j^r \in \Sigma$ where Σ is the set of allowed labels. The time required to calculate kernels increases exponentially as the size of the sub-graphs increases. Actually for graphs of size n the time required to find patterns of size k is of order $O(n^k)$. We propose parsing the networks into tree sub-graphs. The order

of computations to match sub-trees is $O(hm)$ where h is the depth of the longest sub-tree and m is the maximum number of layers. The sub-trees can be compared using a special kind of structured kernels, called tree kernels [26]. Next we're going to talk about tree kernels and how to exploit them in NAS problem.

B. TREE KERNELS

The seminal work on Convolution Kernels by [27] defines a broad class of kernels on discrete structures by counting and weighting the number of substructures they share. Let $G = (V_G, E_G)$ be a graph. We define a sub structure of G as a kernel tree t, where $t = (V_t, E_t)$, $V_t = (n_1, .., n_{|t|})$ is a tree. A sequence of $|t|$ nodes $(v_1, .., v_{|t|}) \in V_G^{|t|}$ builds a tree kernel of G if and only if the following conditions hold:

$$\begin{cases} \forall i \in [1, |t|] \rightarrow label(v_i) = label(n_i) \\ \forall (n_i, n_j) \in E_t \rightarrow (v_i, v_j) \in E_G \\ \text{and } label(v_i, v_j) = label(n_i, n_j) \end{cases} \tag{3}$$

The tree kernel t is a member of the set of all possible tree kernels $\mathcal{T}$, of the graph G. The function $\phi_t(G)$, counts the number of times the tree kernel t rooted from node v occurs in G:

$$\phi_t(G) = |(\alpha_1, \ldots, \alpha_{|t|}) \in \{1, |V_G|\}^{|t|}|, \quad (v_{\alpha_1}, \ldots, v_{\alpha_{|t|}}) \in \mathcal{T} \text{ and } v_{\alpha_1} = v \tag{4}$$

The structural learning now models the problem as identification of the common tree kernels between two distinct graphs as follows:

$$K(G_1, G_2) = \sum_{t \in \mathcal{T}} w(t)\phi_t(G_1)\phi_t(G_2). \tag{5}$$

Here, $\mathcal{T}$ is a set of trees, ϕ_t counts and $w : \mathcal{T} \rightarrow \mathcal{R}$ weights the tree kernels. When $w(t)$ tends to 0, only small linear (single nodes) subspecies are preserved and vise versa.

V. PROPOSED FRAMEWORK

In this section, we first introduce our proposed framework, namely S^2i, which embed both **S**tructural and **S**ymbolic **i**nformation (S^2i) into a latent space, as shown in Figure 1.

The proposed framework is comprised of three components: (1) Building the Latent Space; (2) Learning the Predictor, and (3) Evaluating the Search Space. The first component builds a training set for the predictor, i.e., the second component. Here, the input is a set of fully trained architectures tagged with their performances coming from real execution. These architectures would be converted into distributional vectors in the united latent space, where the structural information are extracted via tree kernels and the symbolic information are directly represented as latent dimensions. For each candidate architecture, the input to the predictor is a $\mathbb{R}^k_{\geq 0}$ feature vector. The i-th element of the vector is the network's similarity score or number of tree kernel with the i-th tree of a proxy set. The second component take the training set as input and train a supervised model for estimating the performance of any given network. Finally, the last component evaluates the search space via the built predictor coming from the previous component.

A. MODELING NETWORK STRUCTURE

Contrary to fully-connected networks, data is transformed via different paths in multi-stream networks [28]. A representation should denote all the processes through which features are extracted. We propose to model a network by a set of operation sequences in a tree structure as shown in figure 2. In this tree any path from the root node to a leaf node is a neural function extracting features. In the sample network (left), the 'INPUT' layer immediately connects to a 'MAXPOOL' layer via a skip connection and to a 'CONV3 × 3' layer via the main stem which already can form two different paths. The 'MAXPOOL' layer immediately connects to the 'Output' and thus a path ends. The main stem contains several skip connections. The 'CONV3 × 3' connects to a 'CONV1 × 1' via a skip connection and to a 'MAXPOOL' via the main stem. The 'CONV1 × 1' reaches to 'OUTPUT' and the path ends. The connections are checked in a semi depth-first order beginning from l_1. This can be stated as follows: *Suppose* $\sigma(l_1, \ldots, l_L)$ *defines the set of paths of the tree composed of* $l_1, \ldots, l_L$ *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* $1 \leq i < j < k \leq L$ *if* $l_i \in \Delta(l_k) \setminus \Delta(l_j)$ *there exists a* l_m $(i < m < j)$ *such that* $l_m \in \Delta(l_j)$. The paths are identified one by one and attached to the previously built 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. The pseudo code of an algorithm for identifying the paths and their representations is presented in Algorithm 1.

1) COMPUTING TREE KERNELS

Instead of entering the feature space directly, tree kernels calculate the degree of similarity between two trees in terms of the amount of common fragments or sub-trees between them. The similarity score is affected by the length of adjustment or probable gaps along the matched sub-trees.

Let $|F| = \{f_1, f_2, \ldots, f_{|F|}\}$ be the space of fragments. The indicator function $I_i(n)$ is equal to 1, if f_i is rooted at node n, otherwise it is 0.

$$I_i(n) = \begin{cases} 1 & \text{if f}_i \text{ is rooted at node } n \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

The kernel function is defined as follows:

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{7}$$

Here N_{T_1} and N_{T_2} are the set of nodes of two trees T_1 and T_2, respectively, and the delta function $\Delta(n_1, n_2) = \sum_{i=1}^{|F|} I_i(n_1)I_i(n_2)$ computes the number of pieces that are common to T_1 and T_2 and have their roots in n_1 and n_2. The delta function can be rewritten as the sum of delta of

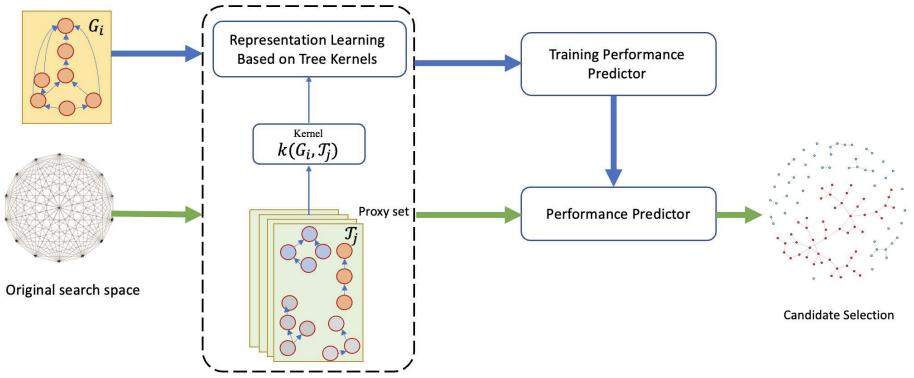


FIGURE 1. The schematic framework for the proposed S^2i method. The framework consists of three phases: constructing feature space, training the predictor and search space evaluation. The blue arrows depict the training process and the green arrows depict the evaluation process.

Algorithm 1 Building The Network Tree Representation

Data: $Deg(A_i)$: Number of A_i's non-zero elements
Result: The tree of the network

Stack ← [];
$tree_1 \leftarrow l_1$;
Num_nodes ← 1 ;
Push_to_Stack($tree_1$);
while *Stack_isNot_Empty* **do**
  parent = *Pop_from_Stack*();
  i = *parent.index*;
  for $j \leftarrow i+1$ **to** L **do**
    if $A[i,j] == 1$ **then**
      $Num\_nodes \leftarrow Num\_nodes + 1$;
      $tree_{Num\_nodes}.parent \leftarrow parent$;
      $tree_{Num\_nodes}.index \leftarrow j$;
      Add $tree_{Num\_nodes}$ to *parent.Children*;
      if $j \neq L$ **then**
        Push_to_Stack($tree_{Num\_nodes}$);
      end if
    end if
  end for
end while

Algorithm 2 Evaluation of the Search Space for Best Architecture

Data: $\mathcal{A}$: Architecture search space, $\mathcal{P}$: The predictor
Result: Best predicted architecture

randomly select $k + m_0$ architectures from $\mathcal{A}$ to build proxy set K and training set D;
foreach *candidate m in D* **do**
  fully train m to acquire its performance.;
  convert the model m to tree using algorithm 1;
  compute kernels with K and embed m;
end
initialize the predictor using D;
repeat
  sample candidate pool C from $\mathcal{A} \setminus D$;
  foreach *candidate m in C* **do**
    embed m using kernels with K;
    evaluate m using the trained predictor;
  end
  $D \leftarrow$ candidates with the $top - k$ scores;
  update $\mathcal{P}$ with the enlarged D;
until *convergence*;

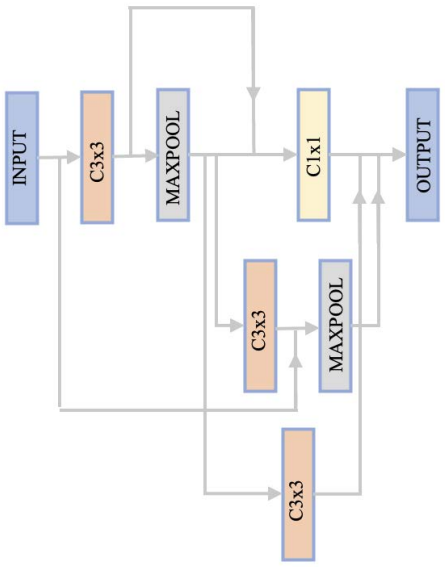
sequences of children of different lengths so that the calculations can be distributed among them [26]:

$$\Delta(n_1, n_2) = \mu\left(\lambda^2 + \sum_{p=1}^{l_m} \Delta_p(c_{n1}, c_{n2})\right) \tag{8}$$

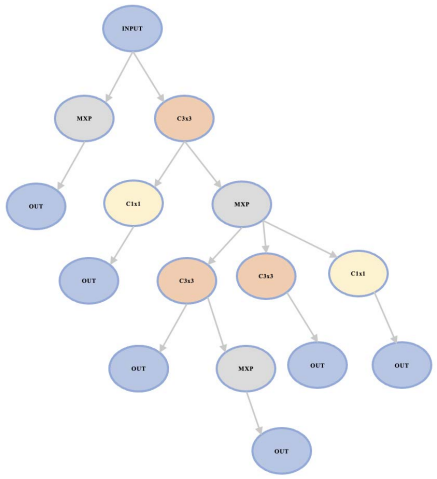
The Δ_p function calculates the number of common sub-trees in exactly p children (from both n_1 and n_2). The two parameters μ and λ are decay factors or can be functions of height and length of the child sequences. Entities like the 'OUTPUT' do not affect the score. If aS_1 and bS_2 are two child sequences (a and b are the first children), then:

$$\Delta_p(aS_1, bS_1) = \Delta(a,b) + \sum_{i=1}^{|S_1|}\sum_{j=1}^{|S_2|} \lambda^{|S_1|-i+|S_2|-j} \times \Delta_{p-1}\left(S_1[1:i], S_2[1:j]\right) \tag{9}$$

where $s_k[1:j]$ is the subordinate of the children from 1 to j in the k-th string. We assume each node of a tree as an entity and assign the operations to the nodes. Our goal is to discover the existence of a one-way connection between the two nodes E_1 and E_2 and therefore to find the dual $< E_1, E_2 >$ in the sequences. Matching the two sequences s_1 and s_2 is of order $O(p|s_1||s_2|)$. In the worst case, the computational complexity of calculating the kernels is $O(p\rho^2|s_1||s_2|)$, where ρ is the



(a) Main cell



(b) Tree structured representation

FIGURE 2. An exemplar neural cell (a) and its converted tree representation (b).

maximum branching factor of the two trees which is almost very small. We also may assign the operations to the edges and discover whether or not there is a relation R between the two entities i.e. to discover if the $< E_1, R, E_2 >$ exists.

B. MODELING LAYER ATTRIBUTES

So far, we solely modelled the structure of a neural model. To encode the non-structural information, we represent each operation by a four-part numeric vector, as shown in Figure 3. The first part, specifies the operation type and the rest three parts, determine the operation's hyperparameters. See Table 1 for the full list of operations. We might note that the '*SkipConnection*' is not a convolutional operation but it models skip connections in residual architectures. The '*No-op*' also is not an operation but it is useful in fully connected architectures and means no transaction between the respective layers. The operation set can be extended based on the task at hand. Repetitive patterns of operations can be grouped and form a combined operation. For instance, the operation set in NASbench 101 [29] includes '$CONVn \times n$' which is a $n \times n$ convolution followed by a BatchNorm and a ReLU operation.

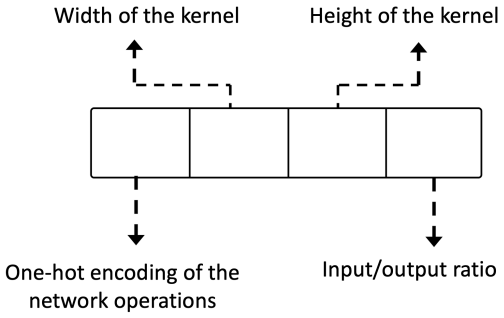


FIGURE 3. Vector representation of neural operations. Each operation can be fully represented by a vector of four values.

C. BUILDING THE PROXY SET

A key step in modeling network structures is to form a representative set of tree kernels. However, tree kernels are built based on a proxy set, i.e., a set of initial trees that hold most structures existing in the problem. The proxy set is a set of k architectures sampled from the search space. As discussed in Section V, tree kernels are computed from proxy set and form the dimensions of the latent space. Retrospective studies utilized various strategies for building the proxy set [30]. In the experiments, we used three major strategies for building the proxy set including random subset, top-k subset and diverse subset.

D. PREDICTIVE MODEL

Our objective is to build a predictor $\mathcal{P}$ to well estimate the performance of an architecture before training. Our model takes a network architecture a and an epoch index t, and produces a scalar value $\mathcal{P}(a, t)$ as the prediction of the performance after exactly t epochs. The hypothesis behind this is that the validation accuracy generally changes as training proceeds so we have to be specific about time point of the prediction. This helps us to better model the possible correlations between training samples. We used a three step predictor to select promising models as follows:

Construction. To obtain a small training dataset, we train a small sample of randomly selected architectures S and construct training multivariates in (a_i, p_{ij}, j) where $a_i \in S$ and p_{ij} are in turn the i-th architecture and its validation accuracy at a certain training epoch, plus the epoch number j. Next we train a regression model $\mathcal{P}$ with this dataset to predict the accuracy of other architectures. To capture the complex relations in data we use a Gradient boosting decision trees, (GBDT) as the model $\mathcal{P}$. A GBDT boosts the prediction by leveraging multiple additive trees and thus different views on

TABLE 1. The proposed integer coding for network operations; consisting of 4 parts: operation type (one-hot code), kernel width, kernel height and output to input ratio.

Operation Name	Operation 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
SkipConnection	6	0	0	0
No-Op	7	0	0	0

data: $\hat{y}_{GBDT}(x) = \sum_{r=1}^{R} \hat{y}_{DT_r}(x)$ Where R denote the number of additive trees, and $\hat{y}_{DT_r}$ denotes the predictive model for the r-th tree. Together GBDT extracts R rules to predict the target value for a given feature vector. When a rule is not good enough to differentiate the architectures, its weight is set to a near zero value and thus it is pruned off the tree.

Ranking. The trained predictor $\mathcal{P}$ estimates the accuracy of a large number of architectures sampled from $\mathcal{A} \setminus S$. These architectures are then ranked based on their predicted accuracy and top n architectures are passed to the next step for final evaluation.

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

VI. EXPERIMENTS

We evaluated our approach on two commonly used datasets for NAS, i.e., **NASBench 101** and **NASBench 201**, which show efficiency of our proposed approach compared to the state-of-the art baseline methods in NAS. Although the architectures in NASBench datasets are limited in structure and type; the proposed representation paradigm can potentially encode various type of cells.

A. NASbench 101

NASbench 101 search space is a dataset of about 425'000 pre-built and tested architectures on CIFAR-10. The search space is built upon NASNet principles [5]. The main stem of the architecture is composed of three times repetition 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 'CONV1 × 1', 'CONV3 × 3', 'MAXPOOL1 × 1', plus 'INPUT' and 'OUTPUT' which refer to convolution with 1 × 1 kernel, 3 × 3 kernel, max pooling with 1 × 1 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 have been trained, validated and tested three times and the results have been 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.[1] To better compare the results with [12], [19], [20], the current state of the art predictors, we use their training setup.

B. NASbench 201

The NASbench 201 [31]set consists of approximately 16,000 cell-based architectures. These architectures are trained, validated, and tested on CIFAR-10, CIFAR-100, and ImageNet-16-120. Each architecture is composed of of three segments, each containing N times repetitions of a fully connected cell. Each cell is made up of input and output nodes and 2 internal nodes. A total of 5 permissible operations are defined for internal nodes including 'CONV3 × 3', 'CONV1 × 1', 'AVG3 × 3', 'ZEROIZE', 'SKIP-C'. The lowest errors reported on this set so far (after 500 trials), are 53.22% for testing and 53.15% for validation reported from two different architectures.

C. BENCHMARK METHODS

To evaluate the proposed method, we compared several state-of-the-art baselines with our proposed method as stated below:

- **Random Search**: The simple and fast cell-based search method that combines random search with weight-sharing and is proposed in [32]
- **NAO**: Encodes the architectures into continuous space and uses gradient descent there to optimize the encoded architectures [33].
- **RE**: A primary method based on evolution which competes against RL in large spaces [2].
- **Neural Predictor**: It uses Graph Convolutional Networks to extract features and is proposed in [20].
- **GBDT-NAS**: Proposes a discrete encoding method and uses GBDT as the predictor [12].
- **BANANAS**: Uses path-based encoding and is based on Bayesian optimization [19].
- **Weak NAS Predictor**: Focuses on progressive sampling to improve predictive accuracy. This method is presented in [34]

[1]The highest test accuracy which is reported by Regularized evolution [2] through extensive search is 94.32% but its mean test accuracy and validation accuracy are in turn 94.1% and 95.13%.

- **GATES**: Proposes a new graphical encoding method that models the data converted under a sequence of operations instead of modeling the operations [23].
- **NASGEM**: Is based on graph encoding and clusters the graphs using Weisfeiler-Lehman kernels [24].
- **BONAS**: a sample-based NAS framework which is accelerated using weight-sharing to evaluate multiple related architectures simultaneously. It also applies a GCN predictor as the surrogate model for Bayesian Optimization to select candidate models in each iteration [35].
- **BRP-NAS**: a hardware-aware NAS method with a GCN-based performance predictor. It improves the sample efficiency by considering binary relations of the models and an iterative data selection strategy [36].
- S^2i: The proposed method based on searching in latent space via a predictor.

D. EXPERIMENTAL SETTING

We first describe preparing the NASbench 101 and NASbench 201 sets and the proprietary model settings. We used an Apple M1 processor for all the experiments and we did not exploit GPU. For conventional test and validation experiments our proxy set contains 15 trees and we used 100 randomly sampled architectures as the training set. We use 300 random samples to validate the results and the rest as the testing set. We leverage a Gradient Boosting Decision Tree (GBDT) model with 100 trees and 31 leaves per tree as a predictor. The whole experiment is run for 50 times and the average result is reported. This setup is chosen so that the results are comparable with the other state of the art methods introduced above.

E. RESULTS ON NASbench 101

The first round is the basic test and validation experiment. For the methods we examine, a total of 2000 (1000+1000) samples are taken randomly to make the training and validation set. As we mentioned in previous section, for our method we used 100 training and 300 validation random samples. For GBDT-NAS [12], Weak Predictors [34], BANANAS [19] and BONAS [35], we used the official codes published by the authors. We also had the researchers' verified code for Regularized Evolution [2], Random Search [32] and Neural Predictor [20]. We run each of the mentioned codes for 50 times and averaged the results. For NAO [33], the values reported in [20] and [12] and for BRP-NAS [36] we used both the code and the values reported in [34]. The reported results for the GATES [23] and NASGEM [24] are both from the context of the respective articles. The average validation and testing results are given in Table 2.

As shown in the Table 2, the proposed method has the highest accuracy in validation data. Of course, the WeakPredictor method has the same validation accuracy. Other basic methods have the best accuracy with significant differences in validation data. The Table 3 shows the number of training samples used by the methods with highest test accuracy.

TABLE 2. Validation and test results on NASbench 101. The entries with '-' corresponds to those that the validation accuracy was not reported in the original paper.

Method	Test Acc(%)	Val Acc(%)	Test Regret
Random Search [32]	93.7	94.5	0.62
NAO [33]	93.90	94.1	0.42
RE [2]	93.96	94.7	0.36
Neural Predictor [15]	94.04	95.1	0.28
NAS-GBDT [12]	94.14	94.5	0.18
BANANAS [19]	93.9	94.5	0.42
NASGEM [24]	94.1	—	0.22
GATES [23]	94.0	—	0.32
NPENAS [37]	94.14	94.25	0.18
Weak Predictors [34]	94.23	94.9	0.09
BRP-NAS [36]	94.22	—	0.10
BONAS [35]	94.22	—	0.10
$\mathbf{S^2i}$	94.21	94.9	0.11

As it can clearly be seen in the results, the lowest number of training samples belong to our method.

1) EVALUATION WITH SPARSE DATA

A major challenge of using the predictive method is pessimistic estimation in some areas of the search space. Thus ranking architectures relative to each other, is more important than estimating their exact performance (numerical accuracy). To do so, we compare the performance of our predictive model with GCN, LSTM, MLP and GATES predictive models through calculating Kendall-Tau correlation coefficients. The values for these methods are derived from [23] and reported in Table 4. To evaluate the resistance of the proposed method to the amount of available data, all predictors are trained with different ratios of the training data and the Kendall-Tau coefficient on the accuracy obtained from testing the architectures is represented. We train our predictor primarily with 100 architectures out of 381262 mentioned set. What remains from the training budget of each round, is divided into 10 parts to update the predictor.

As Table 4 shows, the generalization is especially evident when training samples are scarce. For example, when only 190 architectures (0.05% of architectures) are observed by the predictive model, the ranking in our method clearly results in a higher correlation than the other encoders.

2) SAMPLE EFFICIENCY

To observe how the encoding can improve the sample efficiency, we conduct a series of experiments and sample the proxy architectures with different limitations. In each round, we evaluate the resulting model on 10,000 architectures, but our experiments show that these results are also valid on the whole data-set. First we sample only those architectures which their validation accuracy is above a certain threshold. The results of this experiment are reported in Table 5. For each specific threshold, the test is run 3 times and the results are averaged.

The results approve that the quality of proxy trees is effective in solving the final problem. Generally as the accuracy

TABLE 3. Sample efficiency on NASbench 101.The second row indicates the minimum number of architecture-accuracy pairs queried from the dataset to train the predictor.

Method	BONAS [35]	Weak predictor [34]	NPENAS [37]	BRP-NAS [36]	**S²i (Ours)**
Predictor Type (%)	GCN	MLP/GBDT	MLP	GCN	GBDT
Trained Models	1000	1000	150	150	**100**
Avg. Test Accuracy (%)	94.22	94.23	94.14	94.22	**94.21**

TABLE 4. The Kendall-Tau ratio using different encoders on NASbench 101. 90% of the architectures (381262 samples) were used as total training data and the remaining 42362 samples were used for test.

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
Neural Predictor(GCN)	0.61	0.62	0.68	0.69	0.68	0.76	0.77	0.76
GATES	0.76	0.77	0.84	0.85	0.88	0.89	0.90	0.90
S²i	0.82	0.82	0.83	0.84	0.88	0.88	0.91	0.93

TABLE 5. Test result on NASbench 101 with limited sampling. The proxy sets are built with architectures better than a minimum specified accuracy.

Minimum Accuracy(%)	Size of the proxy set					
	10	20	40	60	80	100
Random	93.1	93.5	93.4	93.3	93.5	93.7
>94	94.23	93.68	93.74	94.23	93.65	93.58
>93	93.97	93.95	94.23	94.18	93.94	94.15
>90	94.15	93.68	93.98	94.07	93.85	94.23
>80	93.7	93.82	93.83	94.21	93.6	94.02

threshold of the proxy architectures increase, the average test accuracy of the predictor and its sample efficiency increase. It is clearly seen that this sampling is much efficient than random selection. What is negative about the results is the slight non-monotonic behavior of the results which makes it difficult to decide on the size of proxy set, which may show the model overfits to a specific regions. So next we use our proposed kernel-based measure to remove similar architectures from the high-accuracy proxy set. Therefore, we might select architectures with slightly lower performances instead. We run this test for 10‘000 samples, totally 5 times and average the results.

We can infer from the Table 6 that second test has a more monotone behavior. So, the better approach to sample proxy architectures is to keep diversity among them even if we cast out some highly accurate networks. To train the predictor, $m + k$ architectures are randomly sampled from the space and are fully trained on the task. Then they are sorted according to their performance. The proxy networks are selected iteratively from the top of the list. The architectures with less diversity compared to their subsequent candidate, are ignored until k architectures are selected. The remaining m architectures are used as the training set.

3) VISUALIZATION OF THE ENCODED SPACE

In Figure 4 about 16000 architectures randomly sampled from the space are visualized using t-sne. The color denotes

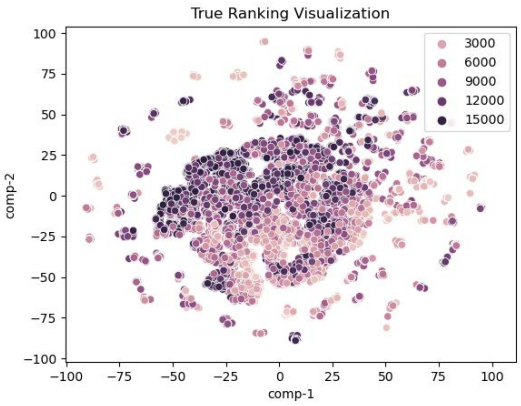


FIGURE 4. The two dimensional visualization of the encoded NASbench 101 architectures. The dots become darker when the accuracy increases.

the architectures‘ rank according to their true performance. Darker points resemble more accurate architectures. The visualization is not exact as the real encoding space is multi-dimensional while it is mapped to two dimensional space. Anyhow it can be seen that architectures with similar performance are mapped to close areas and form small groups. We also move gradually from strong regions to weak regions and vise-versa.

F. RESULTS ON NASbench 201

To test the proposed predictor on a more challenging dataset, we used NASbench 201 and the set of architectures trained on ImageNet-16-120.

1) TEST ON SPARSE DATA

The Kendall-Tau coefficient was also measured on the NASbench 201, the results of which are shown in Table 7.

2) VISUALIZATION OF THE ENCODED SPACE

In Figure 5 the encoded architectures are viusalized using t-sne. The interesting observation is that poorly

TABLE 6. Test result on NASbench 101. The proxy database is built with diverse, accurate architectures.

Sampling method	Size of the proxy set							
	10	15	20	30	40	60	80	100
Best Unvaried (accuracy >93.8%)	93.66	94.23	93.69	94.23	93.94	93.69	93.94	94.01
Best Varied	93.95	94.01	94.01	94.07	94.15	94.21	94.17	94.19

TABLE 7. The Kendall-Tau factor using different encoders on the NASbench 201. 5% of the architectures (7813 samples) were used as training data and the remaining 7812 samples were used for test.

Encoder	Proportions of 7813 training instances (%)							
	0.05%	0.1%	0.5%	1%	5%	10%	50%	100%
MLP	0.09	0.39	0.53	0.82	0.87	0.87	0.8	0.89
LSTM	0.55	0.64	0.72	0.87	0.90	0.85	0.88	0.89
GATES	0.74	0.86	0.88	0.91	0.92	0.87	0.89	0.89
$\mathbf{S^2i}$	0.91	0.91	0.92	0.92	0.94	0.89	0.90	0.90

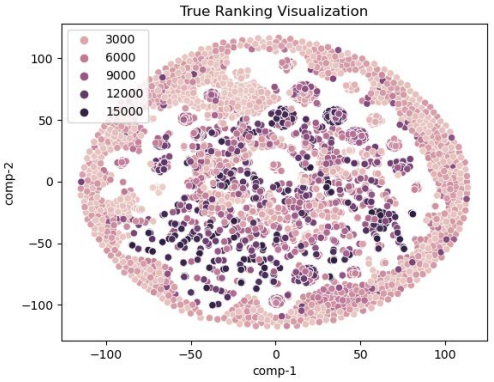


FIGURE 5. The two dimensional visualization of the encoded NASbench 201 architectures. The dots become darker when the accuracy increases.

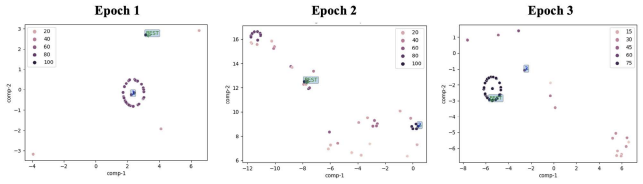


FIGURE 6. The search trajectory after three runs with a trained predictor. The best predicted architecture and the global best is highlighted.

ranked architectures form a colony around the rest of the architectures which are mapped at the center of the space.

3) VISUALIZATION OF THE SEARCH TRAJECTORY

The primary trained predictor, evaluates the search space step by step. Here in each step it samples 1000 new architectures and trains the best 100 ones. The previously evaluated architectures are not selected twice. The search dynamics for three runs is depicted in Figure 6. As it can be seen the predictor soon finds the whereabouts of the best architecture and moves towards it.

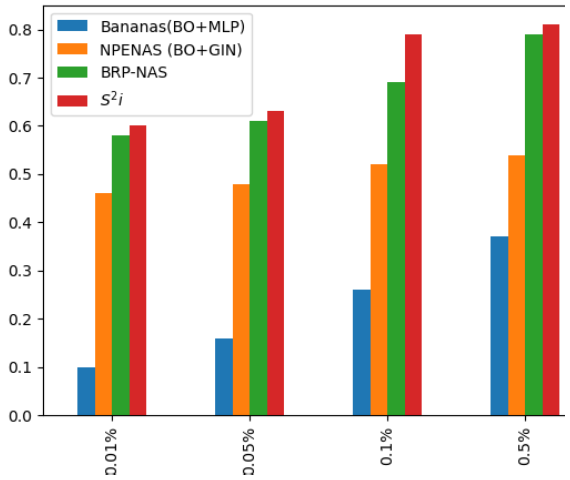


FIGURE 7. The Kendall-Tau ratio related to evaluating ImageNet-16-120 using NASBench-201. The predictors are trained with different ratios of 4000 samples of NASBench-201-CIFAR10 set and tested on NASBench-201-ImageNet.

G. CROSS META-LEARNING DATASET

We further demonstrate the usefulness of our predictor trained on CIFAR-10 by applying it to ImageNet classification. It is revealed that CIFAR accuracy and ImageNet accuracy are strongly correlated [7]. We transfer the best architectures found on CIFAR-10 as proxy trees to the ImageNet problem. We exploit the Kendall-Tau coefficient to assess the quality of the predictor's ranking. To compare the findings to the results in other papers, we conduct experiments under the following settings: On each round, the predictor is primarily trained with 100 samples and updated with the rest of the training budget of the specific round (batch size is 100). The results are summarized in Figure 7. Our predictor achieves slightly better performance BRP-NAS and significantly surpasses NPENAS and Bananas.

VII. CONCLUSION

Surrogate predictive models have shown its effectiveness in addressing the Neural Architecture Search problem where a model attempts to forecast the performance of a neural model ahead of training.

In this paper, we proposed to leverage both structural features of a deep network and per-layer information 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. In particular, we adopt tree

kernel for extracting structural feature and distributional feature learning for non-structural features. Extensive results on two state-of-the-art datasets of NAS task demonstrate the effectiveness of the proposed model especially regarding convergence speed, sample efficiency and performance estimation.

There is also interesting insights focusing on the keyword, *ontology*. Here domain Knowledge can be expressed in the form of an ontology graph that can model the relationship between different types of the same category of operations. Like the relationship and the effect of using different filters, or between Max Pooling and Average Pooling. Therefore, ontology can inject general knowledge into our system. For now, we focused on limited databases. By using the ontology and the knowledge it conveys, the features obtained from other problems or databases can be shared among them. In other words, by gathering this knowledge, it is possible to decide what characteristics the general architecture should have.

Many extension of this work can be exploited. For example, we plan to extend the proxy set sampling method and acquire more effective samples. Further, adapting zero-shot and one-shot learning for effective training of surrogate models can be a promising direction.

DECLARATION OF INTERESTS

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

APPENDIXES

COMPUTING TREE KERNELS

As mentioned in section computing kernels, for the two trees T_1 and T_2 and their set of nodes, N_{T_1} and N_{T_2}, the kernel function is defined as follows:

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} mu\left(\lambda^2 + \sum_{p=1}^{l_m} \Delta_p(c_{n1}, c_{n2})\right) \tag{10}$$

The Δ_p function calculates the number of common sub-trees in exactly p children (from both n_1 and n_2). If aS_1 and bS_2 are two child sequences (a and b are the first children), then:

$$\Delta_p(aS_1, bS_2) = \Delta(a, b) + D_p[|S_1|, |S_2|] \tag{11}$$

Initialize the matrix $D_p(p + 1 \times p + 1)$ with 0 and use the following recursive relation to calculate the components of the matrix:

$$\begin{aligned} D_p[k, l] = {} & \Delta_{p-1}(s_1[1:i], s_2[1:r]) + \lambda D_p[k, l-1] \\ & + \lambda D_p[k-1, l] - \lambda^2 D_p[k-1, l-1] \end{aligned} \tag{12}$$

To avoid the useless calculation of delta function for dissimilar sub-tree roots, the node lists can be sorted to match only the sub-tree roots with similar labels. The delta function checks the equality of two entities. Our entities/nodes consist of the type of assigned operation and its hyperparameters. We define a Delta function as below to consider both:

$$Delta(E_1, E_2) = \left(E_{1_1} = E_{2_1}\right)\left(0.75 + 0.25 \times \left(E_{1_2} = E_{2_2}\right)\right) \tag{13}$$

The E_{i_1} is the operation type which is decisive in matching and E_{i_2} is its most important hyperparameter (like the kernel size). So similar operations with different hyperparameters get a proportional matching score.

REFERENCES

[1] E.-G. Talbi, "Automated design of deep neural networks: A survey and unified taxonomy," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–37, 2021.

[2] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.

[3] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, *arXiv:1611.02167*.

[4] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[5] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8697–8710.

[6] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 1–8.

[7] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–34.

[8] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017, *arXiv:1705.10823*.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through HyperNetworks," 2017, *arXiv:1708.05344*.

[10] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.

[11] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, "Neural architecture search using deep neural networks and Monte Carlo tree search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 9983–9991.

[12] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T.-Y. Liu, "Accuracy prediction with non-neural model for neural architecture search," 2020, *arXiv:2007.04785*.

[13] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[14] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas, and A. C. I. Malossi, "TAPAS: Train-less accuracy predictor for architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 3927–3934.

[15] H. Pham, Y. Melody Guan, B. Zoph, V. Quoc Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *CoRR*, vol. abs/1802.03268, pp. 1–11, Feb. 2018.

[16] B. Deng, J. Yan, and D. Lin, "Peephole: Predicting network performance before training," 2017, *arXiv:1712.03351*.

[17] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2423–2432.

[18] C. White, W. Neiswanger, S. Nolen, and Y. Savani, "A study on encodings for neural architecture search," 2020, *arXiv:2007.04965*.

[19] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian optimization with neural architectures for neural architecture search," 2019, *arXiv:1910.11858*.

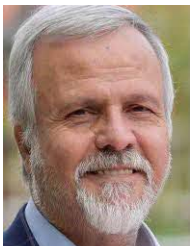
[20] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *Proc. Eur. Conf. Comput. Vis.* Springer, 2020, pp. 660–676.

[21] G. Kyriakides and K. Margaritis, "Evolving graph convolutional networks for neural architecture search," *Neural Comput. Appl.*, vol. 34, no. 2, pp. 899–909, Jan. 2022.

[22] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6861–6871.

[23] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based NAS," in *Computer Vision—ECCV 2020*. Glasgow, U.K.: Springer, Aug. 2020, pp. 189–204.
[24] H. Cheng, T. Zhang, S. Li, F. Yan, M. Li, V. Chandra, H. H. Li, and Y. Chen, "NASGEM: Neural architecture search via graph embedding method," *CoRR*, vol. abs/2007.04452, pp. 1–9, Jul. 2020.
[25] P. Mahé and J.-P. Vert, "Graph kernels based on tree patterns for molecules," *Mach. Learn.*, vol. 75, no. 1, pp. 3–35, Apr. 2009.
[26] A. Moschitti, "Efficient convolution kernels for dependency and constituent syntactic trees," in *Proc. Eur. Conf. Mach. Learn.*, 2006, pp. 318–329.
[27] D. Haussler, "Convolution kernels on discrete structures," Dept. Comput. Sci., Univ. California, Berkeley, CA, USA, Tech. Rep., 1999.
[28] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 550–558.
[29] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114.
[30] A. Moschitti, "Making tree kernels practical for natural language learning," in *Proc. 11th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2006, pp. 113–120.
[31] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," 2020, *arXiv:2001.00326*.
[32] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," 2019, *arXiv:1902.07638*.
[33] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," 2018, *arXiv:1808.07233*.
[34] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan, "Stronger NAS with weaker predictors," 2021, *arXiv:2102.10490*.
[35] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with BONAS," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1808–1819.
[36] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "BRP-NAS: Prediction-based NAS using GCNs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 10480–10490.
[37] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang, "NPENAS: Neural predictor guided evolution for neural architecture search," 2020, *arXiv:2003.12857*.

SAEEDEH ESLAMI is currently pursuing the Ph.D. degree with the Engineering School, Ferdowsi University, Mashhad, Iran. Her thesis is concerned with the optimal encoding of deep architectures, data scarcity and search space pruning in neural architecture search.



REZA MONSEFI (Member, IEEE) received the B.Sc. degree (Hons.) in electrical and electronic engineering from Manchester University, in 1978, Manchester, U.K., and the M.Sc. degree in control engineering and the Ph.D. degree in data communication and supervisory control from Salford University, Manchester, in 1981 and 1983, respectively. He is a Chartered Engineer and has been a Senior Lecturer of artificial intelligence and machine learning with the Computer Engineering Department, Engineering Faculty, Ferdowsi University of Mashhad, Mashhad, Iran, since 1991. His research interests include computer networks, wireless sensor networks, machine learning, and intelligent systems.

MOHAMMAD AKBARI received the Ph.D. degree from the NUS Graduate School for Integrative Sciences and Engineering, National University of Singapore. His research has been published in several major academic venues, including SIGIR, WSDM, and ICMR. His research interests include data mining and machine learning using large-scale datasets, with an emphasis on their applications in health informatics and social informatics.