Combinatorial Routing for Neural Trees

Jiahao Li , Ruichu Cai* , Yuguang Yan
School of Computer Science, Guangdong University of Technology, Guangzhou, China
{jiahaoli.gdut, cairuichu}@gmail.com, ygyan@gdut.edu.cn

Abstract

Neural trees benefit from the high-level representation of neural networks and the interpretability of decision trees. Therefore, the existing works on neural trees perform outstandingly on various tasks such as architecture search. However, these works require every router to provide only one successor for each sample, causing the predictions to be dominated by the elite branch and its derivative architectures. To break this branch dominance, we propose the combinatorial routing neural tree method, termed CombRo. Unlike the previous methods employing unicast routing, CombRo performs multicast schema in each iteration, allowing the features to be routed to any combination of successors at every non-leaf. The weights of each architecture are then evaluated accordingly. We update the weights by training the routing subnetwork, and the architecture with the top weight is selected in the final step. We compare CombRo with the existing algorithms on 3 public image datasets, demonstrating its superior performance in terms of accuracy. Visualization results further validate the effectiveness of the multicast routing schema. Code is available at https://github.com/JiahaoLi-gdut/CombRo.

1 Introduction

Neural Networks (NNs), as de facto standard frameworks, have spread across various fields [Krizhevsky *et al.*, 2012; Graves *et al.*, 2013; Vaswani *et al.*, 2017]. However, while these fields enjoy the convenience of minimal feature engineering [Bengio, 2013; Zeiler and Fergus, 2014], they face a dilemma of how to choose the appropriate network architecture. Meanwhile, Decision Trees (DTs), have demonstrated efficacy in diverse domains [Aboah, 2021; Phu *et al.*, 2017; Xue and Zhao, 2008], benefiting from transparent decision-making architectures [Perner, 2011; Mahbooba *et al.*, 2021] but constrained by limited representation capabilities.

Neural Trees (NTs), as a category of models that integrate NNs and DTs, are expected to achieve higher performance than NNs and DTs. Several pilot studies have made significant efforts. For example, SDT [Irsoy *et al.*, 2012], BT [Irsoy *et al.*, 2014], and RDT [Léon and Denoyer, 2016] use a soft decision tree (SDT) to approximate a network, so that the tree can capture the implicit decisions in the network. However, the lack of representation learning along paths causes better interpretability to be achieved at the expense of performance. DNDF [Kontschieder *et al.*, 2015] converts the scalar output of each neuron in the fully connected layer, into a probability for selecting successors at the neuron-related non-leaf node. Unfortunately, its promising accuracy requires a costly separate optimization. Besides these works, many others also focus on NTs. They, as well as those mentioned earlier, can be categorized based on their hybrid level [Li *et al.*, 2022].

Recently, some works on NTs attempted to mine tree-based architecture. ANT [Tanno *et al.*, 2019] constructs architecture layer by layer according to router decisions. SeBow [Chen *et al.*, 2021] builds architecture by preserving high-contributing nodes within a predefined mother network. Mother network, constraining architecture space, is a significant hallmark that differentiates SeBow from ANT. Both works adopt the same three types of primitive modules: routers, transformers, and solvers. Routers are used for path selection, transformers for feature transformation, and solvers for yielding the class distribution. These modules, as shown on the left side of Figure 1, are embedded in the mother network and further assembled into an architecture by a routing algorithm. However, the utilization of unicast routing usually leads to NTs yielding an architecture where the part close to the root degenerates into a dominant single-branch network. Relevant experimental evidence can be found in Figure 2 of ANT's supplementary materials and Figure 1 of SeBow's supplementary materials.

The aforementioned degeneration results from two factors. (1) First, NTs allow all samples to walk along the same path and reach the same node. If this node is a leaf, then NTs completely degenerate into single-branch NNs. If it is a non-leaf, the deeper the node, the more severe the degeneration of NTs. (2) Second, unicast routing causes competition among different branches, which is reflected in the division of probability measures by these branches. Since the total measure is constant, an increased measure for one branch leads to a reduced measure for some other branches. Thus, it is always ineffective to combine multiple branches in accordance with a fixed measure threshold for multi-node selection. A low threshold

*Corresponding Author

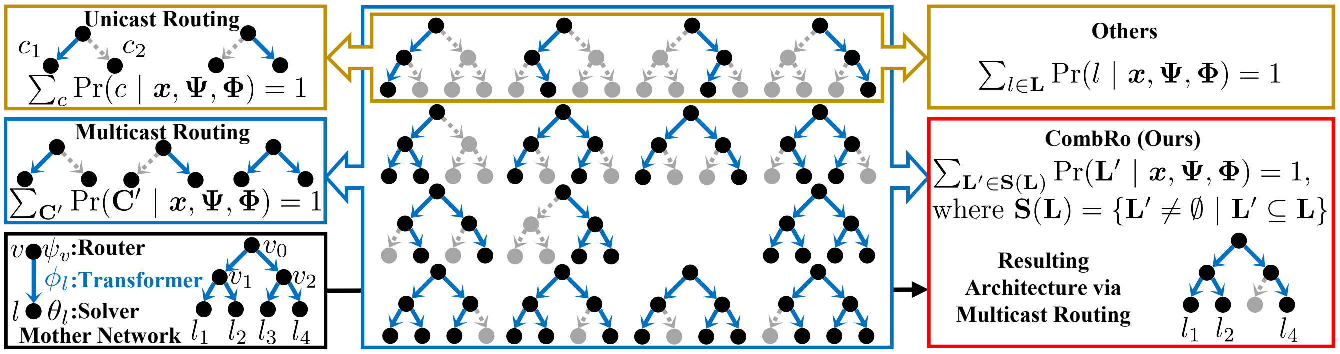
Unicast Routing
c_1
c_2
$\sum_c \Pr(c \mid \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = 1$
Multicast Routing
$\sum_{\mathbf{C}'} \Pr(\mathbf{C}' \mid \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = 1$
v ψ_v:Router
ϕ_l:Transformer
l θ_l:Solver
Mother Network
v_0 v_1 v_2 l_1 l_2 l_3 l_4
Others
$\sum_{l \in \mathbf{L}} \Pr(l \mid \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = 1$
CombRo (Ours)
$\sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \Pr(\mathbf{L}' \mid \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = 1,$
where $\mathbf{S}(\mathbf{L}) = \{\mathbf{L}' \neq \emptyset \mid \mathbf{L}' \subseteq \mathbf{L}\}$
Resulting Architecture via Multicast Routing
l_1 l_2 l_4

Figure 1: Comparison of Existing Routings and Our Combinatorial Routing: Architectural Spaces and Capacity Measures

often yields heavyweight architectures, while a high one may result in architectures lacking forks in their shallow sections. This observation provides a plausible explanation for why SeBow [Chen *et al.*, 2021] employing a high threshold is prone to adopting architectures lacking forks in their upstream sections, which wastes a large number of branches.

In this paper, we propose a novel approach, termed Combinatorial Routing (CombRo), to extract architecture from DNNs using multicast routing (see Figure 1). That is to say, the goal of this work is to mine integrally formed NTs from DNNs. To achieve the goal, we first perform forward propagation on the mother network to obtain all the decisions of non-leaves and all the predictions of leaves. Furthermore, we assemble all architectures according to the structure of the mother network and obtain their probabilities according to the decisions of their corresponding non-leaves. Using these probabilities to weight their corresponding architectures, we can train the importance of different architectures by maximizing the ensemble performance. Finally, the architecture with the highest probability is selected and retrained. In this way, we achieve an architecture with better performance, in accordance with our comparative experiments.

2 Related Works

Our work primarily involves neural decision trees and neural architecture search. Neural decision trees, a super-class of our model, are notable for their internal node-based routing of data features, as detailed in the latest survey [Li *et al.*, 2022]. Neural architecture search, the task executed by our proposed model, seeks to automate network design, reducing dependence on human expertise [Ren *et al.*, 2021].

Neural Decision Trees (NDTs) blend the benefits of DTs with NNs. Early versions of NDTs [Stromberg *et al.*, 1991; Jordan and Jacobs, 1994] learn the input-output patterns of NNs for low-dimensional tabular data. Recent developments [Kontschieder *et al.*, 2015; Ji *et al.*, 2020] employ Multi-Layer Perceptrons (MLPs) or convolution layers to achieve more sophisticated routing. A prime example is Yang's DNDT [Yang *et al.*, 2018], which employs MLPs to discern multiple boundaries in the scalar feature value range, moving beyond the binary tree limitation. However, DNDT's routing efficacy is somewhat hindered as it directly uses transformer-sourced data for routing. To address this, Tanno's ANT [Tanno *et al.*, 2019] and Chen's SeBow [Chen *et al.*, 2021] adopt a bypass NN-based router. However, these models still operate on a unicast routing mechanism, pointing towards a need for more advanced combinatorial routing solutions to address branch competition effectively. Note that we omit the comparison with NBDT [Wan *et al.*, 2020] and TAO [Zharmagambetov and Carreira-Perpinán, 2021] because they require a class hierarchy and a more complex backbone.

Neural Architecture Search (NAS) aims to automate network architecture design with minimal human intervention [Ren *et al.*, 2021]. Most NAS works [Real *et al.*, 2019; Chen *et al.*, 2019; Wu *et al.*, 2019] adopt a cell or block-based search strategy due to its compatibility with various tasks. However, the discrete nature of early NAS methodologies leads to a reliance on Evolutionary Algorithms (EA) [Real *et al.*, 2017], Reinforcement Learning (RL) [Zoph and Le, 2016], and Bayesian optimization [Kandasamy *et al.*, 2018]. To overcome these limitations, Differentiable Architecture Search (DAS) [Shin *et al.*, 2018] was introduced, transforming the neural architecture space into a continuously differentiable domain. This advancement enables gradient-based optimization methods to perform more efficient search. Our combinatorial routing-based search also benefits from DAS.

3 Notations and Preliminaries

In this section, we first introduce foundational concepts related to neural trees, which serve as the basis for the advanced topics discussed later. Additionally, in preparation for understanding combinatorial routing strategies, we elucidate the principles of installing routers at non-leaf nodes in neural tree structures. This groundwork is essential for comprehending the intricate combinations and permutations that play a crucial role in optimizing neural trees.

3.1 Notations

Let $\mathbf{T} = (v, \mathbf{C})$ represent the tree, where v denotes the parent of m children $\mathbf{C} = \{c_1, \ldots, c_m\}$. The k-th child c_k is denoted as either a leaf l or a tree with the same definition as $\mathbf{T}$. According to this construction, $\mathbf{T}$ is such a layer-wise and multi-branch structure, in which each leaf l corresponds to a unique path $\mathbf{P}(l)$ toward it from root r. Given several paths like $\mathbf{P}(l)$, another type of tree $\mathbf{T}'$ can be constructed as a sub-graph of $\mathbf{T}$ by merging the duplicated nodes depth-wise between paths. Because of the uniqueness of $\mathbf{P}(l)$, $\mathbf{T}'$ is determined only by its leaves $\mathbf{L}' \subseteq \mathbf{L}$, where $\mathbf{L}$ represents all the leaves of $\mathbf{T}$. Suppose that $\mathbf{L}$ contains n leaves, then $\mathbf{T}'$ has $2^n - 1$ different potential architectures and further varies with $\mathbf{L}'$. Since $\mathbf{L}'$ changes with the successors chosen by non-leaves $\mathbf{V} = \{v\}$, we can introduce a sample $\boldsymbol{x}$ to intervene these selections and then the form of $\mathbf{L}'$. In detail, given $\boldsymbol{x}$, let a non-leaf v choose some children $\mathbf{C}' \subseteq \mathbf{C}$ with probability

$$\Pr(\mathbf{C}' \mid z_v(\boldsymbol{x})), \quad \text{s.t.} \quad \sum_{\mathbf{C}' \in \mathbf{S}(\mathbf{C})} \Pr(\mathbf{C}' \mid z_v(\boldsymbol{x})) = 1,$$

where $\mathbf{S}(\mathbf{C})$ could be any non-empty subset of the power set of $\mathbf{C}$. If $\mathbf{S}(\mathbf{C}) = \{\{c\} \mid c \in \mathbf{C}\}$, successor selections are based on unicast schema. If $\mathbf{S}(\mathbf{C}) = \{\mathbf{C}' \mid \mathbf{C}' \subseteq \mathbf{C} \wedge |\mathbf{C}'| \geq 1\}$, selections are based on multicast schema. Both schemas are also illustrated on the left side of Figure 1. As for the operator z_v, it is defined as a sequence of transformations from r to v, i.e. $z_v = \psi_v \circ \phi_v \circ \cdots \circ \phi_r$. The operators ψ_v and ϕ_v are the router and the transformer of v respectively, as shown in the bottom left corner of Figure 1. Obviously, we can write

$$\Pr(\mathbf{C}' \mid v, \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = \Pr(\mathbf{C}' \mid z_v(\boldsymbol{x})),$$

where $\boldsymbol{\Psi}$ and $\boldsymbol{\Phi}$ contain all routers and all transformers, respectively. Moreover, each subset $\mathbf{C}' \subseteq \mathbf{C}$ can be identified as a state $\rho(\mathbf{C}') = \sum_k^m 2^{k-1} I(c_k \in \mathbf{C}')$ of the event variable $\varrho(v)$ at v. Therefore, we have

$$\Pr(\varrho(v) = \rho(\mathbf{C}') \mid v, \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = \Pr(\mathbf{C}' \mid v, \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}). \quad (1)$$

Similarly, given $\boldsymbol{\Phi}$ and all solvers $\boldsymbol{\Theta}$, a terminal node l predicts the sample $\boldsymbol{x}$ as the label y with probability

$$\Pr(y \mid l, \boldsymbol{x}, \boldsymbol{\Theta}, \boldsymbol{\Phi}) = \Pr(y \mid z_l(\boldsymbol{x})), \quad (2)$$

where z_l is a sequence of transformations from r to l, i.e. $z_l = \theta_l \circ \phi_l \circ \cdots \circ \phi_r$. The operator $\theta_l \in \boldsymbol{\Theta}$ is the solver of l, as shown in the bottom left corner of Figure 1.

3.2 Installing Routers

As mentioned above, non-leaves serve not only as computation nodes to transform features but also as routers to make decisions. In line with existing methodologies, while retaining the feature transformation capability of these nodes, we also install a router for each non-leaf by embedding a bypass network that outputs a probability vector. Different from unicast routers, our routers are compatible with the number of combinations of children instead of the number of children. If non-leaf v has m children, then the router of v generates

$$z_v(\boldsymbol{x}) = \left[z_v^{(1)}(\boldsymbol{x}), \ldots, z_v^{(2^m-1)}(\boldsymbol{x})\right],$$

where $z_v^{(\varrho)}(\boldsymbol{x})$ is an affinity metric expressing the suitability to handle $\boldsymbol{x}$ on the successors $\mathbf{C}' \in \{\mathbf{C}' \subseteq \mathbf{C} \mid \rho(\mathbf{C}') = \varrho\}$ of v. In the light of these metrics, the router can exploit the Gumbel-Max trick [Gumbel, 1948; Maddison *et al.*, 2014] to make a discrete decision drawn from a categorical distribution with probabilities $z_v(\boldsymbol{x})$:

$$h_v(\boldsymbol{x}) = \text{one\_hot}\left(\arg\max_{\varrho}\left(z_v^{(\varrho)}(\boldsymbol{x}) + \epsilon_\varrho\right)\right),$$

Here $h_v(\cdot)$ is a one-hot vector with the same size as $z_v(\cdot)$. $\epsilon = [\epsilon_1, \ldots, \epsilon_{2^m-1}]$ is a vector in which all the elements are i.i.d samples drawn from Gumbel distribution $(0, 1)$. The introduction of ϵ aims at avoiding routers always selecting the element with the highest probability. To be compatible with the backward propagation, we approximate argmax through the softmax function [Jang *et al.*, 2016]:

$$\begin{aligned}\tilde{h}_v(\boldsymbol{x}) &= \frac{\exp\left((z_v(\boldsymbol{x}) + \epsilon)/\tau\right)}{\sum_{\varrho=1}^{2^m-1} \exp\left(\left(z_v^{(\varrho)}(\boldsymbol{x}) + \epsilon_\varrho\right)/\tau\right)} \\ &= \left[\Pr(\varrho \mid v, \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi})\right]_{\varrho \in \{1, \ldots, 2^m-1\}},\end{aligned} \quad (3)$$

where τ is a temperature controlling the sharpness of target distribution. On the right-hand side of the second equal sign, $\tilde{h}_v(\boldsymbol{x})$ is designated as the basis for decision-making.

4 Combinatorial Routing

In this section, we introduce our proposed CombRo, which breaks one-successor pattern of unicast routing by allocating a non-zero probability to multi-successor events, to mitigate the emergence and overfitting of dominant branch. To display all the details of CombRo, we first answer how to acquire the probability of an architectures after expanding the successor selection space. Then, we describe how to optimize our objective function for extracting the architecture with the top performance.

4.1 Acquiring the Probability of Architecture

Significance

Unlike existing approaches, CombRo searches for various architectures instead of various paths. More exactly, CombRo adopts multi-successor pattern to send data features throughout the architecture with a single probability, while others use one-successor pattern to route a sample through different paths with different probabilities. Consequently, the former, compared with the latter, can search for a well-defined architecture. However, such an architecture complicates the computation of its probability. To explain this point, let us consider such a tree-based architecture that contains some leaves $\mathbf{L}' \subseteq \mathbf{L}$. By practice, previous methods back-trace a specific path $\mathbf{P}(l)$ for each leaf $l \in \mathbf{L}'$ to calculate its arrival probability $\Pr(\{l\}|\boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi})$, and then weight the leaves $\mathbf{L}'$ to evaluate ensemble performance. However, it is meaningless to do this for the architecture where all leaves are interdependent, since this type of architecture forms a unified entity in accordance with the bijection between architecture space and leaf combination space. In other words, conventional practice can not calculate the probability of an integrally-formed architecture based on the probabilities of multiple paths. Thus, it is necessary to design an algorithm to acquire the probability of an architecture.

Implementation

Mathematically, we compute $\Pr(\mathbf{L}' \mid \mathbf{X}, \boldsymbol{\Psi}, \boldsymbol{\Phi})$ as the probability of an architecture using its equivalent form derived from Equation 1:

$$\prod_{v \in \mathbf{T}' \setminus \mathbf{L}'} \Pr\left(\varrho(v) = \sum_{k=1}^{m(v)} 2^{k-1} R(c_k(v), \mathbf{L}') \,\middle|\, v, \mathbf{X}, \boldsymbol{\Psi}, \boldsymbol{\Phi}\right), \quad (4)$$

where $R(c, \mathbf{L}') = I(\exists\, l \in \mathbf{L}',\ c \to l)$ represents the reachability between c and $\mathbf{L}'$. To eliminate reachability computation, we design our bottom-up algorithm to collect probabilities during traversing architecture.

With the help of Formula 4, the algorithm we want should receive a non-empty subset $\mathbf{L}' \subseteq \mathbf{L}$ as the architecture $\mathbf{T}'$ with the leaves $\mathbf{L}'$ and then output a column vector in which each element denotes the probability of a specific sample being passed throughout $\mathbf{T}'$. In addition, this algorithm requires all non-leaves to provide an exclusive probability for all routing events, not just one-successor events. Now, with the conception laid before, we can design our algorithm as follows:

1. To begin with, all the predecessors of $\mathbf{L}'$ are deduplicated as $\mathbf{V}(\mathbf{L}') = \{v(l) \neq \text{Null} \mid l \in \mathbf{L}'\}$, in which $v(l)$ is the predecessor of l. To facilitate subsequent steps, for each $v \in \mathbf{V}(\mathbf{L}')$, the successor set $\mathbf{\Lambda}(v) = \{l \in \mathbf{L}' \mid v(l) = v\}$ of v is created during the deduplication. In detail, $\mathbf{V}(\mathbf{L}')$ is an empty set at first, and then for each $l \in \mathbf{L}'$, of which $v(l) \neq \text{Null}$, if $v(l) \notin \mathbf{V}(\mathbf{L}')$, $\mathbf{V}(\mathbf{L}') = \mathbf{V}(\mathbf{L}') \cup \{v(l)\}$ and $\mathbf{\Lambda}(v(l)) = \{l\}$, otherwise $\mathbf{\Lambda}(v(l)) = \mathbf{\Lambda}(v(l)) \cup \{l\}$.

2. Then, for each predecessor $v \in \mathbf{V}(\mathbf{L}')$, the successor index set $\mathbf{K}(v) = \{k \in \{1, \ldots, m(v)\} \mid c_k(v) \in \mathbf{\Lambda}(v)\}$ of v is yielded, where $c_k(v)$ is the k-th child of v, and $m(v)$ represents the number of children at v. According to the definition of $\mathbf{K}(v)$, $\varrho(v) = \rho(\mathbf{\Lambda}(v)) = \sum_{k \in \mathbf{K}(v)} 2^{k-1}$ stands for such an event that v selects $\mathbf{\Lambda}(v)$ as its successors. Moreover, v and $\varrho(v)$ are packed together as a pair $(v, \varrho(v))$ and stored in an external container $\boldsymbol{\varrho}$.

3. After replacing $\mathbf{L}'$ with $\mathbf{V}(\mathbf{L}')$, Step 1,2, and 3 are repeated sequentially until $\mathbf{V}(\mathbf{V}(\ldots\mathbf{V}(\mathbf{V}(\mathbf{L}'))\ldots))$ is an empty set. Here, all variables can be released before the next iteration starts, except for $\mathbf{L}'$ and $\boldsymbol{\varrho}$.

4. Finally, $\boldsymbol{\varrho}$ is equal to $\boldsymbol{\varrho}(\mathbf{L}') = \{(v, \varrho(v)) \mid v \in \mathbf{T}' \setminus \mathbf{L}'\}$ and denotes the probability expression of $\mathbf{T}'$. Therefore, $\Pr(\mathbf{L}' \mid \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi}) = \prod_{(v,\varrho) \in \boldsymbol{\varrho}(\mathbf{L}')} \Pr(\varrho \mid v, \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi})$ is acquired as the probability of $\mathbf{T}'$ given $\boldsymbol{x}$, $\mathbf{\Psi}$ and $\mathbf{\Phi}$, where $\Pr(\varrho \mid v, \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi})$ denotes the probability of ϱ at v given $\boldsymbol{x}$, $\mathbf{\Psi}$ and $\mathbf{\Phi}$. Moreover, if $\boldsymbol{x}$ is replaced with an input set $\mathbf{X}$, then $\Pr(\varrho \mid v, \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi}) = [\Pr(\varrho \mid v, \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi})]^{\top}_{\boldsymbol{x} \in \mathbf{X}}$ and $\Pr(\mathbf{L}' \mid \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi}) = [\Pr(\mathbf{L}' \mid \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi})]^{\top}_{\boldsymbol{x} \in \mathbf{X}}$.

To boost the algorithm, for each non-empty $\mathbf{L}' \subseteq \mathbf{L}$, the expression $\boldsymbol{\varrho}(\mathbf{L}')$ of $\mathbf{L}'$ is stored for the future use. The reason why doing so can accelerate the algorithm is that the tree-based mother network $\mathbf{T}$ remains constant and $\boldsymbol{\varrho}(\mathbf{L}')$ depends only on $\mathbf{T}$ and $\mathbf{L}'$. In other words, only those probability variables change. Thus, when calling the algorithm again, only Step 4 needs to be executed.

4.2 Optimizing Objective Function

Probabilistic Model and Inference

The original input set $\mathbf{X}$ spreads throughout the entire architecture $\mathbf{T}'$ according to router decisions, and progressively changes its own representations until it reaches all the terminals $\mathbf{L}'$. At these terminals, a set of solvers operates jointly to predict different labels $\mathbf{\Upsilon}$ with a stochastic matrix:

$$\begin{aligned}
&\Pr(\mathbf{\Upsilon} \mid \mathbf{X}, \mathbf{\Theta}, \mathbf{\Psi}, \mathbf{\Phi}) = [\Pr(y \mid \mathbf{X}, \mathbf{\Theta}, \mathbf{\Psi}, \mathbf{\Phi})]_{y \in \mathbf{\Upsilon}} \\
&= \sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \Pr(\mathbf{L}' \mid \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi}) \diamond \underbrace{[\Pr(y \mid \mathbf{L}', \mathbf{X}, \mathbf{\Theta}, \mathbf{\Phi})]_{y \in \mathbf{\Upsilon}}} \\
&= \sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \Pr(\mathbf{L}' \mid \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi}) \diamond \quad \Pr(\mathbf{\Upsilon} \mid \mathbf{L}', \mathbf{X}, \mathbf{\Theta}, \mathbf{\Phi}),
\end{aligned} \tag{5}$$

where $\diamond$ performs a Hadamard product of a vector with every column of a matrix. $\mathbf{S}(\mathbf{L}) = \{\mathbf{L}' \neq \emptyset \mid \mathbf{L}' \subseteq \mathbf{L}\}$ is the space of terminal combination, i.e. the space of architecture. The definition of $\Pr(\mathbf{L}' \mid \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi})$ has been introduced in Equation 4. And $\Pr(y \mid \mathbf{L}', \mathbf{X}, \mathbf{\Theta}, \mathbf{\Phi}) = [\Pr(y \mid \mathbf{L}', \boldsymbol{x}, \mathbf{\Theta}, \mathbf{\Phi})]^{\top}_{\boldsymbol{x} \in \mathbf{X}}$ is a vector where each element has the following definition:

$$\Pr(y \mid \mathbf{L}', \boldsymbol{x}, \mathbf{\Theta}, \mathbf{\Phi}) = g_y\left(\{z_l(\boldsymbol{x})\}_{l \in \mathbf{L}'}\right), \tag{6}$$

where z_l has been defined in Equation 2. The operator g_y outputs the probability of $\boldsymbol{x}$ being predicted as y based on several results of leaves.

Minimizing Prediction Error

To predict as accurately as possible, CombRo optimizes the ensemble performance on $\mathbf{S}(\mathbf{L})$ by maximizing a probability vector:

$$\begin{aligned}
&\Pr(\mathbf{Y}|\mathbf{X},\mathbf{\Theta},\mathbf{\Psi},\mathbf{\Phi}) = [\Pr(y_i|\boldsymbol{x}_i,\mathbf{\Theta},\mathbf{\Psi},\mathbf{\Phi})]^{\top}_{i \in \{1,\ldots,|\mathbf{X}|\}} \\
&= \sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \Pr(\mathbf{L}'|\mathbf{X},\mathbf{\Psi},\mathbf{\Phi}) \circ \underbrace{[\Pr(y_i|\mathbf{L}',\boldsymbol{x}_i,\mathbf{\Theta},\mathbf{\Phi})]^{\top}_{i \in \{1,\ldots,|\mathbf{X}|\}}} \\
&= \sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \Pr(\mathbf{L}'|\mathbf{X},\mathbf{\Psi},\mathbf{\Phi}) \circ \qquad \Pr(\mathbf{Y} \mid \mathbf{L}', \mathbf{X}, \mathbf{\Theta}, \mathbf{\Phi}),
\end{aligned} \tag{7}$$

where the target set $\mathbf{Y} = \{y_1, \ldots, y_{|\mathbf{X}|}\}$ corresponds to the input set $\mathbf{X}$ and $\Pr(y_i \mid \mathbf{L}', \boldsymbol{x}_i, \mathbf{\Theta}, \mathbf{\Phi})$ is defined in Equation 6. By practice, the maximization of Equation 7 is converted to the minimization of *Negative Log-Likelihood* (NLL) loss:

$$\mathcal{J}_1 = -\log \Pr(\mathbf{Y} \mid \mathbf{X}, \mathbf{\Theta}, \mathbf{\Psi}, \mathbf{\Phi}). \tag{8}$$

Decreasing Architecture Uncertainty

To reduce the ambiguity of architecture selection, CombRo considers the following stochastic matrix:

$$\mathbf{M} = [\mathbf{M}(\boldsymbol{x})]^{\top}_{\boldsymbol{x} \in \mathbf{X}} = [\Pr(\mathbf{L}' \mid \mathbf{X}, \mathbf{\Psi}, \mathbf{\Phi})]_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})},$$

where every column corresponds to a particular architecture and every row $\mathbf{M}(\boldsymbol{x}) = [\Pr(\mathbf{L}' \mid \boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi})]_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})}$ denotes the discrete distribution of architecture given the sample $\boldsymbol{x}$. The probability mass of $\mathbf{M}(\boldsymbol{x})$ should be concentrated onto some architecture. Therefore, CombRo imitates the form of unbiased variance to maximize the sharpness of $\mathbf{M}(\boldsymbol{x})$:

$$\zeta(\boldsymbol{x}) = \frac{1}{|\mathbf{S}(\mathbf{L})|-1} \sum_{\mathbf{L}' \in \mathbf{S}(\mathbf{L})} \left(\Pr(\mathbf{L}'|\boldsymbol{x}, \mathbf{\Psi}, \mathbf{\Phi}) - \frac{1}{|\mathbf{S}(\mathbf{L})|} \right)^2.$$

The above equation reaches its maximum value $1/|\mathbf{S}(\mathbf{L})|$ iff $\mathbf{M}(\boldsymbol{x})$ contains element 1. To eliminate the impact of $\mathbf{S}(\mathbf{L})$ and maintain the same scale as probability, the range of $\zeta(\boldsymbol{x})$ is adjusted to $[0, 1]$ by multiplying $|\mathbf{S}(\mathbf{L})|$. At last, CombRo minimizes the smoothness of $\mathbf{M}$:

$$\mathcal{J}_2 = -\log\left(|\mathbf{S}(\mathbf{L})|\zeta(\mathbf{X})\right). \tag{9}$$

Objective Function and Architecture Selection

CombRo is conducted in three phases: *search phase*, *selection phase* and *retraining phase*. In the search phase, CombRo optimizes the following objective function:

$$\mathcal{J} = \mathcal{J}_1 + \gamma \mathcal{J}_2, \tag{10}$$

where $\mathcal{J}_1$ and $\mathcal{J}_2$ have been defined in Equation 8 and 9, respectively. The hyper-parameter γ controls the fuzziness of architecture selection. When the number of iterations reaches the maximum number of iterations, the search phase ends and then the selection phase begins. During the selection phase, CombRo evaluates which architecture is most popular among the samples. That is to say, CombRo selects the architecture with the following leaves:

$$\mathbf{L}^* = \underset{\mathbf{L}' \in \mathbf{S}(\mathbf{L})}{\arg\max} \sum_{\boldsymbol{x} \in \mathbf{X}} \Pr(\mathbf{L}' \mid \boldsymbol{x}, \boldsymbol{\Psi}, \boldsymbol{\Phi}). \tag{11}$$

Once the most popular architecture is determined according to Equation 11, CombRo proceeds to the retraining phase. In detail, CombRo first fixes $\Pr(\mathbf{L}' \mid \mathbf{X}, \boldsymbol{\Psi}, \boldsymbol{\Phi}) = I(\mathbf{L}' = \mathbf{L}^*)$ for each $\mathbf{L}' \in \mathbf{S}(\mathbf{L})$, and then optimizes Equation 8.

5 Experiments

5.1 Experimental Settings

Datasets

To evaluate the effectiveness of CombRo, three classification benchmarking datasets, including CIFAR10 [Krizhevsky *et al.*, 2009], CIFAR100 [Krizhevsky *et al.*, 2009] and tiny-ImageNet [Le and Yang, 2015], are adopted. These datasets vary in size and distribution. CIFAR10 and CIFAR100 each consists of 60k images in resolution 32×32, but the latter has a more complex distribution than the former. Tiny-ImageNet dataset comprises a collection of 110k images from 200 different classes with 64×64 resolution.

Mother Network

Unlike SeBow [Chen *et al.*, 2021], we use the same architecture space in different dataset experiments. This space is generated by a mother network with the shape of a perfect binary tree. The mother network has 4 layers, each of which contains $\{1, 2, 4, 8\}$ nodes, respectively. The reason we set the depth to 4 is that 4-layer mother network can generate sufficiently low-dimensional features and a sufficiently large architecture space. To comprehensively evaluate the proposed method, we devise three variants of CombRo, symbolized by CombRo-A, CombRo-B and CombRo-C, with varying capacities. Details of these models are summarized in Table 1.

Training Settings

We train CombRo using SGD optimizer with the initial learning rate of 0.1. After 30 epochs, the learning rate is decayed by half every 20 epochs until the training stops at the 100th epoch. In addition, we set the batch size to 128, the weight decay to 10^{-4}, the Nesterov momentum to 0.9, and the Gumbel softmax temperature τ to 0.5. The hyperparameter γ for regularization term is set to 0 for the first 80 iterations, and then set to 0.1 for the subsequent 20 iterations. We retrain the resulting architecture on the training sets using the same training setting in the search phase but set the weight decay to be 5×10^{-4}. The operator g_y calculates the average of the class distributions of several branches.

Model	Router	Transformer	Solver
CombRo-A	2× Conv3-48 + GAP + LC	2× Conv3-96 + BN + ReLU + MaxPool	GAP + LC
CombRo-B	2× Conv3-72 + GAP + LC	2× Conv3-144 + BN + ReLU + MaxPool	GAP + LC
CombRo-C	2× Conv3-128 + GAP + LC	2× Conv3-256 + BN + ReLU + MaxPool	GAP + LC

Table 1: Settings of the primitive modules. ”Conv3-48” denotes a 2D convolution with 48 filters of spatial size 3×3. ”GAP”, ”LC”, ”BN” and ”MaxPool” stand for global-average-pooling, linear classifier, batch normalization and max-pooling operations, respectively.

Inference Schemes

Since CombRo evaluates architectures rather than branches, CombRo supports multi-architecture and single-architecture inferences. Multi-architecture inference applies Equation 5 to make an expected prediction in the sub-architecture space yielded by the mother architecture obtained during the selection phase. Although this probabilistic inference is cumbersome, we still test its performance. Single-architecture inference performs deterministic multi-path inference on the selected architecture. Unlike probabilistic multi-path inference, single-architecture inference does not require routing guidance, as the derived architecture is elected by the majority of samples in the dataset. As for the probabilistic single-path inference performed by existing algorithms, it is not considered in our model. For a fair comparison, CombRo still performs probabilistic single-path inference, by distributing the probability of selecting a child combination back to each child in the combination. Note that probabilistic inference requires the optimization of routers during the retraining phase.

5.2 Benchmark Comparisons

We compare the performance of CombRo against a range of models: (1) classic CNNs, representative works of human engineering, including MobileNet [Howard *et al.*, 2017], GoogleNet [Szegedy *et al.*, 2015], VGG-13 [Simonyan and Zisserman, 2014] and ResNet-18 [He *et al.*, 2016]; (2) multi-branch networks, a super set of neural trees widely applied in multi-task learning, including Routing network [Rosenbaum *et al.*, 2017], Learning to Branch [Guo *et al.*, 2020], Cross-stitch [Misra *et al.*, 2016] and MPPS [Shi *et al.*, 2023]; (3) neural decision trees, a technical path in which CombRo is situated, including Adaptive Neural Trees (ANT) [Tanno *et al.*, 2019], Self-born Wiring (SeBow) [Chen *et al.*, 2021], Neual Decision Tree Towards Neural Graph (NDT) [Xiao, 2017], Deep Neural Decision Forests (DNDF) [Kontschieder *et al.*, 2015], Conditional CNN [Ioannou *et al.*, 2016].

The results in accuracy and resulting architecture size (the number of parameters) on three datasets are presented in Table 2, 3 and 4, respectively. To reduce randomness, all experimental results are obtained from the average of three independent experiments with the same setting. Although we

Method	**Params.**	**Accuracy(%)**
MobileNet	2.2M	85.90±0.23
VGG-13	28.3M	92.51±0.15
ResNet-18	11.2M	93.01±0.19
LearnToBranch#	3.5M	91.98±0.57
Max-Cut DT	N/A	*34.90*
Compact BT	N/A	*48.56*
gcForest	N/A	*61.78 / 61.78*
Conditional CNN	*> 0.5M*	*< 90.00*
*ANT-C**	*0.7M / 0.5M*	*90.69 / 90.66*
*ANT-B**	*0.9M / 0.6M*	*90.85 / 90.82*
*ANT-A**	*1.4M / 1.0M*	*91.69 / 91.68*
ANT-A(ensemble)*	*8.7M / 7.4M*	*92.29 / 92.21*
SeBow-A	*1.0M / 0.7M*	*93.45±0.12 / 93.41*
SeBow-B	*2.7M / 1.6M*	*94.00±0.18 / 93.93*
SeBow-C	*5.8M / 4.6M*	*94.33±0.14 / 94.24*
CombRo-A	1.1M / 0.7M	93.45±0.11 / 93.42
CombRo-B	2.5M / 1.6M	94.03±0.14 / 93.93
CombRo-C	6.8M / 4.6M	**94.48**±0.17 / **94.25**

Table 2: Performance comparison on CIFAR-10. Italic fonts mean that the results are provided by the original paper. Underlined numbers represent the inference results on a single path. The method names with * omit the text related to the dataset, i.e. ANT-{A,B,C}* corresponds to ANT-CIFAR10-{A,B,C} called in the original paper, respectively. LearnToBranch# omits some texts and has the full name LearnToBranch-Deep-Wide. N/A stands for not applicable.

Method	**Params.**	**Accuracy(%)**
MobileNet	2.4M	53.91±0.32
VGG-13	28.7M	72.70±0.42
ResNet-18	11.2M	72.19±0.23
Cross Stitch	N/A	*53.0*
Routing network	N/A	*60.50±0.75*
MPPS	N/A	*71.06*
LearnToBranch#	6.7M	72.04±0.23
Max-Cut DT	N/A	*12.40*
NDT	*14.1M*	*15.48*
DNDF	> 11.2M	67.18
ANT-C*	4.2M / 4.2M	65.81±0.12 / 65.71
SeBow-B	*1.9M / 1.5M*	*71.79±0.23 / 71.59*
SeBow-C	*4.2M / 4.2M*	*74.59±0.33 / 74.59*
CombRo-B	3.8M / 1.6M	72.04±0.21 / 71.77
CombRo-C	5.7M / 4.6M	**75.67**±0.14 / **74.61**

Table 3: Performance comparison on CIFAR-100.

Method	**Params.**	**Accuracy(%)**
MobileNet	2.5M	46.12±0.73
GoogleNet	6.8M	48.85±0.52
VGG-13	28.7M	56.10±0.57
ResNet-18	11.2M	55.33±0.75
DNDF	> 11.2M	44.56
SeBow-C	*8.4M / 4.8M*	*58.77±0.39 / 58.43±0.45*
CombRo-C	8.5M / 4.8M	**60.87**±0.21 / **59.09**±0.25

Table 4: Performance comparison on tiny-ImageNet.

primarily focus on the results of architectures containing several paths, we also provide the results of some models under the single-path schema, for comprehensive comparison.

Based on the experimental results, the following conclusions can be drawn: (1) On three image datasets, CombRo, with a more appropriate model size, achieves superior performance compared to other methods. Due to multicast routing, CombRo has a slightly increased number of parameters compared to some methods, but this is to achieve better generalization. Moreover, traditional architectures require a large number of parameters to achieve competitive performance. For example, ResNet-18 reaches 93.01% with 11.2M parameters on CIFAR10. (2) The accuracy comparison results can be summarized as CombRo-A < CombRo-B < CombRo-C, which indicates that the more parameters available for the model to use, the stronger its performance. (3) In all experiments, the performance of single-path inference is not significantly different from that of multi-path inference, suggesting that the lower bound of an architecture's performance is dependent on the performance of its internal branches.

5.3 Ablation Study

Resulting Architecture versus Comparable Others

To validate the effectiveness of the architectures extracted by CombRo, we randomly examine several architectures and select some competitive ones for comparison. From the results shown in Table 5, we can make the following observations: (1) Architectures derived by CombRo can achieve optimal performance with a moderate number of parameters (see the gray rows). (2) Architectures with a large number of parameters do not necessarily exhibit superior performance (e.g. the cyan row). (3) The architectures obtained vary for different datasets and experiment settings (see the gray rows).

Deterministic Retraining and Probabilistic Retraining

In this part, we evaluate the impacts of deterministic retraining and probabilistic retraining on the predictive capabilities of the resulting architectures. In detail, we perform deterministic single-architecture inference after deterministic retraining and probabilistic multi-architecture inference after probabilistic retraining. Both results are listed in Table 6 and convey that probabilistic retraining reduces the predictive performance of the architecture. This is because probabilistic multi-architecture retraining and inference reintroduce architectures that are less favorable to the dataset.

Retraining versus Fine-tuning

After the search and selection phases, we obtain an architecture containing trained parameters. Here, we make research on whether the parameters within the architecture should be retrained. Based on the results provided in Table 7, we find that retrained models achieve higher accuracy than fine-tuned models. This is because the parameters obtained during the search phase cause the model to develop a dependency on the pruned branches.

5.4 Interpretability

To effectively understand the dataset's preference for different architectures, we collect statistical data on the preferences

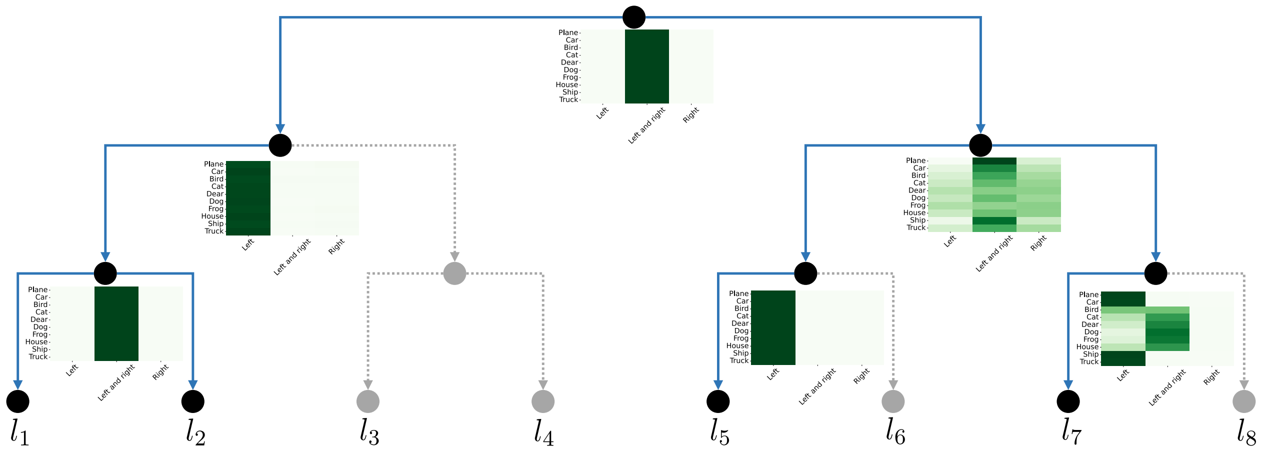

Figure 2: Visualization of preferences for successor combinations among different categories at each internal node. In the heat map, each row and each column represent a single category and a specific successor combination, respectively. The blue solid lines and the gray dashed lines point to the selected (black) nodes and unselected (gray) nodes, respectively.

Dataset	Method	Architecture	Params.	Acc.(%)
CIFAR10	CombRo-B	$\{l_3, l_6, l_7, l_8\}$	2.5M	**94.03**
	Random	$\{l_2, l_3, l_5, l_7, l_8\}$	3.5M	93.60
	Random	$\{l_1, l_2, l_3, l_4, l_5, l_6\}$	3.8M	93.28
	Random	$\{l_4, l_5, l_6, l_7, l_8\}$	3.3M	92.84
	Random	$\{l_3, l_4, l_7\}$	2.0M	92.19
	CombRo-C	$\{l_1, l_2, l_5, l_7\}$	6.8M	**94.48**
	Random	$\{l_4, l_7, l_8\}$	6.5M	94.18
	Random	$\{l_1, l_3, l_5, l_7\}$	8.9M	94.13
	Random	$\{l_1, l_5, l_6, l_7, l_8\}$	10.5M	93.73
	Random	$\{l_6, l_7\}$	4.7M	93.69
CIFAR100	CombRo-B	$\{l_2, l_4, l_5, l_6, l_8\}$	3.8M	**72.04**
	Random	$\{l_4, l_5, l_6, l_7, l_8\}$	3.6M	71.77
	Random	$\{l_1, l_3, l_6, l_8\}$	3.2M	71.65
	Random	$\{l_4, l_6, l_7, l_8\}$	2.5M	70.57
	Random	$\{l_1, l_2, l_3, l_8\}$	3.0M	69.65
	CombRo-C	$\{l_1, l_2, l_4\}$	5.7M	**75.67**
	Random	$\{l_2, l_5, l_6, l_7, l_8\}$	10.9M	74.53
	Random	$\{l_5, l_6, l_7\}$	6.6M	73.55
	Random	$\{l_5, l_6, l_7, l_8\}$	8.3M	73.54
	Random	$\{l_4, l_5, l_6, l_7\}$	9.2M	73.38

Table 5: Ablation study on network architecture. "Architecture" represents the combination of leaves.

Dataset	Method	Probabilistic	Deterministic	Params.
CIFAR10	CombRo-B	92.98±0.16	**94.03**±0.14	2.5M
	CombRo-C	93.15±0.21	**94.48**±0.17	6.8M
CIFAR100	CombRo-B	70.22±0.28	**72.04**±0.21	3.8M
	CombRo-C	73.42±0.34	**75.67**±0.14	5.7M

Table 6: Performance of CombRo with probabilistic retraining or deterministic retraining in the third phase.

Dataset	Method	Fine-tuning	Retraining	Params.
CIFAR10	CombRo-B	92.81±0.13	**94.03**±0.14	2.5M
	CombRo-C	93.17±0.21	**94.48**±0.17	6.8M
CIFAR100	CombRo-B	70.56±0.29	**72.04**±0.21	3.8M
	CombRo-C	72.94±0.24	**75.67**±0.14	5.7M

Table 7: Performance of CombRo with retraining or fine-tuning in the third phase.

of different categories for successor combinations at each internal node. The visualization of these data helps us gain a clearer insight into the formation of the resulting architecture. To facilitate visualization, we perform statistical analysis on the preference behaviors of the CIFAR10 dataset, which has fewer categories, under the settings of CombRo-C.

The results, shown in Figure 2, indicate that the preference of different categories for node combinations is consistent, except for the second node in the second layer and the fourth node in the third layer. As for the second node in the second layer, most categories tend to choose both successor nodes. In the case of the fourth node in the third layer, categories belonging to the transportation category show a preference for the left child node, while other categories tend to choose both successor nodes. However, this does not affect the overall dataset's tendency to favor the left child node.

6 Conclusion

In this paper, we introduce a novel method named CombRo, aimed at extracting an effective architecture from a tree-based mother network. Unlike previous approaches that use unicast routing and thus face the risk of falling into the local optimum caused by the dominant branch, CombRo uses multicast routing to alleviate the emergence of the dominant branch. Since multicast routing provides a learnable weight individually for each architecture, CombRo can weigh different architectures, train them ensemble, and excavate the architecture with the top weight. Experimental results demonstrate that the derived architecture produces performances even on par with those of DNNs on large-scale datasets. In the future, we will explore the potential of CombRo in tasks beyond architecture search.

Ethical Statement

There are no ethical issues.

Acknowledgements

This research was supported in part by National Key R&D Program of China (2021ZD0111501), National Science Fund for Excellent Young Scholars (62122022), National Natural Science Foundation of China (62206064, 62206061), Guangzhou Basic and Applied Basic Research Foundation (2024A04J4384, 2023A04J1700), Major key project of PCL (PCL2021A12), Guangdong Basic and Applied Basic Research Foundation (2023B1515120020, 2024A1515011901), and the Jihua laboratory scientific project (X210101UZ210).

References

[Aboah, 2021] Armstrong Aboah. A vision-based system for traffic anomaly detection using deep learning and decision trees. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4207–4212, 2021.

[Bengio, 2013] Yoshua Bengio. Deep learning of representations: Looking forward. In *International conference on statistical language and speech processing*, pages 1–37. Springer, 2013.

[Chen *et al.*, 2019] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1294–1303, 2019.

[Chen *et al.*, 2021] Ying Chen, Feng Mao, Jie Song, Xinchao Wang, Huiqiong Wang, and Mingli Song. Self-born wiring for neural trees. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5047–5056, 2021.

[Graves *et al.*, 2013] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[Gumbel, 1948] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office, 1948.

[Guo *et al.*, 2020] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *International conference on machine learning*, pages 3854–3863. PMLR, 2020.

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

[Howard *et al.*, 2017] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[Ioannou *et al.*, 2016] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[Irsoy *et al.*, 2012] Ozan Irsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pages 1819–1822. IEEE, 2012.

[Irsoy *et al.*, 2014] Ozan Irsoy, Olcay Taner Yildiz, and Ethem Alpaydin. Budding trees. In *2014 22nd international conference on pattern recognition*, pages 3582–3587. IEEE, 2014.

[Jang *et al.*, 2016] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[Ji *et al.*, 2020] Ruyi Ji, Longyin Wen, Libo Zhang, Dawei Du, Yanjun Wu, Chen Zhao, Xianglong Liu, and Feiyue Huang. Attention convolutional binary neural tree for fine-grained visual categorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10468–10477, 2020.

[Jordan and Jacobs, 1994] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994.

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

[Kontschieder *et al.*, 2015] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Master's thesis, University of Tront*, 2009.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[Le and Yang, 2015] Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7(7):3, 2015.

[Léon and Denoyer, 2016] Aurélia Léon and Ludovic Denoyer. Policy-gradient methods for decision trees. In *ESANN*, 2016.

[Li *et al.*, 2022] Haoling Li, Jie Song, Mengqi Xue, Haofei Zhang, Jingwen Ye, Lechao Cheng, and Mingli Song. A survey of neural trees. *arXiv preprint arXiv:2209.03415*, 2022.

[Maddison *et al.*, 2014] Chris J Maddison, Daniel Tarlow, and Tom Minka. A* sampling. *Advances in neural information processing systems*, 27, 2014.

[Mahbooba *et al.*, 2021] Basim Mahbooba, Mohan Timilsina, Radhya Sahal, and Martin Serrano. Explainable artificial intelligence (xai) to enhance trust management in intrusion detection systems using decision tree model. *Complexity*, 2021:1–11, 2021.

[Misra *et al.*, 2016] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003, 2016.

[Perner, 2011] Petra Perner. How to interpret decision trees? In *Industrial Conference on Data Mining*, pages 40–55. Springer, 2011.

[Phu *et al.*, 2017] Vo Ngoc Phu, Vo Thi Ngoc Tran, Vo Thi Ngoc Chau, Nguyen Duy Dat, and Khanh Ly Doan Duy. A decision tree using id3 algorithm for english semantic analysis. *International Journal of Speech Technology*, 20(3):593–613, 2017.

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

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

[Ren *et al.*, 2021] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys (CSUR)*, 54(4):1–34, 2021.

[Rosenbaum *et al.*, 2017] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *arXiv preprint arXiv:1711.01239*, 2017.

[Shi *et al.*, 2023] Haosen Shi, Shen Ren, Tianwei Zhang, and Sinno Jialin Pan. Deep multitask learning with progressive parameter sharing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19924–19935, 2023.

[Shin *et al.*, 2018] Richard Shin, Charles Packer, and Dawn Song. Differentiable neural network architecture search. In *Proceedings of the Workshop on Neural Architecture Search at International Conference on Learning Representations*, 2018.

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

[Stromberg *et al.*, 1991] J-E Stromberg, Jalel Zrida, and Alf Isaksson. Neural trees-using neural nets in a tree classifier structure. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, pages 137–140. IEEE Computer Society, 1991.

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

[Tanno *et al.*, 2019] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR, 2019.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wan *et al.*, 2020] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: neural-backed decision trees. *arXiv:2004.00221*, 2020.

[Wu *et al.*, 2019] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10734–10742, 2019.

[Xiao, 2017] Han Xiao. Ndt: neual decision tree towards fully functioned neural graph. *arXiv preprint arXiv:1712.05934*, 2017.

[Xue and Zhao, 2008] Jian Xue and Yunxin Zhao. Random forests of phonetic decision trees for acoustic modeling in conversational speech recognition. *IEEE transactions on audio, speech, and language processing*, 16(3):519–528, 2008.

[Yang *et al.*, 2018] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

[Zeiler and Fergus, 2014] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[Zharmagambetov and Carreira-Perpinán, 2021] Arman Zharmagambetov and Miguel A Carreira-Perpinán. A simple, effective way to improve neural net classification: Ensembling unit activations with a sparse oblique decision tree. In *2021 ICIP*, pages 369–373. IEEE, 2021.

[Zoph and Le, 2016] Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2016.