IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 2, 2021, accepted May 20, 2021, date of publication June 21, 2021, date of current version June 29, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3090918

Gradient Descent Effects on Differential Neural Architecture Search: A Survey

SANTANU SANTRA[1], **JUN-WEI HSIEH**[2], **AND CHI-FANG LIN**[1]
[1]Department of Computer Science and Engineering, Yuan Ze University, Taoyuan 32003, Taiwan
[2]College of Artificial Intelligence and Green Energy, National Chiao Tung University, Hsinchu 30010, Taiwan

Corresponding author: Jun-Wei Hsieh (jwhsieh@nctu.edu.tw)

This work was supported by the Ministry of Science and Technology (MOST), Taiwan, under Grant 109-2221-E-009-116-MY3.

ABSTRACT Gradient Descent, an effective way to search for the local minimum of a function, can minimize training and validation loss of neural architectures and also be incited in an appropriate order to decrease the searching cost of neural architecture search. In recent trends, the neural architecture search (NAS) is enormously used to construct an automatic architecture for a specific task. Mostly well-performed neural architecture search methods have adopted reinforcement learning, evolutionary algorithms, or gradient descent algorithms to find the best-performing candidate architecture. Among these methods, gradient descent-based architecture search approaches outperform all other methods in terms of efficiency, simplicity, computational cost, and validation error. In view of this, an in-depth survey is necessary to cover the usefulness of gradient descent method and how this can benefit neural architecture search. We begin our survey with basic concepts of neural architecture search, gradient descent, and their unique properties. Our survey then delves into the impact of gradient descent method on NAS and explores the effect of gradient descent in the search process to generate the candidate architecture. At the same time, our survey reviews mostly used gradient-based search approaches in NAS. Finally, we provide the current research challenges and open problems in the NAS-based approaches, which need to be addressed in future research.

INDEX TERMS Gradient descent, neural architecture search, reinforcement learning, evolutionary algorithm, back-propagation.

I. INTRODUCTION

Automatic machine learning (AutoML) has become a favorable solution for developing deep learning (DL) systems without any human efforts. An AutoML system consists of data preprocessing, feature generation, network model generation, and performance evaluation. Although an AutoML system consists of several stages, the most critical stages are model generation and performance estimation. The model generation stage is either created by machine learning experts or by an automatic design process. The automated architecture design process is known as neural architecture search (NAS). The rapid development and demand of NAS continue to overwhelm the human experts designing architectures in many applications.

Constructing an automatic architecture with different network topologies is first explored in [1]. The pioneering frameworks developed in [2] and [3] have attracted much attention, which bring many exciting ideas for NAS with high-performance outputs. Unfortunately, most NAS approaches require many GPU days and memory, which make the NAS approaches fatally hindered. Hence, advanced approaches that ensure low memory, low computing resources, and power requirements over neural architecture search are obligatory.

Apart from satisfying low memory and computing resources requirements of searching processes, NAS approaches must include some features, such as scalability, efficiency, reliability, and flexibility. Candidate architecture searches in NAS can be performed by reinforcement learning (RL), evolutionary algorithm (EA), gradient-based (GB), or random search (RS) approaches. At present, the gradient-based NAS approach [4]–[8] is considered one of the better candidates for architecture search strategies. Gradient descent has the ability to search for better architectures with a local (or preferably global) minimum to satisfy the requirements, including low memory and computational loading. It is often adopted in back-propagation to repeatedly

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

TABLE 1. Comparative study of different survey papers. ✓and × are used to indicated the covered topic or not covered, respectively.

Works	Context of NAS	Search Space	Search Strategy	Evaluation Strategy	Challenges and solutions
[9]	✓	✓	✓	✓	x
[13]	✓	✓	✓	✓	x
[10]	✓	x	✓	x	x
[11]	✓	✓	✓	✓	✓
[14]	✓	x	x	x	✓
[15]	✓	✓	✓	✓	✓

update *weight parameters*, *architecture parameters* or *both* to minimize expected functional loss. Usually, NAS method can be divided into three sequential stages: *search space construction*, *architecture search*, and *performance evaluation*. The *search space construction* stage explores a large set of possible network architectures that can match or outperform expert-designed architectures. The *architecture search stage* explores possible searching techniques to identify the better (or best) architecture from search space. The last stage, *i.e., performance evaluation* calculates predictive performance on some collected datasets. NAS significantly reduces lots of human efforts over traditional deep Convolutional Neural Networks (CNN) designing, either by tuning architecture parameters, weight parameters, or both. As NAS has been recognized as the core technology of neural architecture designing in next-generation, researchers have focused on extending their knowledge to automatic architecture design processes. Along this line, Elsken *et al.* [9] presented a survey on NAS, where they have addressed the elementary ideas of NAS, different search approaches and performance estimation strategies, and future directions of NAS.

A review highlighting reinforcement learning (RL) for NAS is provided by Jaâfra *et al.* [10]. The survey [9] categorized NAS into different classes such as Bayesian reasoning, evolution, reinforcement learning, and gradient search. Moreover, an extensive study of NAS is presented in [11]. Latterly, White *et al.* [12] have provided a survey of NAS for candidate architecture generation. Table 1 shows detailed comparisons among different survey papers [9]–[11], [13]–[15]. In this study, we begin by summarizing the characteristics of different state-of-art (SoTA) NAS approaches and their challenges. We present an in-depth study that explores architecture optimization strategies to generate candidate architectures with good performance and helps readers obtain possible research ideas and further directions, which inspires us to write this survey article.

A. CONTRIBUTIONS

Our study provides an in-depth explanation that directs towards gradient descent-based searching strategy in neural architecture search. Its contents are categorized into two parts: (i) Detailed backgrounds on automatic architecture search for readers, and (ii) Comparisons among several GD-based architecture search strategies, including their usability, efficiency, and stability. Significant contributions in this article are summarized below.

- Our survey explores the basics of NAS, its work principal, and its associated search strategies.
- We discuss the usability and effect of the gradient descent method in NAS strategies.
- Our survey defines the taxonomies for GD-based search and discusses different evaluation strategies to generate better candidate architectures.
- We evaluate different search strategies in terms of their validation errors, their numbers of architecture parameters, and computational costs (GPU days).
- Finally, we summarize different research challenges and discuss their issues in NAS approaches that further researches can be conducted.

B. ORGANIZATION

The remainder of this survey paper is organized as follows. Section II discusses the elementary concept of NAS and the gradient descent technique. Section III provides an overview of various commonly used search spaces. Different searching strategies in NAS to find out a better candidate architecture are explored in Section IV. Section V explores different gradient-descent problems which are occurred during architecture search process. Different tricks to regularize gradient descent method are explored in Section VI. Section VII explores Differentiable Neural Architecture Search (DNAS) techniques. The performance evaluations of various State-of-The-Art (SoTA) GD-based NAS approaches are highlighted in Section VIII. Different research challenges of DNAS are discussed in Section IX. Finally, we summarize and conclude this study in Section X.

II. BACKGROUND

A. OVERVIEW OF NAS

In last few decades, computer vision research attracts much attention to find a well-performed structure that can extract rich features for image or video understanding. Although there are different feature extractors for image and speech recognition, they can achieve only near 70% and 80% accuracy. AutoML has taken the attention of most researchers in the field of computer vision. Instead of manually designing architectures (*e.g.*, Alex-Net [16], VGG-Net [17], Google-Net [18], Res-Net [19], Dense-Net [20]), NAS explores the possibility of discovering unexplored architectures with an automatic algorithm. NAS finds the best performing neural network by exploring all possible candidate architectures that can match or outperform hand-designed architectures. NAS has become an essential step to automate neural architecture design and save a lot of expert efforts for boring trial-and-error routines [3], [21] in various fields. The stick diagram of a NAS with its stages is shown in Figure II-A. The search module extracts each candidate architecture one by one from the search space, and the evaluation module calculates the performance (in terms of desired requirements) of each candidate architecture. During the search loop, the performance is returned

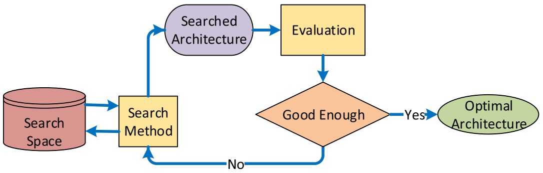

FIGURE 1. Basic building blocks of neural architecture search methods.

to the search module for finding next candidate with better performance.

B. PRELIMINARY OF GRADIENT DESCENT METHOD

Gradient descent (GD)-based optimization approaches use the gradient information and iteratively tune the parameters of architecture, weight, or both to search for the best architecture candidate based on the desire tasks. Let $\mathcal{L}oss_W$ denote the loss function and its gradient be $\nabla_W \mathcal{L}oss(W)$ *w.r.t.* to the parameters $W \in \mathbb{R}^d$. During searching, the loss function $\mathcal{L}oss_W$ is minimized by updating W in the opposite direction of $\nabla_W \mathcal{L}oss(W)$ with a learning rate η. η plays an important role in the optimization loop but is often trapped at a local minimum stationary point.

The GD methods can be divided to two stages, *i.e.*, *parameter updating* and *parameter scaling*. For parameter updating, GD methods can be further categorized to four classes: *vanilla GD*, *stochastic GD*, *momentum GD*, and *adaptive GD*. For the stage of parameter scaling, GD methods can be further divided to two stages *i.e.*, *batch GD* and *mini-batch GD*. *The vanilla GD* based approach calculates the gradient $\nabla_W \mathcal{L}oss(W)$ for each iteration and updates the weight parameters according to Eq.(1) as follows.

$$W_t = W_{(t-1)} - \eta \cdot \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}). \tag{1}$$

It takes small steps towards the direction of the minima by taking gradient of the cost function. *The stochastic GD (SGD) approach* uses shuffle to randomly sample data from training dataset, and in each iteration, the randomly selected single data-point is used for gradient computation and tuning the weight parameters by the following equation.

$$W_t = W_{(t-1)} - \eta \cdot \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}; x^{(i)}; y^{(i)}), \tag{2}$$

where each training sample and its corresponding label is indicated by $x^{(i)}$, and $y^{(i)}$, respectively. The frequent update strategy of SGD can provide a pretty detailed rate of improvement. However, a bigger learning rate may produce wrong gradient updates, results in an inefficient learning path, and slowly decreases loss function. *The momentum GD method* works better and faster than the SGD one by considering some kind of moving average gradient direction to the updating rule as:

$$\begin{aligned} V &= momentum * PreviousUpdate; \\ W_t &= W_{(t-1)} + V - \eta \cdot \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}). \end{aligned} \tag{3}$$

The SGD with momentum can accelerate the gradient vectors in right directions as a result of faster converging. Due to its superior effect in convergence, many SoTA models are trained using it.

The adaptive GD method performs small updates (*i.e.,* low learning rate) for parameters associated with recurring features and performs larger updates (*i.e.,* higher learning rates) for parameters associated with infrequent features; that is,

$$W_t = W_{(t-1)} - \frac{\eta}{\sqrt{G_{t-1} + \epsilon}} \cdot \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}), \tag{4}$$

where G_{t-1} is a diagonal matrix, and each diagonal element is sum of squares of the past gradients to all parameters W, and ϵ is a smoothing term that avoids division by zero.

The SGD method uses only one sample to compute gradient direction and leads to lots of local maxima/minima. *The batch GD* approach computes the gradient using the whole dataset. It is computationally efficient with stable gradient error and great for convex or relatively smooth error manifolds. However, it often leads to an over-fitting model. Thus, in most conditions, the batch GD method uses a mini-batch of several samples instead of the whole dataset to compute the gradient direction. *The mini-batch gradient descent* approach is the combination of both SGD and batch GD methods, where the dataset is divided into many batches, and each batch is used to calculate the gradient errors and update weight parameters (as shown in Eq.5), *i.e.*,

$$W_t = W_{(t-1)} - \eta \cdot \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}; x^{(i:i+B)}; y^{(i:i+B)}), \tag{5}$$

where B represents the mini-batch size of the training samples. Hence it can generate better gradient errors while balancing the robustness like SGD and efficiency likes batch gradient descent.

In GD methods, to make the learning process faster and more stable, the inputs of a model should be re-centered and re-scaled for parameter scaling; if the mean and variance of each layer's input are estimated from the whole dataset, the GD method is categorized as the *batch GD*; otherwise, it is named as the *mini-batch GD*. For the *mini-batch GD* approach, the mean and variance are calculated as follows:

$$\mu_B = \frac{1}{B}\sum_{i=1}^{B} X_i \quad \text{and} \quad \sigma_t^2 = \frac{1}{B}\sum_{i=1}^{B}(X_i - \mu_B)^2, \tag{6}$$

where the mini-batch size B is the whole dataset for the batch GD method, the inputs X_i are then normalized as follows:

$$X_i = \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}. \tag{7}$$

III. SEARCH SPACE

The search space includes all possible candidate architectures generated or evolved with a supernet according to various desired properties. A well-organized search space can reduce the size, simplify the search strategies, and save the search time for a specific task. However, there are some caveats for representing good architecture. *First*, it requires manually defining criteria (sometimes with biases), which often prevent producing good and stable architectures beyond human knowledge. *Second*, it also needs prior knowledge about various types of datasets. Hence, it seldom works well for all unknown domains.

There are two kinds of architecture search structures created for NAS, namely, the *macro* and *micro* architectures (shown in Figure 2). The *macro* architecture is usually used for constructing the topological structure of a neural network, and the *micro* architecture details the operations between nodes (or cells) inside a neural network architecture. The macro structure is represented by a supernet which is evolved by an automata. The micro architecture is often referred to as *cell structure* that are optimized according to different tasks. Due to huge computational loading and search time, NAS algorithms search for either *macro* or *micro* architectures but not necessarily both of them together. In recent trends, most of the NAS approaches consider a fixed *macro* architecture and investigate better *micro* architectures because *micro* architecture search is cheap in computational resource requirements and has much more flexibility in the search space than *macro* architecture.

Some popular search spaces used for vision-based tasks and their evolution are illustrated in Figure 3. A reinforcement learning automata evolve the NAS-RL search space, and each macro cell structure has n layers where each layer is connected to its precursors and forms a densely connected structure. A macro cell architecture from the NAS-RL search space has a maximum $2^{(n-1)(n-2)/2}$ possible connections. A layer in micro cell architecture also consists of other hyperparameters, such as filter numbers and stride parameters. Clearly, the search space is huge, and impossible to find the best architecture with only fewer GPU resources (or days). As to the NASNet search space [21], two repeated types of *convolutional cells* are used to construct a network architecture, *i.e.*, *Normal Cell* and *Reduction Cell*. As shown in Figure 2(a), in the NASNet search space, each cell receives bi-chain style inputs from two precursor layers or the input image. Then, a child architecture is generated by a recurrent neural network (RNN) controller, which repeats five prediction steps to select two possible hidden inputs and their operations to construct the micro structure of each convolutional cell.

DARTS search space adopts the same strategy as NASNet; instead of 13 candidate operations, it only considers 7 candidate operations and a dummy ***zero*** operation, which is only used during micro cell searching process but discarded in the final architecture. The DARTS search space also limits the intermediate hidden nodes to 4, and each hidden node can receive the inputs from any preceding hidden nodes, but only two of them will survive for the intermediate operations; hence the time for finding micro cell structure is reduced. The space for operations searching in DARTS is continuous. MobileNet search space simplifies NAS-RL search space; instead of using dense connections and bi-chain styled connection in layers of macro cell, it uses ***chain-styled*** connections between layers. Each micro cell structure should be chosen from the MobileNet (MB) search space [22].

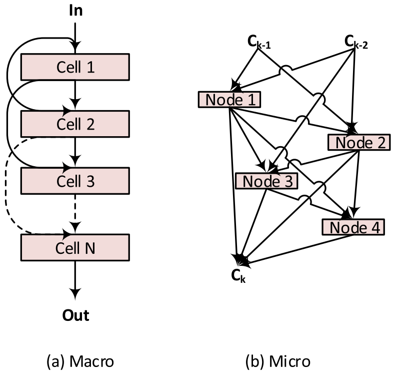

FIGURE 2. Macro and micro cell structures for a search space. (a) Macro structure. (b) Micro structure.

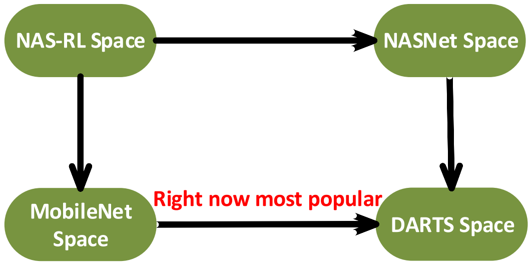

FIGURE 3. Relationships among four most popular search spaces.

IV. STRATEGY FOR ARCHITECTURE SEARCH SPACE GENERATION

A good strategy to generate a compact set of architecture candidates can significantly impact search efficiency and effectiveness of the final candidate architecture. Thus, selecting an appropriate generation strategy can ensure that the generated space is good and small to be fully explored, and the generated architectures are as close as possible to the global best solution. There are three commonly-used strategies for architecture generation, *i.e.*, *evolutionary algorithm*(EA), *reinforcement learning*(RL), and *gradient-based scheme*. In what follows, their details are discussed.

A. EVOLUTIONARY ALGORITHMS

Inspired by a natural evolution process, evolutionary algorithms effectively optimize a function via various mutation, crossover, and selection operators. Due to its effectiveness, the recent development of deep learning also adopts EAs to optimize a neural network in various applications. In EA, the gene contains information on how a problem can be optimized by evolving both the neural architecture and its parameters. However, evolving millions of weights is hugely time-consuming and impractical for a target task. Thus, more recent EA-based approaches solely optimize the neural architectures and then use an SGD-based method to optimize their weights via back-propagation. EA explores large model-architecture search spaces starting with basic initial architectures and evolving them by randomly selecting a parent from a population of models with size S to generate different off-springs to promote the system with good performance. Since '*mutation*' is a local operator and cheap in computation cost, most EAs adopt it to evolve next generations. In addition, EA tries to kill the worst model from the population of S samples to keep the population size constant throughout iterations. This non-aging evolution scheme will result in worse models (not worst) that remain alive in the population for a long time. In [23], Real *et al.* introduced an age property to favor the younger genotypes and designed a new EA-based model, *i.e.,* AmoebaNet which outperforms other RL-based approaches on several open datasets such as CIFA10, CIFA100, and Imagenet. EA tends to evolve a population of architectures that guarantees the diversity of potential results on random uncontrollable mutations. The random uncontrollable mutations make the evolution of EA much slow, and its efficiency no guarantee.

B. REINFORCEMENT LEARNING

The RL-based NAS methods [3], [25] use an agent to generate different candidate architectures by optimizing a reward function. The generation of a new neural architecture can be considered as an agent's action, and the action space is identical to the search space. Usually, the agent is an RNN controller. The agent's reward is based on an estimate of the performance of the trained architecture on training data. Designing a proper reward function is critical and important to guide the optimization process to satisfy the requirements of a task. Different RL approaches differ in how they represent the agent's policy and how they optimize it. For example, in [3], Zoph *et al.* adopts an RNN controller to control convolution operations between two cells. These cells are then stacked in a predefined manner to find candidate architectures. Evolution error of the architecture is used to be the reward and further used to generate another better architecture. Although [3] is the pioneering work of RL-based NAS, its searching time is very expensive and results in poor performance of the finally found architecture. Furthermore, in Efficient NAS (ENAS) [25], a share weight strategy between sub-models is proposed to improve the search time by using an RL-based controller to select various subgraphs from a directed acyclic graph (DAG) so that expected return on a validation set can be maximized. Sharing parameters between sub-models enables ENAS to save search time and provides powerful empirical performance. In [26], Cai *et al.* introduces a Net2Net transform strategy in ENAS, where an RL-based meta controller is used to select each edge operation. The Net2Net strategy modifies different existing architectures through a transformation operation, hence no need to search and train the architecture from scratch.

RL-based NAS uses an RL controller to search an architecture layer by layer and calculates rewards for further searching. Although RL-based NAS approaches can construct a stable architecture for evolution, the RL-based controller needs a huge number of tries to get a positive reward for updating architectures. Hence, RL-based methods are computationally expensive during training. At to EA, its evolution progress relies heavily on random uncontrollable mutations and results in its inefficiency. Thus, in [27], Chen *et al.* integrate the advantages of both of them and ensure the search efficiency to propose a new neural architecture search framework. It introduces a reinforced mutation controller to efficiently explore the search space and benefits from the nature of EA to make the child model inherit most parameters from its parent, so the search for weight parameters becomes more efficient. However, both EA- and RL-based methods are very time-consuming since the search space is huge and discrete.

C. GRADIENT BASED METHOD

To make the search stage of DNAS more efficient, the search space should be continuous so that the SGD method can be applied to find the final architecture directly. DAS [28] converts the discrete network architecture search space into a continuously differentiable one from which gradient optimization can be applied for architecture search. In [4], the framework "Differentiable ARchiTecture Search (DARTS)" converts the combinatorial problem of searching the optimal operations into a continuous and differentiable search space, where a robust cell architecture can be efficiently determined via gradient descent. DARTS [4] is a cell-based neural architecture search approach and works on a Directed Acyclic Graph (DAG) of nodes, where each node represents a set of feature maps.

In DARTS, a supernet is constructed with a set of architectural parameters to form the search space. Two different subnetworks or cells *i.e.* normal cell (operations have stride one) and reduction cell (operations have stride two) iteratively updated through continuous relaxation. An essential issue of DARTS is that easy-to-optimized operators (such as skip-connections and pooling operations) may dominate in early stages, hence hinder the selection of more powerful operations (such as convolutions of large kernels). In [29], P-DARTS enforces a strong prior to limit the number of skip connections within a cell to a pre-determined value by gradually increasing the depth of the network and reducing the candidate operations according to a mixed operation weight. To address this issue, Fair-DARTS [30] is proposed by relaxing the choice of operations, such that each operator has an equal opportunity to balance the architecture strength.

Another issue of NAS optimization is the *embedding* of the evaluation procedure into the search procedure, which is not explicitly performed in the aforementioned frameworks. Various methods are developed to alleviate this problem, *e.g.*, early stopping [6], [31] and progressive optimization [29], with an aim to overcome the issue of discretization gap [6], [32]. The discretization gap means performance of the derived architecture often collapses at the final evaluation stage when discretizing the continuous architecture representation into a discrete one [32]. To address this problem, SGAS [32] proposes a greedy strategy to prevent the problematic skip connections or other weak operators from

TABLE 2. NAS-based methods with search spaces, search strategies, and search types.

Algorithms	Search Space				Search Method			Search type	
	NASNet	DARTS	MobileNet	Others	RL	EA	GB	Micro	Macro
MetaQNN[2]				✓	✓				✓
SMASH[34]				✓		✓			✓
Large-Scale Evolution of ICs 2017[24]				✓		✓			✓
NOS with RL 2017[35]				✓	✓				✓
NASBOT 2018[36]				✓			✓		✓
SNAS[8]		✓					✓	✓	
BlockQNN 2018[37]				✓	✓				✓
DARTS[4]		✓					✓	✓	
Understanding One-Shot Models [38]				✓		✓			✓
ENAS[25]	✓				✓			✓	✓
Progressive NAS[39]	✓				✓			✓	
NASNet [21]	✓				✓			✓	
NAONet [40]					✓	✓		✓	
Proxylessnas[33]		✓					✓	✓	
FBNet[41]			✓				✓		✓
MNASNet[42]			✓						✓
ChamNet[43]				✓					✓
SPNAS[44]			✓				✓		✓
AmoebaNet [23]	✓					✓		✓	
GDAS[45]		✓					✓	✓	
EfficientNet[46]			✓			✓			✓
FairNAS[30]		✓					✓	✓	
PCDARTS[5]		✓					✓	✓	
RDARTS[6]		✓					✓	✓	
BayenNAS[7]		✓					✓	✓	
PDARTS[29]		✓					✓	✓	
XNAS[47]		✓					✓	✓	
DARTS+[31]		✓					✓	✓	
NAT[48]		✓					✓	✓	
SETN[49]		✓					✓	✓	
SPOSNAS[50]				✓			✓		✓
Smooth DARTS[51]			✓				✓	✓	

being often selected. However, potentially good operations might be pruned out as well due to this greedy underestimation. Unlike SNAS, ProxylessNAS [33] constructs an over-parameterized network with all operations in search space and binarizes architecture parameters so that the over parameterized network through gradient-based algorithm can be better trained. DARTS-based methods prune operations on every edge except the one with the largest architecture weight. A significant performance drop will happen in deriving the discrete architecture from the continuous version after projection. Zela *et al*. [6] empirically point out that the stability is highly correlated with the dominant eigenvalue $\lambda_{\max}^{\boldsymbol{A}}$ of the Hessian matrix of the validation loss function of an architecture $\boldsymbol{A}$. Other approaches, *e.g.*, partial channel connection [5], scheduled drop path [21], and regularization of architecture parameters are proposed to address the stability of DARTS. The stability and generalization of DARTS have become an important issue in the research topics of differentiable architecture search.

Table 2 summarizes different types of methods with their search spaces (in Section III), search strategies (in Section IV), and their search types proposed in the literature.

V. GRADIENT DESCENT PROBLEMS

Gradient descent-based approaches can solve an optimization problem with limited computing resources, but they also suffer from several issues. In this section, we explore different issues and their tricks for tuning optimization functions.

A. VANISHING GRADIENT PROBLEM

In deep learning, architecture usually consists of more than hundreds of layers. A lot of weight parameters are then learned and updated with a back-propagation technique via the SGD method for each layer. In this method, each of the weight parameters receives an update proportional to partial derivative of cost function concerning current weight in each iteration during training. In some cases, the gradient value becomes very small or vanishes, thus preventing the weight from further changing its value. This vanishing gradient problem may ultimately stop the neural network from further training. This problem also makes architecture shallow with few layers. In traditional neural network architectures, the activation function is often designed as the Sigmoid function or hyperbolic tangent one whose gradient value ranges (0, 1) or (−1,1). In this case, when the back-propagation technique uses the chain rule to compute the gradient of each weight layer by layer, the gradient value will decrease exponentially and lead to training failure.

To overcome this problem, two solutions can be proposed; one is to change the activation function, and the other is to change the link connections between network layers. For the first one, most researchers chose the Rectified Linear Unit (ReLU) [52] as the activation function. The ReLU function maps x to $max(0, x)$ to avoid vanishing gradient problem, where the gradient of ReLU(x) is one if x is positive. Its variants include Leaky ReLU [53], Swish [54], and Mish [55]. For the second solution to deal with the gradient vanishing problem, in ResNet [56], He *et al.* proves that a skip connection in candidate architecture often passes gradient updates directly to other layers and thus avoids the vanishing gradient problem.

B. EXPLODING GRADIENT PROBLEM

Exploding gradient problem is another issue often happening during the candidate architecture searching process. It is a problem where large error gradients accumulate and result in huge updates to neural network model weights during training and the failure of architecture learning. Gradients are used during searching to update both architecture and weight parameters. The updating process can find a suitable solution if the updates for parameter tuning are properly controlled. Smaller magnitudes of gradients will cause the vanishing gradient problem. Larger magnitudes of gradients may make architecture unstable and cause poor (or over) prediction results. It is useful to know how to identify exploding gradients so that the searching process is corrected to find the expected solution. The standard solution for an explosion gradient is to normalize architecture and weight parameters before propagating back through network via batch normalization. Changing the link connections between network layers via skip connection in candidate architecture is another solution to reduce gradient exploding issues. This connection in candidate architecture often directly passes gradient values to next layer through back-propagation so that the exploding problem is avoided. Similar to the microphone exploding effect, this exploding problem still is an essential issue in recurrent networks.

C. LEARNING RATE

Learning rate is one of the most important hyper-parameters for any gradient descent approach. It is used to control each step size for updating candidate architecture and weight parameters. In general, its default value is initialized with a tiny positive value. A lower learning rate requires more training epochs to optimize a candidate architecture for different tasks, whereas a larger learning rate quickly converges with a suboptimal result with fewer epochs. A larger learning rate often skips stationary points (local minimum) and produces deficient performance in the validation stage, whereas a too small learning rate often gets stuck, as jumping step is tiny. Hence a carefully chosen learning rate is necessary to find optimal parameters for a candidate architecture. In what follows, techniques for gradient updating (learning rate and direction) will be discussed.

VI. GRADIENT DESCENT UPDATING TECHNIQUES

A. MOMENTUM

Gradient descent is an iterative algorithm for optimizing an objective function with its negative gradient. One of its problems is: the search moves downhill towards the minima, but during the progression, it may move in another direction event uphill due to the gradient of some specific (or noisy) points, which slow down the progress of search. It can be improved and accelerated by using momentum from past updates to the search position. More precisely, a fraction of history (the gradient encountered in the previous update) is added to the parameter update equation as shown in Eq.(3). Let γ denote the fraction of momentum term, and V_{t-1} be the history or the gradient encountered in the previous update. Then, the update at the current tth iteration will add the change used at the previous time weighted by the momentum term γ, as follows:

$$V_t = \gamma V_{t-1} + \eta \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)}). \tag{8}$$

Then, from Eq.(8), the set of current parameters W_t is updated by

$$W_t = W_{(t-1)} - V_t. \tag{9}$$

In other words, the momentum rapidly moves towards descent direction whereas slowly moves in opposite direction. As a result, the SGD method with momentum can quickly converge and decrease oscillation.

B. NESTEROV ACCELERATED GRADIENT (NAG)

Gradient descent with momentum can improve the performance of loss function optimization, but still can be further improved via extrapolation. In Eq.(8), we know that we will use our momentum term to move next position. A smarter version of momentum, *i.e.*, Nesterov accelerated gradient (NAG), is to extrapolate the gradient of next point, an approximation to guess where our parameters W are going to be. NAG looks "ahead" to where the parameters will be to calculate the gradient as follows:

$$V_t = \gamma V_{t-1} + \eta \nabla_{W_{(t-1)}} \mathcal{L}oss(W_{(t-1)} - \gamma V_{t-1})). \tag{10}$$

NAG first takes a larger step towards exponentially accumulated gradient, measures gradient, and then updates parameters. This "ahead" update prevents the optimization function from going too fast and increases the performance of the final candidate architecture. The NAG method uses the new version V_t defined in Eq.(10) to update W_t based on Eq.(9).

C. AdaGrad METHOD

In addition, to find a better gradient direction, another important issue is to choose a proper learning rate to control the step size of movement so that the performance of optimization can be improved. The early mentioned SGD methods use a fixed learning rate to optimize the loss function. A small learning rate will result in inefficiency of learning, and a large value will cause an overfitting problem. Thus, the learning rate $\eta_{t,r}$ for the rth parameter $W_{t,r}$ at the tth iteration in the Adaptive gradient (AdaGrad) method is proportional to the inverse of the sum of gradient magnitudes of all parameters during the optimization process; that is,

$$\eta_{t,r} \propto \frac{1}{\sqrt{\sum_{t'=1}^{t}\left(\frac{\partial \mathcal{L}oss(W_{t',r})}{\partial W_{t',r}}\right)^2 + \epsilon}}, \tag{11}$$

where ϵ is used to discard the division zero problem. With a predefined learning rate η, a new adjustable learning rate η'_r can be defined and used to update the parameter movements of a loss function as follows:

$$\eta'_{t,r} = \frac{\eta}{\sqrt{\sum_{t'=1}^{t}\left(\frac{\partial \mathcal{L}oss(W_{t',r})}{\partial W_{t',r}}\right)^2 + \epsilon}}. \tag{12}$$

With Eq.(12), the AdaGrad method updates $W_{t,r}$ by

$$W_{t,r} = W_{t-1,r} - \eta'_{t,r} \times \frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}, \tag{13}$$

where η'_r is adaptively scaled with the inverse of sum of gradients of $W_{t,r}$ from its past iterations.

D. ADADELTA

The AdaGrad method accumulates all past squared gradients as the denominator to adaptively scale the learning rate. Since each added term is positive, the accumulation will lead to an infinitesimally small learning rate $\eta'_{t,r}$; that is, $\eta'_{t,r} \to 0$ when $t \to \infty$. Hence, a modified version of AdaGrad (AdaDelta) is used to avoid this problem. Instead of accumulating all past squared gradients, in AdaDelta method, the sum of gradients is recursively defined as a decaying average of all past squared gradients. Let $G_{t,r}$ denote the average of all past squared gradients for the parameter $W_{t,r}$ as

$$G_{t,r} = \frac{1}{t}\sum_{t'=1}^{t}\left(\frac{\partial \mathcal{L}oss(W_{t',r})}{\partial W_{t',r}}\right)^2. \tag{14}$$

In the AdaDelta method, the running average $G_{t,r}$ at time step t is calculated only depending on the previous average $G_{t-1,r}$ and current gradient:

$$G_{t,r} = \gamma G_{t-1,r} + (1-\gamma)\left(\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}\right)^2, \tag{15}$$

where the ratio γ is often set to around 0.9. Then, in the AdaDelta method, Eq.(13) is written as

$$W_{t,r} = W_{t-1,r} - \frac{\eta}{\sqrt{G_{t,r} + \epsilon}}\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}. \tag{16}$$

The main advantage of AdaDelta is that default learning rate is not necessary for AdaDelta. It is tuned automatically through the average gradient $G_{t,r}$.

E. ADAM

Another approach to calculate the adaptive learning rate for parameter is the Adaptive Moment (AdaM) estimation algorithm. It combines the advantages of momentum (see Eq.(8)) and Adadelta (see Eq.(15)) to adaptively update the parameters. In AdaM, the momentum term makes the average gradient $V_{t,r}$ for $W_{t,r}$ be recursively updated with a balance ratio β_V as

$$V_{t,r} = \beta_V V_{t-1,r} + (1-\beta_V)\nabla_{W_{(t-1),r}}\mathcal{L}oss(W_{(t-1),r}). \tag{17}$$

Similar to Eq.(15), $G_{t,r}$ is recursively updated with another ratio β_G by

$$G_{t,r} = \beta_G G_{t-1,r} + (1-\beta_G)\left(\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}\right)^2. \tag{18}$$

The initial value of $V_{t,r}$ and $G_{t,r}$ are recommended to 0 and result in a bias of moment estimates towards zero. To overcome this issue, they are bias-corrected as follows:

$$\begin{aligned}\hat{V}_{t,r} &= \frac{V_{t,r}}{1-\beta_V},\\ \hat{G}_{t,r} &= \frac{G_{t,r}}{1-\beta_G}.\end{aligned} \tag{19}$$

With Eq.(19), the Adam method updates $W_{t,r}$ as Eq. (20) below:

$$W_{t,r} = W_{t-1,r} - \frac{\eta}{\sqrt{\hat{G}_{t-1,r} + \epsilon}}\hat{V}_{t-1,r}. \tag{20}$$

The Adam approach works well compared to other gradient-based approaches but needs three parameters β_V, β_G, and η for updating the architecture weights.

F. ADAMAX

Adamax is a variant of Adam based on the infinity norm. In Eq.(18), the current gradient term is generalized to the I_p norm as

$$G_{t,r} = \beta_G^p G_{t-1,r} + (1-\beta_G^p)\left|\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}\right|^p. \tag{21}$$

When putting $p \to \infty$, a modified version of Adam, *i.e.*, AdaMax calculates $G_{t,r}$ by the following equation:

$$\begin{aligned}G_{t,r}^{\infty} &= \beta_G^{\infty} G_{t-1,r} + (1-\beta_G^{\infty})\left|\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}\right|^{\infty}\\ &= \max(\beta_G \cdot G_{t-1,r}, \left|\frac{\partial \mathcal{L}oss(W_{t,r})}{\partial W_{t,r}}\right|),\end{aligned} \tag{22}$$

where $G_{t,r}^{\infty}$ denotes the infinity norm of $G_{t,r}$. Then, the AdaMax method updates the parameter $W_{t,r}$ as follows:

$$W_{t,r} = W_{t-1,r} - \frac{\eta}{G_{t-1,r}^{\infty}}\hat{V}_{t-1,r}, \tag{23}$$

where $\hat{V}_{t-1,r}$ is defined in Eq.(19).

G. DATA ADJUSTMENT

Apart from the tricks mentioned above, in the following, we discuss some mostly used tricks that can be combined with any previously mentioned algorithms to enhance the performance of SGD further.

1) SHUFFLING TRAINING DATA

It is common in NAS to shuffle data and normalize it before every epoch for increasing data variance during the architecture searching process. Different shuffling orders of training data can reduce the risk of overfitting for training a candidate architecture. Shuffling a data point can create an ''independent'' change in architecture. Most gradient-based NAS methods have adopted this strategy to improve regularization during architecture searching.

2) BATCH NORMALIZATION

In searching an architecture, both architecture and weight parameters are updated to different extents and scales, which will cause the vanishing or exploding gradient problem and make the learning process unstable. To reduce the risk of searching failure to find a well performing network architecture, parameters are often normalized with zero mean and unit variance. For example, ENAS [25] applies this strategy to prevent gradient explosion during the architecture search process. Suppose the mean and variance of each layer's input are estimated from the whole dataset (see Eq.(6)), the method named as *batch normalization* is inefficient and time-consumed. Another method named as *mini-batch normalization* is adopted to estimate the mean and variance from a small set of randomly selected training data to enhance its efficiency.

3) EARLY STOPPING

Searching processes often face challenges of how long to search for good architecture. An architecture with fewer searching epochs will be under-fitted; with more searching epochs, it is often over-fitted on current training datasets and performs poorly during inference. To make the searching process more efficient and effective, it is better to continually observe the loss on a validation dataset; if the loss does not decrease further, it is better early-stopped. Most gradient descent-based search approaches such as DARTS+ [31], R-DARTS [6], Fair-DARTS [57], and so on, adopt this early-stopping strategy to reduce searching time, and thus significantly improve the performance of candidate architecture searching.

4) GRADIENT NOISE

A dataset with less data often faces difficulties in finding suitable parameters for architectures. Hence, adding tiny values in parameters can make searching process more suitable and decrease generalization errors. For example, Neelakantan *et al.* [58] and Chu *et al.* [59] used a Gaussian distribution to add random noise in parameters for each gradient update to improve data regularization and reduce computational cost for candidate architecture search.

VII. DIFFERENTIABLE NAS

Both RL- and EA-based methods create a discrete search space to find the desired candidate architecture. Since the discrete search space is enormous and evaluating each candidate is both time- and resource-consuming, it is almost impossible to finish the searching task with limited computation devices during a few GPU days or weeks. Another approach is to create a continuous and differentiable search space from which a better candidate architecture can search via a gradient-based optimizer in an end-to-end manner, and thus the search efficiency can be significantly improved. Instead of searching over a discrete set of candidate architectures, the ''DARTS'' framework [4] applies continuous relaxation that converts the categorical choice problem into a continuous and differentiable search space in which two different subnetworks (or cells), *i.e*. normal cell and reduction cell are iteratively updated by gradient descent. There are different issues for a gradient-based method to accelerate the search process and find the desired candidate within a few hours on limited GPU resources: (I) Supernet, (II) Parameter Sharing, and (III) Stability and continuous Relaxation. Details of all the issues are discussed as follows.

A. SUPERNET

The NAS-based approaches require substantial computational resources to find the best candidate architecture for a targeted task. The architecture searching time can be drastically reduced by encoding a search space into an over-parameterized neural network (i.e., supernet) with a weight-sharing strategy. The NAS methods like [25], [50], [60] construct or adopt a controller, which can sample the architecture to train the supernet and, through a heuristic search method, generates the best performing architecture for a specific task from the discrete search space. Instead of training each architecture separately, the differentiable NAS builds a supernet that assembles all the architectures as its subnetworks to form a continuous space from which the candidate architecture can be found via a decent gradient method. The supernet is an over-parameterized network that can be categorized into two classes according to how the architectures are modeled and elaborated; that is, parameter-based and path-based. For parameterized architectures, a real-value distribution is introduced to categorize the architectures and then their weights are jointly learned such as DARTS [4], FBNet [41] and MdeNAS [61]. Figure 4 shows an example of a supernet constructed by DARTS. Figure 4(a) is a searched architecture, and (b) is an expansion of normal cells. Figure 4(c) represents hidden nodes and their connections inside a micro cell. The micro cell structure is searched from the supernet and stacked to obtain the desired architecture. After training the supernet, the optimal architecture can be found by sampling from the categorical distribution via a gradient method. Besides accuracy, another very important issue

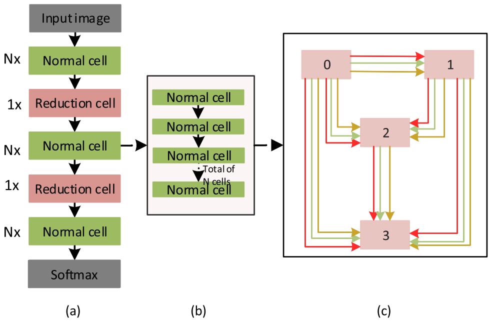

FIGURE 4. DARTS based architecture searching process from the search space.

is integrating some hardware constraints, *e.g.*, FLOPS and latency, for designing efficient neural network architectures for hardware. However, since these constraints are not differentiable, it is challenging to integrate a hard constraint into the parameterized approaches during a search. For path-based methods such as Greedy NAS [62], the searching process is split into two consecutive stages, *i.e.*, supernet training and architecture sampling. For training supernet, only one path consisting of a single operation choice is activated, and thus the memory cost is less than parameterized methods and scales well on large-scale datasets, *e.g.*, ImageNet. After training, the supernet acts as a performance estimator to greedily filter out weak paths so that potentially good paths for generating good candidates architecture. In this way, training efficiency can be much improved since weak paths involving unnecessary weight optimizations are avoided. Unlike the previous parameterized methods, the hard hardware constraint can be easily integrated into the searching process.

B. PARAMETER SHARING AND CONTINUOUS SEARCH SPACE

To improve the efficiency of NAS, different architectures are derived from the supernet and share the same weights. The technique of parameter sharing can avoid many redundant searches, exploration time and thus improve the efficiency of architecture search. For example, in ENAS [25], a sharing strategy is adopted to optimize weights between sub-models by using an RL-based controller to select various subgraphs from a directed acyclic graph (DAG). However, the search space for convolutional architectures is still discrete. As described in [25], if there are 6 operations available for deciding what feature function between two layers, the number of possible networks in the search space will be $6^L \times 2^{L(L-1)/2}$, where L is the number of layers in a network. If $L = 12$, the number of networks will become 1.6×10^{29}. Instead of searching over a discrete set of candidate architectures, the operation sharing technique [4] relaxes the search space to be continuous so that the architecture can be optimized by gradient descent. It relaxes the categorical choice of a particular operation as a set of mixing probabilities, where operations are not binary in terms of their existence and can be searched via a gradient descent-based method. With orders of magnitude fewer computation resources, the unique method "DARTS" [4] outperforms many existing approaches.

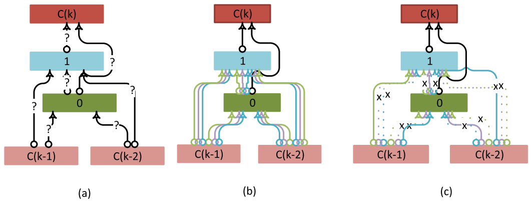

FIGURE 5. DARTS cell searching. (a) Unknown operations on edges. (b) Continuous relaxation by placing a mixed operation on edge. (c) Bi-level optimization and final architecture generation (best viewed in color).

DARTS [4] introduces a novel algorithm to transform the categorical selection problem to a differentiable search space by weighting all possible operations as a continuous function. Let $O = \{o_k\}_{k=1,\ldots,N}$ be the set of all possible N candidate operations (*e.g.*, convolution, pooling, skip, identify, *etc.*). Thus, given a network architecture represented by a directed acyclic graph (DAG), there are N paths (operations) to be chosen between two adjacent nodes. To make search space continuous and differentiable, DARTS sets the choice to be a mixed operation with N parallel paths instead of setting the choice to be a definite primitive operation. Figure 5 shows the structure of micro cell and its intermediate node connection used in DARTS [4]. Only two hidden nodes and their connection instead of four hidden nodes in a micro cell are shown for better understanding and a clear view. Here, $C(k-1)$ and $C(k-2)$ represent two predecessor micro cells (input features), Node 0 and node 1 are two hidden nodes, and the current micro cell is $C(k)$.

Given an input x, the output of a mixed operation $\mathcal{M}_O^{(i,j)}$ for an edge (i,j) in the DAG representation of network architecture is obtained by mixing the N candidate operations with Softmax:

$$\mathcal{M}_O^{(i,j)}(x) = \sum_{o_k \in O} \frac{\exp(\alpha_{o_k}^{(i,j)})}{\sum_{o \in O} \exp(\alpha_o^{(i,j)})} o_k(x), \tag{24}$$

where $\alpha_{o_k}^{(i,j)}$ is a real-valued weight parameter for the operation $o_k^{(i,j)}$. Then, the mixing weights for operations on the edge (i,j) are parameterized by an N-dimensional vector $\alpha^{(i,j)}$. The architecture search is then relaxed to learning a set of continuous architecture variables, *i.e.*, $A = \{\alpha^{(i,j)}\}$. With such relaxation, NAS becomes a bi-level optimization problem: optimizing the architecture A and its weights $w(A)$ alternatively. In real implementations, since the outputs of feature map of all N paths should be calculated and stored in the memory, DARTS easily exceeds the memory limits of hardware on large-scale datasets. Despite being computationally efficient, the stability and generalizability of DARTS have been challenged recently [6], [63]. At the end of DARTS, the continuous architecture should be projected onto a discrete representation to derive the best discrete architecture. Often this projection step can cause a significant performance drop between the mixture architecture and the obtained discrete architecture. Other GD-based NAS approaches transform discrete search space operations into a differentiable objective by applying either continuous relaxation [4], [8] or stochastic relaxation [25], [64] to improve searching performance and enhance GPU resource engagement. Apart from this, Lian *et al.* [65] propose a general transferability-based learning approach (T-NAS) to train supernet, and some specialized candidate architectures are extracted to easily adapt new tasks through a few gradient steps. As it takes fewer steps to find a suitable architecture for targeted tasks, the computational cost is very low and much more flexible. Yan *et al.* [66] propose a hierarchical mask-based neural architecture search approach to mask intermediate nodes, edges, and weights parameters. Their multi-level encoding strategy enables an architecture candidate to have arbitrary numbers of edges and operations with different importance. This strategy reduces the architecture search time since the supernet does not require training. A different type of gradient-based NAS approach, *i.e.*, ISTA-NAS [67] formulates the searching problem as a sparse encoding problem that compresses the search space and recovers sparse candidate architectures in an alternative way. The sparsity constraint is inherently satisfied at each update, so the search is more efficiently performed and consistent with evaluation. This strategy can improve accuracy and reduce the searching time.

C. STABILITY AND CONVERGENCE ANALYSIS

Gradient descent is an iterative optimization approach applied in NAS to minimize loss of a candidate architecture. A specific loss function is often defined and used to calculate the performance of a current candidate architecture in its parameters, and its gradients are used to tune architecture parameters A, network parameters w, or both. Let $\mathcal{L}_{valid}$ and $\mathcal{L}_{train}$ denote the objectives estimated from the validation dataset and the training dataset, respectively. DARTS aims to learn a set of continuous variables $A = \{\alpha^{(i,j)}\}$ by solving the following bi-level optimization:

$$\begin{aligned} &\min_{A} \mathcal{L}_{valid}(A, w^*(A)), \\ &s.t.\ w^*(A) = \arg\min_{w} \mathcal{L}_{train}(A, w), \end{aligned} \tag{25}$$

where A represents the searched architecture in DARTS. Finally, for the edge (i, j) in the DAG representation of an architecture A, its mixed operation $\mathcal{M}_O^{(i,j)}$ is replaced by the most similar operation, and a discrete architecture is generated for further training from the equation:

$$o^{(i,j)} = arg \max_{o \in O} \alpha_o^{(i,j)}. \tag{26}$$

Although the DARTS-based approach effectively reduces computational cost, this bi-level optimization problem is difficult to solve directly since both A and w parameters are high dimensional. It suffers from several problems as it alternatively optimizes (as shown in Eq.(25)) the architecture parameters and weight parameters. After convergence, DARTS removes operations with relatively weak attentions (see Eq.(26)) and causes a performance gap between the derived child networks and converged parent networks. Hence, SNAS [8] searches for operations and architecture topology simultaneously. It defines a matrix Z whose rows indicate masks multiplied to edges (i, j) in the DAG, and columns correspond to operations as a random variable to sample architectures. Then, a single-level stochastic optimization is used to optimize a generic loss via a Monte Carlo estimate. However, single-level optimization will overfit the architecture A and result in performance degradation during the validation process. Then, a mixed-level optimization [68] is proposed to deal with this problem and reduce gradient errors; that is,

$$\min_{w,A} [\mathcal{L}_{train}(w^*, A) + \lambda \mathcal{L}_{val}(w^*, A)], \tag{27}$$

where λ is a non-negative regularization variable. When $\lambda = 0$, Eq.(27) degrades to single-level optimization. If $\lambda \approx \infty$, Eq.(27) becomes a bi-level optimization (see Eq.(25)).

While current differentiable NAS methods have achieved impressive results, several works [6], [51] have cast doubt on their stability and generalization. Firstly, the searched architectures are often over-fitted and dominated by parameter-free operations. Secondly, the differential NAS methods work well often only on smaller datasets for shallower and narrower networks due to the large memory consumption of differentiable NAS approaches. Another common shortcoming is: the differential NAS approaches such as DARTS [4] often converge to a sharp region of validation loss landscape, *i.e.*, $\mathcal{L}_{valid}$, which results in significant performance drops in the final architecture. To tackle the problem of memory consumption, in PC-DARTS [5], Yan *et al.* proposed a partial connection strategy to randomly sample a subset of all channels in each step while bypassing the rest directly in a shortcut. The channel sampling task brings a tremendous reduction in memory and computation costs and thus can not only accelerate the network search but also stabilize the process, particularly for large-scale datasets. In RDARTS [6], Zela *et al.* pointed out that the approach adopted in DARTS, keeping only one operation with the largest weight in every edge and removing all operations from the edge, will make the final architecture converge into a sharp region of loss landscape. They also found that such stability is highly correlated with the dominant eigenvalue λ_{max}^A of the Hessian matrix $\nabla_\alpha^2 \mathcal{L}_{valid}$ of the estimated architecture, and thus proposed an early stop strategy to prevent λ_{max}^A from exploding. However, an early stop will fail if the convergence occurs from the beginning of the search process. For other approaches, *e.g.*, scheduled drop path [21], a new regularization of architecture parameters was proposed to address the stability of DARTS. Furthermore, to overcome the discretization gap, Fair-DARTS [57] observed that the amount of weak operators (such as skip connections) increases as the search process proceeds and will cause unfair competition among the operators. Then, this issue is addressed by relaxing the choice of operations, such that each operator has an equal opportunity to develop the architecture strength. In [8], Xie *et al.* used a concrete distribution to relax and make the discrete search space continuous and differentiable. In addition, Noy *et al.* [69] proposed an annealing-based soft pruning strategy to gradually prune out weaker operations so that the final candidate architecture can be efficiently searched and found. The comparisons among different gradient descent-based architecture search approach addressed in the study are summarized in Table 3.

VIII. PERFORMANCE EVALUATION

The commonly used performance evaluation metrics are:

(i) *Accuracy and efficiency*, which can be calculated during the learning process or inference stage. In order to select different optimization strategies during search procedures, it is necessary to measure the performance of each selected candidate architecture by comparing intermediate results and the expected ones. However, the evaluation of candidate architectures is usually computationally expensive and dominates the main computation cost. There have been different frameworks proposed for reducing the computational loading of performance evaluation during the search process in literature. One of the most effective ways to significantly reduce search cost in an evaluation process is to train an architecture with few epochs. It is often adopted in different SoTA architecture search approaches [4], [6], [23], [31], [33], [78] with useful performance metrics. However, Zela *et al.* [79] mentioned that the searching and training epochs do not differ drastically. Kyriakides and Margaritis [80] pointed out there is a positive correlation between the searching and training epochs. Actually, the absolute performance of each candidate architecture in search process is not very useful. In order to determine the quality of a candidate architecture, another way to speed up the efficiency of architecture evaluation is to evaluate architectures on small datasets during search and finely tune their real architecture parameters on large datasets. For example, DARTS [4], SmoothDARTS [51], and FairDARTS [57] search the cell structures on small dataset, *i.e.*, CIFAR10, construct architectures using these cells and estimate the performance on same dataset. Then the best cell structure is used to construct the final architectures on large targeted dataset, *i.e.*, ImageNet. Invariant of this to speed up the searching process, Stamoulis *et al.* [44] proposes a single path supernet that encodes architectures with sharing convolutional kernel parameters.

(ii) *Latency*, apart from accuracy, hardware-friendly convolutional neural networks are also necessary for different devices. Generally, latency time and the number of parameters inside an architecture are used to measure the performance of a hardware-friendly architecture. In architecture search on latency constraint, instead of searching for the best performing architecture, the architecture that satisfies the latency constraints with less latency (or better efficiency) must be extracted from a supernet. In this line, ProxylessNAS [33] defines a supernet with binary variables and combines the cross-entropy with latency constraints to finely tune weight parameters and architecture parameters during the searching stage. FBNet [41] also uses the same strategy as ProxylessNAS; instead of binary parameters in a supernet, it uses SGD to define the parameters. Based on the theory of prediction with expert advice, in XNAS [47], a suitable architecture is predicted and extracted from the supernet, and the Exponentiated-Gradient (EG) algorithm is adopted for mitigating the hard pruning.

Another way is to train a supernet as a performance estimator without further training. For example, GreedyNAS [62] constructs a supernet using a chain-based search space and adopts an evolution-based heuristic approach to find architectures with less latency on targeted devices. This progressive search space reduction strategy can significantly speed up the entire search process to find the final target architecture. Furthermore, SGNAS [81] proposes an architecture generator that can find out different architectures with different latency constraints in only one search iteration. In contrast, other one-shot approaches such as GreedyNAS [62] find out one architecture with only one latency constraint. To sum up, different NAS approaches propose different varieties of architectures for satisfying different constraints and purposes. Some of them can improve performance, but computational cost is still high. Another kind can be light-weighted (for mobile devices) with low latency or power, *etc.* So we only highlight their accuracy and efficiency (model parameters and computational costs) in Table 4 on different image classification datasets (*i.e.,* CIFAR-10, CIFAR-100, and ImageNet).

IX. RESEARCH ISSUES AND CHALLENGES

So far, we have discussed many interesting GD-based search strategies, their working principles, and how they can improve the search process performance in NAS. This section will discuss different challenging issues in NAS that need to be further explored. Although GD-based NAS can achieve impressive performance on different datasets with fewer computing resources (less than 0.5 GPU days on CIFAR10 dataset to search a candidate architecture [4]), it is difficult to predict why some approaches work well and how we can expand generalized structures for different datasets. Apart from this, gradient-based NAS suffers from the following issues: (i) the discretization nature making performance gap, (ii) challenging to support data-parallel technique, and (iii) searched candidate architectures being precarious and unstable. Most differential NAS approaches such as ProxylessNAS [33], DARTS [4], and SNAS [8] focus only on reducing the validation errors, which might not be good enough for searching a stable and generalized architecture. It is also unclear why NAS methods adopt a "Concatenation" operation instead of an element-wise addition operation for each block or cell [37]. Most explanations to results are more likely to hindsight, and there is no mathematical evidence [6], [84]. Many parameters in SoTA approaches need to be finely tuned through

TABLE 3. Gradient based NAS innovations.

Algorithm	Main context	Solving strategies	Outcomes
DARTS [4]	Formulate NAS as gradient descent based optimization problem.	Relax the discrete search space to be continuous. The softmax is used for smoothing the operation choices, and a candidate architecture is constructed by stacking the cell for training.	Optimize the architecture parameters via gradient descent and thus dramatically reduces the high search cost of NAS.
SNAS [8]	Formulate NAS as a stochastic model. Enhance RL with a smooth sampling scheme.	Samples and optimizes candidate architectures directly with concrete optimization [70].	More efficient and less regularization biased framework (compared with DARTS)
Proxylessnas [33]	A model trained and tested on different datasets often not guaranteed to be optimal.	Directly learn the architectures for large-scale target tasks and target hardware platforms.	Latency regularization loss helps for different hardware.
GDAS [45]	Formulate NAS as gradient descent problem	Samples one sub-graph at one training iteration	Better performance with less computing resources.
FairNAS [30]	Unfair bias in supernet sometimes reduce the performance of candidate architectures	Two levels of constraints: expectation fairness and strict fairness.	It can be adopted on any search pipeline.
PCDARTS [5]	DARTS based NAS suffered from large memory and computing overhead	Sample the supernet into a subnet and partially connect to construct a candidate architecture	Edge normalization can stabilize the search process.
RDARTS [6]	DARTS does not work robustly for new problem	Add different types of regularization methods with early stops.	Generalization improves in the search process.
BayesNAS [7]	Nodes inside normal and reduction cells often disregard their predecessors and successors.	A Hierarchical automatic relevance determination (HARD) approach is used to model architecture parameters.	Compress CNN by enforcing structural sparsity without accuracy deterioration
PDARTS [29]	Bridging the Depth Gap between Search and Evaluation	Gradually increase the searched architecture during training.	Regularized search space and improve accuracy.
XNAS [47]	New optimization method for differential NAS.	Designing for wiping out inferior architectures and enhance superior ones dynamically.	Fewer hyper-parameters need to be tuned.
DARTS+ [31]	Skip connection increases for larger epochs.	Early stopping into the original DARTS [4]	Improved the performance of DARTS.
NAT [48]	New optimization method for NAS	Redundant operations are replaced by the Markov decision process (MDP).	Reduces hyper-parameters and improve the accuracy
SETN [49]	After the search, a lengthy training requires to train the hyper-parameters for evaluations.	Template network shares parameters among all candidates.	Improve the quality of the candidate architecture for evaluation
StacNAS [71]	DARTS performs poorly when the search space is changed	Calculates correlation of similar operators incurs unfavorable competition among them.	Increase the stability and performance
Smooth DARTS [51]	Stabilize the architecture search process.	Perturbation-based regularization for improving the generalizability.	Stable candidate architecture.
DOTS [72]	Operation weights cannot indicate the importance of cell topology	Decouple the Operation and Topology Search (DOTS)	Topology search space to improve accuracy.
PARSEC [73]	Search directly on large scale problems.	Probability based architecture search approach	Reduce the computing costs.
SGAS [32]	Searched architectures often fail to generalize in the final evaluation.	Divides the search procedure into sub-problems, chooses, and greedily prunes candidate operations.	State-of-the-art architectures for tasks such as image classification
GDAS-NSAS [74]	Performance of preceding candidate architecture often degraded during training of new architecture with partially share weights.	Formulate supernet training as One-Shot NAS. During training, the performance of current architecture should not degrade the performance of preceding candidate architecture.	Improve predicatively of supernet in One-Shot NAS
DropNAS [75]	Co-adaption problem and Matthew Effect	Propose a novel grouped operation dropout algorithm	Achieves promising performance
DARTS- [76]	Instability issue during architecture searching	Skip connections with a learnable architectural coefficient	Improves the robustness of DARTS.
DrNAS [77]	Formulate the DARTS as a distribution learning problem	Progressive learning scheme to search architectures in a large dataset	Improves the generalization ability and induces stochasticity in search space

expert's advice during searching. However, more details are not mentioned about these parameters, such as in DARTS [4] why 8 cells are stacked together during searching for preferred cell structures (*i.e.*, normal and reduction cells), and 20 cells are stacked to construct the supernet for a target dataset. Usually, more cells to construct shallow and large networks can improve accuracy and increase the computational cost, but most SoTA approaches do not try to improve their architectures using more cells.

Another interesting study is that a labeled dataset is often with a limited scale; hence it is difficult to conclude whether a searched architecture will work well for real-world problems. Liu *et al.* [85] scoured the question, "can we find high-quality neural architecture without human-annotated labels ?" and demonstrated a new idea called Unsupervised Neural Architecture Search (UnNAS). Furthermore, a NAS-based approach must learn features from new data without discarding old learning, but when some pretrained

TABLE 4. Performance analysis of gradient based architecture search approaches on Cifar10, Cifar100, and ImageNet datasets.

Algorithm	Cifar10			Cifar100			ImageNet		
-	Error (%)	Params(M)	GPUDays	Error (%)	Params(M)	GPUDays	Error (%)	Params(M)	GPUDays
DARTS (1^{st}) [4]	3.00 ± 0.14	3.3	0.4	$17.76^{\#\triangle}$	$3.3^{\#\triangle}$	$1.5^{\#\triangle}$	26.7/8.7	4.7	4.0
DARTS (2^{nd}) [4]	2.76 ± 0.09	3.3	1	17.76	3.3	1.5	–	–	–
SNAS [8]	2.85 ± 0.02	2.8	1.5	–	–	–	$27.3/9.2^{*\triangle}$	$4.3^{*\triangle}$	$1.5^{*\triangle}$
ProxylessNAS [33]	2.08	5.7	4	–	–	–	$24.9/7.5^{*\square}$	$7.1^{*\square}$	$8.3^{*\square}$
PARSEC [73]	2.81 ± 0.03	3.7	1	–	–	–	$26.0/8.4^{\dagger\triangle}$	$5.6^{\dagger\triangle}$	$1.0^{\dagger\triangle}$
GDAS [45]	2.93	3.4	0.3	$18.38^{\dagger}$	$3.4^{\dagger}$	$0.2^{\dagger}$	$26.0/8.5^{\dagger\triangle}$	$5.3^{\dagger\triangle}$	$0.2^{\dagger\triangle}$
FairNAS[30]	1.8	–	–	22.7	–	–	$24.9/7.5^{\dagger\triangle}$ $24.4/7.4^{\dagger\square}$	$4.8^{\dagger\triangle}$ $4.3^{\dagger\square}$	$0.4^{\dagger\triangle}$ $3.0^{\dagger\square}$
PC-DARTS [5]	$2.57 \pm 0.07^{\ddagger}$	$3.6^{\ddagger}$	$0.1^{\ddagger}$	17.01±0.06	4.0	0.1	$24.2/7.3^{*\square}$ $25.1/7.8^{\ddagger\triangle}$	$5.3^{*\square}$ $5.3^{\ddagger\triangle}$	$3.8^{*\square}$ $5.3^{\ddagger\triangle}$
R-DARTS [6]	$2.95 \pm 0.21^{\dagger}$	4.1	1.6	$18.01 \pm 0.26^{\dagger}$	–	$1.6^{\dagger}$	–	–	
BayesNAS [7]	$2.81 \pm 0.04^{\ddagger}$	$3.4^{\ddagger}$	$0.2^{\ddagger}$	–	–	–	$26.5/8.9^{\dagger\triangle}$	$3.9^{\dagger\triangle}$	$0.2^{\dagger\triangle}$
P-DARTS[29]	$2.50^{\ddagger}$	$3.4^{\ddagger}$	$0.3^{\ddagger}$	$15.92^{\#}$	$3.6^{\#}$	$0.3^{\#}$	$24.4/7.4^{*\triangle}$ $24.7/7.5^{\dagger\diamondsuit}$	$4.9^{*\triangle}$ $5.1^{\dagger\diamondsuit}$	$0.3^{*\triangle}$ $0.3^{\dagger\diamondsuit}$
XNAS [47]	1.60	7.2	0.3	–	–	–	$24.0/-^{*}$	5.2^{*}	0.3^{*}
DARTS+ [31]	$2.37 \pm 0.13^{\#}$	$4.3^{\#}$	$0.6^{\#}$	$15.45 \pm 0.30^{\#}$	$3.9^{\#}$	$0.5^{\#}$	$23.7/7.2^{*\diamondsuit}$ $23.9/7.4^{*IM}$	$5.1^{*\diamondsuit}$ $5.1^{*\square}$	$0.2^{*\diamondsuit}$ $6.8^{*\square}$
NAT[48]	1.99	113	–	–	–	–	25.2 /07.7	8.36	–
SETN [49]	2.69	4.6	1.8	17.25	–	–	25.7 / 8.0	5.3	1.8
StacNAS [71]	$2.48 \pm 0.08^{\#}$	$3.9^{\#}$	$0.8^{\#}$	$16.11 \pm 0.2^{\#}$	$4.3^{\#}$	$0.8^{\#}$	$24.3/6.4^{\#}$	$5.7^{\#}$	$20^{\#}$
SDARTS[51]	$2.61 \pm 0.02^{\ddagger}$	$3.3^{\ddagger}$	$1.3^{\ddagger}$	20.56	–	–	$25.2/7.8^{\dagger}$	$5.4^{\dagger}$	$1.3^{\dagger}$
DATA + Cutout [82]	2.59	3.4	1	15.45 ± 0.3	3.9	0.5	–	–	–
TAS [83]	6.82	N/A	0.3	–	–	–	–	–	–
SGAS [32]	2.66	3.7	0.25	–	–	–	$24.4 \pm 0.16/$ 7.29 ± 0.09	5.3	0.25
GDAS-NSAS [74]	2.73	3.4	0.4	18.02	–	–	–	–	–
ASAP [69]	$2.49 \pm 0.04^{\#}$	$2.5^{\#}$	$0.2^{\#}$	$15.6^{\#}$	$2.5^{\#}$	$0.2^{\#}$	$26.7/-^{*}$	–	0.2^{*}
DropNAS [75]	$2.58 \pm 0.14^{\#}$	$4.1^{\#}$	$0.6^{\#}$	$16.95 \pm 0.41^{\#}$	$4.4^{\#}$	$0.7^{\#}$	$23.4/6.7^{\#\triangle}$ $23.5/6.8^{\#\diamondsuit}$	$5.7^{\#\triangle}$ $6.1^{\#\diamondsuit}$	$0.6^{\#\triangle}$ $0.7^{\#\diamondsuit}$
DOTS [72]	2.44 ±0.05	3.6	0.2	16.44	3.9	0.2	24.0 /7.2	5.3	1
DARTS- [76]	$2.5^{\dagger}$	$3.5^{\dagger}$	$0.4^{\dagger}$	$17.51 \pm 0.25^{\dagger}$	$3.3^{\dagger}$	$0.4^{\dagger}$	$23.8/7.0^{\dagger\square}$	$4.9^{\dagger\square}$	$4.5^{\dagger\square}$
DrNAS [77]	$2.46 \pm 0.03^{\ddagger}$	$4.1^{\ddagger}$	$0.6^{\ddagger}$	–	–	–	$23.7/7.1^{\ddagger\square}$	$5.7^{\ddagger\square}$	$4.6^{\ddagger\square}$

– represents the corresponding information is not provided in the original paper or in reputed articles. $*$, †, ‡, and # indicated the results are taken from [31], [76], [77], and [75].

$^{\triangle}$, $^{\diamondsuit}$, and $^{\square}$ denote the performances of architecture search evaluated on CIFAR-10, CIFAR-100, and ImageNet datasets, respectively.

architectures are adapted for new data, their performances often degrade significantly on old datasets. To deal with this issue, in [86], Li *et al.* proposed the Learning without Forgetting (LwF) approach to training architecture only for new data without forgetting old learning. Although GD-based NAS approaches can produce imperial performance, they cannot explain why a specific candidate architecture produces better solutions and how similarly derived candidate architectures are in independent runs. Identifying these common issues and understanding what elements are essential for designing architecture with high performance may need to be studied in the future. Better mathematical interpretation of NAS will be good for future research.

X. CONCLUDING REMARKS

A. LESSONS LEARNED

This study has reviewed various GD-based NAS approaches from different directions, and here we summarize the lessons learned from this survey. Gradient descent is a better solution for architecture search in NAS approaches and ignoring it will increase architecture search cost in terms of GPU days. The working principle of NAS is divided into three stages: search space, search strategy, and performance estimation. In search strategy, different methods are used to optimize the candidate architectures, such as evolutionary algorithm (EA), reinforcement learning (RL), gradient-based (GD), and random method. Among these optimization approaches, random method is treated as a baseline for architecture search, and gradient-based methods require low computational cost and improve validation accuracy. Various types of search spaces and their search strategies are also explored in this survey. In Table 4, we have evaluated and compared significant performance matrices in terms of a validation error, number of architecture parameters, and computational costs (GPU days) for different GD-based NAS approaches.

B. CONCLUSION

NAS has become one of the main steps for Auto-ML in the current era. Automatic generation of neural architecture goes through search space, search strategy, and performance evolution stages. However, the demand for automatic architectures is gradually increasing as data increases continuously at an exponential scale. GD-based architecture search approaches are typically used to reduce computing costs and increase the efficiency of architecture searching. This survey has discussed different GD-based architecture search techniques and their management approaches.

We started with a simple discussion of NAS and listed its unique properties of GD approaches. We also discussed different impacts of gradient descent during the search process and compared their pros and cons. We compared different gradient descent-based architecture search approaches in terms of their representation, accuracy, search costs, and parameters on CIFAR-10, CIFAR-100, and ImageNet classification datasets. Finally, we addressed the research challenges, and open issues on NAS approaches.

REFERENCES

[1] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[2] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent., (ICLR)*, 2017, pp. 1–16.

[3] B. Zoph and Q. V. Le. (2017). *Neural Architecture Search With Reinforcement Learning*. [Online]. Available: OpenReview.net

[4] H. Liu, K. Simonyan, and Y. Yang. (2019). *Darts: Differentiable Architecture Search*. [Online]. Available: OpenReview.net

[5] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. (2020). *Pc-Darts: Partial Channel Connections for Memory-Efficient Architecture Search*. [Online]. Available: OpenReview.net

[6] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–28.

[7] H. Zhou, M. Yang, J. Wang, and W. Pan, "Bayesnas: A Bayesian approach for neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, vol. 97, K. Chaudhuri R. Salakhutdinov, Eds. Long Beach, CA, USA: PMLR, 2019, pp. 7603–7613.

[8] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. 7th Int. Conf. Learn. Represent., (ICLR)*, 2019, pp. 1–17.

[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, pp. 55:1–55:21, Mar. 2019.

[10] Y. Jaafra, J. Luc Laurent, A. Deruyver, and M. Saber Naceur, "Reinforcement learning for neural architecture search: A review," *Image Vis. Comput.*, vol. 89, pp. 57–66, Sep. 2019.

[11] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *CoRR*, vol. abs/2006.02903, 2020.

[12] C. White, W. Neiswanger, S. Nolen, and Y. Savani, "A study on encodings for neural architecture search," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 1–17.

[13] M. Wistuba, A. Rawat, and T. Pedapati, "A survey on neural architecture search," 2019, *arXiv:1905.01392*. [Online]. Available: https://arxiv.org/abs/1905.01392

[14] R. E. Shawi, M. Maher, and S. Sakr, "Automated machine learning: State-of-the-art and open challenges," 2019, *arXiv:1906.02287*. [Online]. Available: https://arxiv.org/abs/1906.02287

[15] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl. Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[16] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360

[17] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-VGGNet models for scene recognition," 2015, *arXiv:1508.01667*. [Online]. Available: http://arxiv.org/abs/1508.01667

[18] P. L. Ballester and R. M. de Araújo, *On the Performance of GoogleNet and AlexNet Applied to Sketches*, D. Schuurmans M. P. Wellman, Eds. Palo Alto, CA, USA: AAAI Press, 2016, pp. 1124–1128.

[19] S. Targ, D. Almeida, and K. Lyman, "ResNet in ResNet: Generalizing residual architectures," 2016, *arXiv:1603.08029*. [Online]. Available: http://arxiv.org/abs/1603.08029

[20] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, "DenseNet: Implementing efficient ConvNet descriptor pyramids," 2014, *arXiv:1404.1869*. [Online]. Available: http://arxiv.org/abs/1404.1869

[21] 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.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861

[23] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. 33rd AAAI Conf. Artif. Intell., (AAAI)*. Palo Alto, CA, USA: AAAI Press, 2019, pp. 4780–4789.

[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[25] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.

[26] 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, no. 1, 2018, pp. 1–8.

[27] Y. Chen, G. Meng, Q. Zhang, S. Xiang, C. Huang, L. Mu, and X. Wang, "RENAS: Reinforced evolutionary neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4787–4796.

[28] R. Shin, C. Packer, and D. Song. (2018). *Differentiable Neural Network Architecture Search*. [Online]. Available: OpenReview.net

[29] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1294–1303.

[30] X. Chu, B. Zhang, R. Xu, and J. Li, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," 2019, *arXiv:1907.01845*. [Online]. Available: http://arxiv.org/abs/1907.01845

[31] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, "DARTS+: Improved differentiable architecture search with early stopping," 2019, *arXiv:1909.06035*. [Online]. Available: http://arxiv.org/abs/1909.06035

[32] G. Li, G. Qian, I. C. Delgadillo, M. Muller, A. Thabet, and B. Ghanem, "SGAS: Sequential greedy architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1620–1630.

[33] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *Proc. 7th Int. Conf. Learn. Represent., (ICLR)*, 2019, pp. 1–13.

[34] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–5.

[35] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 459–468.

[36] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 1–25.

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

[38] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.

[39] 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)*, Sep. 2018, pp. 19–34.

[40] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2018, pp. 7827–7838.

[41] B. Wu, K. Keutzer, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, and Y. Jia, "FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10734–10742.

[42] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.

[43] X. Dai, Y. Jia, P. Vajda, M. Uyttendaele, N. K. Jha, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, and Y. Wu, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11398–11407.

[44] D. Stamoulis, R. Ding, D. Wang, D. Lymberopoulos, B. Priyantha, J. Liu, and D. Marculescu, "Single-path NAS: Device-aware efficient ConvNet design," 2019, *arXiv:1905.04159*. [Online]. Available: http://arxiv.org/abs/1905.04159

[45] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1761–1770.

[46] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[47] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik, "XNAS: Neural architecture search with expert advice," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1977–1987.

[48] Y. Guo, Y. Zheng, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang, "NAT: Neural architecture transformer for accurate and compact architectures," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 1–31.

[49] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 3681–3690.

[50] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, 2020, pp. 544–560.

[51] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 1554–1565.

[52] V. Nair and G. E. Hinton, *Rectified Linear Units Improve Restricted Boltzmann Machines*, J. Fürnkranz and T. Joachims, Eds. Madison, WI, USA: Omnipress, 2010, pp. 807–814.

[53] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1. Princeton, NJ, USA: Citeseer, 2013, p. 3.

[54] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," in *Proc. 6th Int. Conf. Learn. Represent., (ICLR)*, 2018. [Online]. Available: https://openreview.net/forum?id=SkBYYyZRZ

[55] D. Misra, "Mish: A self regularized non-monotonic neural activation function," 2019, *arXiv:1908.08681*. [Online]. Available: https://arxiv.org/abs/1908.08681

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[57] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair darts: Eliminating unfair advantages in differentiable architecture search," in *Proc. Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, 2020, pp. 465–480.

[58] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," 2015, *arXiv:1511.06807*. [Online]. Available: http://arxiv.org/abs/1511.06807

[59] X. Chu, B. Zhang, and X. Li, "Noisy differentiable architecture search," 2020, *arXiv:2005.03566*. [Online]. Available: http://arxiv.org/abs/2005.03566

[60] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. Uncertainty Artif. Intell.*, 2020, pp. 367–377.

[61] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1304–1313.

[62] S. You, T. Huang, M. Yang, F. Wang, C. Qian, and C. Zhang, "GreedyNAS: Towards fast one-shot NAS with greedy supernet," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1996–2005.

[63] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *Proc. 8th Int. Conf. Learn. Represent., (ICLR)*, 2020, pp. 1–16.

[64] S. Shirakawa, Y. Iwata, and Y. Akimoto, *Dynamic Optimization of Neural Network Structures Using Probabilistic Modeling*. Palo Alto, CA, USA: AAAI Press, 2018, pp. 4074–4082.

[65] D. Lian, Y. Zheng, Y. Xu, Y. Lu, L. Lin, P. Zhao, J. Huang, and S. Gao, "Towards fast adaptation of neural architectures with meta learning," in *Proc. Int. Conf. Learn. Represent.*, 2020. [Online]. Available: https://openreview.net/forum?id=r1eowANFvr

[66] S. Yan, B. Fang, F. Zhang, Y. Zheng, X. Zeng, M. Zhang, and H. Xu, "HM-NAS: Efficient neural architecture search via hierarchical masking," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop (ICCVW)*, Oct. 2019, pp. 1942–1950.

[67] Y. Yang, H. Li, S. You, F. Wang, C. Qian, and Z. Lin, "ISTA-NAS: Efficient and consistent neural architecture search by sparse coding," 2020, *arXiv:2010.06176*. [Online]. Available: http://arxiv.org/abs/2010.06176

[68] C. He, H. Ye, L. Shen, and T. Zhang, "MiLeNAS: Efficient neural architecture search via mixed-level reformulation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11993–12002.

[69] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik, "Asap: Architecture search, anneal and prune," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 493–503.

[70] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. 5th Int. Conf. Learn. Represent., (ICLR)*, 2017, pp. 1–19.

[71] G. Li, X. Zhang, Z. Wang, Z. Li, and T. Zhang, "StacNAS: Towards stable and consistent optimization for differentiable Neural Architecture Search," *CoRR*, vol. abs/1909.11926, 2019. [Online]. Available: http://arxiv.org/abs/1909.11926

[72] Y.-C. Gu, L.-J. Wang, Y. Liu, Y. Yang, Y.-H. Wu, S.-P. Lu, and M.-M. Cheng, "DOTS: Decoupling operation and topology in differentiable architecture search," 2020, *arXiv:2010.00969*. [Online]. Available: http://arxiv.org/abs/2010.00969

[73] F. P. Casale, J. Gordon, and N. Fusi, "Probabilistic neural architecture search," 2019, *arXiv:1902.05116*. [Online]. Available: http://arxiv.org/abs/1902.05116

[74] M. Zhang, H. Li, S. Pan, X. Chang, and S. Su, "Overcoming multi-model forgetting in one-shot NAS with diversity maximization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7809–7818.

[75] W. Hong, G. Li, W. Zhang, R. Tang, Y. Wang, Z. Li, and Y. Yu, "DropNAS: Grouped operation dropout for differentiable architecture search," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 1–7.

[76] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "DARTS-: Robustly stepping out of performance collapse without indicators," in *Proc. Int. Conf. Learn. Represent.*, 2021. [Online]. Available: https://openreview.net/forum?id=KLH36ELmwIB

[77] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh, "DRNAS: Dirichlet neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2021. [Online]. Available: https://openreview.net/forum?id=9FWas6YbmB3

[78] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Amsterdam, The Netherlands: Elsevier, 2019, pp. 293–312.

[79] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," 2018, *arXiv:1807.06906*. [Online]. Available: http://arxiv.org/abs/1807.06906

[80] G. Kyriakides and K. Margaritis, "The effect of reduced training in neural architecture search," *Neural Comput. Appl.*, vol. 32, no. 23, pp. 17321–17332, 2020.

[81] S.-Y. Huang and W.-T. Chu, "Searching by generating: Flexible and efficient one-shot NAS with architecture generator," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021. [Online]. Available: https://pythonrepo.com/repo/eric8607242-SGNAS

[82] J. Chang, Y. Guo, G. Meng, S. Xiang, and C. Pan, "Data: Differentiable architecture approximation," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 876–886.

[83] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 759–770.

[84] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: When to warp?" in *Proc. Int. Conf. Digit. Image Computing: Techn. Appl. (DICTA)*, Nov. 2016, pp. 1–6.

[85] C. Liu, P. Dollár, K. He, R. Girshick, A. Yuille, and S. Xie, "Are labels necessary for neural architecture search," in *Proc. Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, 2020, pp. 798–813.

[86] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

SANTANU SANTRA received the B.Sc. and M.Sc. degrees from Vidyasagar University, India, in 2004 and 2006, respectively, and the M.Tech. degree from West Bengal University of Technology, India, in 2009. From 2009 to 2014, he was an Assistant Professor with the Bengal Institute of Technology and Management, India. He is currently a Research Scholar with the Department of Computer Science and Engineering, Yuan Ze University, Taoyuan, Taiwan. His research interests include computer vision, image processing, and VLSI backend design.

JUN-WEI HSIEH was an Associate Professor with the Department of Electrical Engineering, Yuan-Ze University, and a Visiting Researcher with the MIT AI Laboratory. Since August 2009, he has been a Professor and the Dean of the Department of Computer Engineering, National Taiwan Ocean University. Since August 2019, he has been a Professor with the College of AI, National Chiao-Tung University. He hosted or co-hosted a lot of large-scale AI projects from different companies and governments in the past. He has a lot of successful experiences in industrial-academic cooperation and technology transferring, especially in ITS. He finished more 30 technology transferring projects, from 2013 to 2020. His research interests include AI, deep learning, smart farming, video surveillance, intelligent transportation systems, image and video processing, object recognition, machine learning, 3D printing, medical image analysis, and computer vision. In May 2019, he received the First Prize of the Ministry of Science and Technology Best Display Award and the Third Place of the AI Investment Potential Award. Due to his contributions in traffic flow estimation, he helped the Elan company received the Gold Award from Taipei International Computer Show, in 2019. He also received the Outstanding Research Award of National Taiwan Ocean University, in 2012, 2016, 2017, and 2019, the Outstanding Research Award of Yuan Ze University, in 2006. He and his students received the Silver Medal of 2019 National College Software Creation Competition, the Silver Medal of 2018 National Microcomputer Competition, the Best Paper awards of Information Technology and Applications in Outlying Islands Conference, in 2013, 2014, 2016, 2017, and 2018, respectively, and the Best Paper Award of Tanet 2017. He also received the Best Paper Award of CVGIP Conference, in 1999, 2003, 2005, 2007, 2014, 2017, and 2018, the Best Paper Award of DMS Conference, in 2011, the Best Paper Award of IIHMSP 2010, and the Best Patent Award of Institute of Industrial Technology Research, in 2009 and 2010, respectively.

CHI-FANG LIN was born in Taiwan, Republic of China, in 1960. He received the B.S. degree in transportation engineering and management from National Chiao Tung University, in 1983, and the M.S. and Ph.D. degrees from the Institute of Computer Engineering and Science, National Chiao Tung University, in 1986 and 1991, respectively. From 1987 to 1989, he was an Instructor with the Lien Hu Junior College of Technology. He joined the Department of Computer Science and Engineering, Yuan Ze University (YZU), in August 1991, where he is currently a Professor. From 1999 to 2002, he was the Chairman of the Department of Information Networking Technology. He was the Director of the Information Technology Research Center. From 2005 to 2008, he was the Chief Information Officer of YZU, where he is also the Head of the Department of Computer Science and Engineering, and also the Head of the Graduate Program in Biomedical Informatics.