Neural Architecture Search by Estimation of Network Structure Distributions

Anton Muravev, *Graduate Student Member, IEEE,* Jenni Raitoharju, *Member, IEEE,* and Moncef Gabbouj, *Fellow, IEEE*

***Abstract*—The influence of deep learning is continuously expanding across different domains, and its new applications are ubiquitous. The question of neural network design thus increases in importance, as traditional empirical approaches are reaching their limits. Manual design of network architectures from scratch relies heavily on trial and error, while using existing pretrained models can introduce redundancies or vulnerabilities. Automated neural architecture design is able to overcome these problems, but the most successful algorithms operate on significantly constrained design spaces, assuming the target network to consist of identical repeating blocks. We propose a probabilistic representation of a neural network structure under the assumption of independence between layer types. A matrix of probabilities is equivalent to the population of models, but simpler to interpret and analyze. We construct an architecture search algorithm, inspired by the estimation of distribution algorithms, to take advantage of this representation. The probability matrix is tuned towards generating high-performance models by repeatedly sampling the architectures and evaluating the corresponding networks. Our algorithm is shown to discover models which are superior to handcrafted architectures and competitive with those produced by existing architecture search methods, both in accuracy and computational costs, while being conceptually simple and highly extensible.**

***Index Terms*—Automatic architecture design, estimation of distribution algorithm, deep learning, convolutional neural network.**

I. Introduction

THE recent successes of deep learning have attracted significant interest from theoretical and practical standpoints in numerous fields of knowledge [1]. Computer vision in particular has witnessed the development of multiple successful models, based on convolutional neural networks (CNNs), for tasks such as classification [2], [3], semantic segmentation [4], and detection [5]. Deep learning driven approaches have notably contributed to the fields of audio processing [6], bioinformatics [7], among other fields. While the growth of deep learning solutions over the years is impressive, their adoption brings many significant challenges. Some of these fall into the category of practical issues and have been a subject of extensive research. For example, the tendency of powerful models to overfit the training data is addressed by parameter regularization techniques, such as dropout [8], while the vanishing gradient problem is tackled by normalization [9], [10]. On the other hand, the theoretical foundation of deep learning is not yet comprehensively established, but receives growing attention, as optimality conditions are formulated [11], [12] and common techniques are explained [13]. The lack of interpretability of decisions made by deep models [14], [15] is a difficult problem to tackle, but has attracted increasing research attention recently [16]. Further concerns have been raised regarding secure practical use of deep models, as they were shown to be vulnerable to attacks utilizing malicious data [17].

One aspect of the neural networks, intricately tied to these challenges, is the architectural design: the choice of layer count, connection patterns, neuron operations, and their hyperparameters (convolution filter sizes, channel depth). It is well-known that some structural choices are associated with training difficulties; for example, a depth increase causes the vanishing gradient problem [18]. Meanwhile, on a system level, the design guidelines of creating a deep network for a particular practical problem are not well established. The network design task thus becomes a time-demanding process, involving extensive trial and error. In practice, this issue is commonly avoided by using an already established pre-trained model of the same or related data domain as a feature extractor [19]. While effective, the latter approach presents problems of its own. Pre-trained models tend to be large and can lead to resource-consuming, largely redundant systems, whereas a too small network may produce insufficient accuracy for the problem at hand. Specific features of the data may require specific layer types to be fully exploited [20]. Pre-trained models can carry undesired biases from their original datasets [21]. Additionally, sharing the foundation means that such systems will naturally be more vulnerable to adversarial attacks. A promising way to avoid these problems lies in developing appropriate methods for automated task-specific network design.

The idea of automated neural network design dates back to the early 1990s [22]. The following decade saw a large volume of research on this problem, primarily focusing on evolutionary algorithms as solvers, both due to their gradient-free nature and shared biological inspirations [23]. This family of approaches would later be coined *neuroevolution* [24]. The research continued into the 2000s, with both improved evolutionary algorithms [24], [25] and other metaheuristic approaches, such as particle swarm optimization [26], as the search method. However, all of these algorithms share the need to perform many evaluations of intermediate solutions, which, in the case of automated architecture design, requires training numerous candidate networks from scratch. Therefore, these approaches were computationally restricted to rather limited

The authors would like to acknowledge the financial support of the Academy of Finland, project no. 289364.

All authors are with Faculty of Information Technology and Communication Sciences, Tampere University, FI-33100 Tampere, Finland.

model complexity, and their practical applications remained primarily in control tasks and robotics, where these limitations have a smaller impact [27].

The advent of deep learning, where training a single model can take days or weeks, caused manual design to once again become the primary approach. However, as architectural discoveries paved the way to models with a small number of parameters and superior performance [2], [3], the interest in automated design reemerged, taking advantage of both evolutionary optimization [28]–[32] and newer approaches, such as reinforcement learning [33]–[35]. The computational demand remains a major limitation and has hence been the focus of most recent works in the area [35]–[37].

Another concern is the growing *semantic* complexity of such algorithms. While they may yield successful architectures [31], their search behaviour is hard to analyze, which obscures the effects (whether positive or negative) of individual algorithmic steps and hinders comparisons. For instance, Zoph *et al.* [34] proposed a reduced search space: the network is represented as a repeating sequence of cells of a few types, where the internal structure of cells of the same type is identical and subject to optimization. Consequently, this design space has been adopted by a multitude of other approaches (see [31], [35], [38]), but random search has recently been shown to be highly competitive as well, suggesting that previous successes may come from the expressivity of this space rather than algorithm specifics [39].

Despite the variety of the proposed solutions for the architecture search problem, the majority of them rely on the repeating blocks, cells, or motifs. While effective in reducing the search space and thus (indirectly) the computation required, such an approach is by design biased towards deeper and uniformly structured models. This brings them closer to handcrafted networks, but limits the possibilities to discover less regular architectures, which can potentially be superior in performance or have other valuable properties.

We propose a conceptually simple and extensible architecture search method, based on the estimation of distribution algorithms. We specifically draw from Population-Based Incremental Learning [40] and Univariate Marginal Distribution Algorithm [41]. We utilize a set of discrete probability distributions to describe the choice of layers in a feedforward deep network, assuming their independence. Together they form a *network prototype*, which is then iteratively updated by sampling and evaluating network models, until convergence is reached. The contributions of this paper can be summarized as follows:

- We propose a probabilistic representation of deep neural networks by expressing their structure as a set of layer type probabilities. A single prototype of the proposed form corresponds to not a single network, but a family of models, which can span entire regions of the design space.
- We propose a CNN architecture search method based on the optimization of the above prototype, denoted Architecture Search by Estimation of network structure Distributions (ASED). As candidate networks are sampled and individual probabilities converge to their extreme values, the algorithm naturally transitions from global to local search, avoiding suboptimal areas.
- We propose additional techniques to introduce non-linear connectivity patterns to solutions and to control the speed of search convergence.
- We experimentally demonstrate the comparable performance of our method to existing approaches in terms of both model performance and computational requirements, while using only feedforward structures without explicit repeated motifs.

The rest of the paper is structured as follows. In Section 2, we review the related developments in the field of neural architecture optimization as well as elaborate on our inspirations. Section 3 describes our approach to architecture search and the additional related techniques. Section 4 contains the experimental results and their analysis. Finally, Section 5 concludes the work and outlines some potential future studies.

II. Related Work

Ever since the wider adoption of multilayer perceptron structures [42], their architectures became subject to optimization. While general neural network design involved heuristic rules and empirical tests, a promising alternative was found in evolutionary optimization methods due to their ability to solve problems defined only by the target function, without requiring any gradient information [43][44]. Early works on neuroevolution included [45]–[47]. However, all of these works shared common issues of prohibitive computational requirements and lack of robustness due to the highly noisy nature of the search space [23].

The small-scale neuroevolution reached a new peak when NEAT (NeuroEvolution of Augmented Topologies) [24] was introduced in 2002. The techniques that made NEAT differ from its predecessors are historical gene markings, allowing for straightforward and meaningful crossover, and speciation with fitness sharing, which allows promising individuals to more consistently reach their full potential. Despite the advantages and the flexibility it offered, NEAT remained limited to small-scale applications, such as control tasks with limited inputs (a problem which would later be tackled within reinforcement learning). Multiple subsequent variants of NEAT [25], [48], [49] aimed at efficient generation and representation of more complex networks with repeatable structural patterns. For example, HyperNEAT [25] encodes network architectures via a metric space, called *substrate*, where every neuron is mapped to a point with fixed coordinates. A separate hypermodel, called a Compositional Pattern-Producing Network (CPPN), takes as inputs the coordinates of a pair of neurons and outputs the connection weight between them, thus defining a network structure. An evolved substrate variant of HyperNEAT, or es-HyperNEAT [49], avoids the need to explicitly define node geometry, allowing for discovery of a wider scope of structures. However, despite the greater representational expression of these methods, the overarching problems remained and the use was limited to specific small-scale applications [50].

The resurgence of interest for architecture optimization started in 2016, after the introduction of reinforcement learning driven Neural Architecture Search (NAS) in [33]. An LSTM-based recurrent neural network (a controller) is trained to output sequences of tokens (symbols), which correspond to specific values of convolutional layers' parameters, such as filter size, count, and stride. The resulting neural network can be trained and evaluated. The controller can then be updated by the REINFORCE rule to maximize the expected accuracy of the generated networks. The major weakness of NAS is the computational cost of over 22000 GPU days on a standard CIFAR-10 image classification dataset. The follow-up work NAS-Net [34] represents the target network as a predefined sequence of repeating elements, known as *cells*. Each cell type shares the same internal structure, which is optimized in a graph form and can contain different convolution and pooling operations. During the search process, the total number of cells in the network is reduced to speed up computation, while the final discovered architecture is evaluated in a full-length sequence. Such a reduction of the design space has proven effective in guiding the search, thus boosting the accuracy and reducing the running time to 2000 GPU days, and has since been used in other works. Efficient NAS, or ENAS [35], achieves further speed-up (to less than 16 GPU hours) at the cost of some accuracy loss. It utilizes *weight sharing*, where the convolutional filter weights are identical between the cells and depend only on the position of the corresponding edge in the structural graph. Thus, training from scratch (which was necessary for the network evaluation) is no longer needed, and the tensor of shared weights can be finetuned via gradient updates in-between controller updates.

Evolutionary algorithms arose once again as a primary competitor to reinforcement learning based solutions. CoDeepNEAT [51] adapts the well-known NEAT procedure for deep networks by using two separate populations - *blueprints* and *modules* - for easier representation of repeating patterns. Genetic CNN [28] encodes layer connectivity in a population of binary strings and runs a standard genetic algorithm. EvoCNN [32] instead opts for variable-length gene encoding to represent networks of arbitrary depth, where crossover is made possible via matching the genes that share a type (e.g. a pooling gene can only be matched with another pooling gene). Real *et al.* [29] run a distributed large-scale evolutionary process directly on a population of networks, where mutations can alter the network structure, parameters, or training process. The following work of Real *et al.* [31] combines the evolutionary approach with the NAS-Net search space, surpassing reinforcement learning in anytime accuracy and setting a new state-of-the-art performance on the popular CIFAR-10 dataset, as well as generating comparatively simpler models. However, the computational cost remains extensive, clocking above 3000 GPU days. between and utilizes a special variant of crossover that only allows . and The similar approach is taken by the automatically evolving CNN (AE-CNN) , which runs a genetic algorithm on the population of networks composed of customized ResNet and DenseNet blocks, achieving competitive results.

While deep networks can be difficult for the neuroevolution to handle, a viable alternative can be found in expanding the operation set of the shallow networks, allowing for more powerful representations. Generalized Operational Perceptron (GOP) [52] model substitutes the standard neuron by offering a wider choice of nodal and pooling operations instead of the standard multiplication and addition. The choice of operations can be optimized simultaneously with the network architecture by a greedy incremental procedure. Operational Neural Networks (ONNs) [53], composed of such units, have been shown to achieve superior performance to CNNs on some practical problems. Most recently heterogeneous GOP structures, where each layer can have neurons with differing operations, have received increasing attention [54]. While flexibility of operators allows ONNs to stay relatively shallow, it also results in a vast unstructured design space which is computationally costly to traverse.

Many recent works in architecture optimization utilize various techniques to reduce the computation needed, primarily by simplifying the evaluation procedure. SMASH [55] learns a hypernetwork that can predict weights for all the connections of an arbitrary deep network (given a specific representation), which reduces the need for training and makes random search a viable solution for discovering architectures. Progressive Neural Architecture Search (PNAS) [36] uses a separate recurrent network to approximately rank the candidate models without training them, allowing the search to focus only on more promising options. NASH [56] and LEMONADE [37] take advantage of network morphisms—operations that modify the structure of a trained network without affecting its output—to navigate the search space without training the models from scratch. Differentiable Architecture Search (DARTS) [38] provides a continuous relaxation of the NAS-Net cell structure problem and performs the search via gradient descent, iterating between the architecture and weight updates. While relatively more efficient in terms of computation, these methods do not address the issues of interpretability and semantic complexity.

There exists a number of works that model the network construction as a probabilistic process, sharing some similarities with the proposed approach. Methods based on reinforcement learning, such as NAS and its successors, use the probability of a given network to be produced from the current policy as a weight for the corresponding reward. InstaNAS [57] also has reinforcement learning at its core, but differs from other algorithms in this group, as it takes an instance-aware approach. Specifically, InstaNAS processes each data point by a separate network (a path within a large trained model), sampled from a parameterized distribution. NASBOT [58] models the architecture search as a Gaussian process. To facilitate this, the authors introduce a (pseudo)distance in the network design space and utilize an evolutionary algorithm as an optimizer. The most similar approach to ours is Probabilistic Neural Architecture Search (PARSEC) [59]. As in our work, PARSEC explicitly models a distribution to produce neural architectures of cells, including the assumption of independence between individual operations. However, this distribution operates on a level of NAS-Net cell, allowing PARSEC to take advantage of full weight sharing between the sampled model instances, while our method models the network as a whole. Moreover, the search procedure is different: PARSEC uses Monte Carlo empirical Bayes to iteratively update both the architectural priors and the tensor of shared weights, while we completely recompute the marginal probabilities over a subset of the samples and do not use weight sharing.

Our work draws inspiration from the estimation of distribution algorithms (EDAs) – the family of optimization methods originating from mid-1990s, which are closely related to genetic algorithms [60]. While most evolutionary algorithms maintain a candidate population, which implicitly defines the probability distribution of the solutions, EDAs define this distribution explicitly and tune its parameters throughout the optimization process. Our work mainly draws on two discrete univariate EDAs, Population-Based Incremental Learning (PBIL) [40] and Univariate Marginal Distribution Algorithm (UMDA) [41]. PBIL generates an intermediate population via sampling, applies a selection procedure, and updates the probabilistic model in the direction of selected samples, using a learning rate parameter. UMDA maintains the population of solutions, estimates a set of marginal probabilities from the best candidate(s), and uses them to produce the population of the next generation. For more information on EDAs, their applications and recent developments, we direct the reader to the survey by Hauschild and Pelikan [61]. To the best of our knowledge, ours is the first work to explicitly apply the EDA formulation to the deep neural network architecture search problem.

III. Methodology

In this section, we describe and justify the proposed network representation, the design of the proposed algorithm Architecture Search by Estimation of network structure Distributions (ASED), as well as additional techniques to improve its capabilities.

A. Search Space and Network Representation

The problem of optimizing the structure of a neural network is extremely high-dimensional. The choice of layer types (convolution, pooling) alone produces a combinatorial problem that grows exponentially with the increase in depth, and that is without taking into account layer hyperparameters (filter size, stride, channel count) and weights. Connectivity patterns add another dimension of complexity, as structures such as skip connections and parallel branches have been found beneficial in manually designed models [3]. For this reason many recent architecture optimization algorithms, starting with NAS-Net by Zoph *et al.* [34], utilize a constrained search space based on repeated structural motifs. Instead of searching for the architecture of the entire network, they instead work with *cells*, which are small subnetworks containing only a few layers. The target network is then constructed by repeating the cell a given number of times. This relaxation allows the cells to have almost arbitrary structures while maintaining the viability of the search. Another advantage is the directly controllable trade-off between the network power and complexity by varying the number of cell repetitions. It is common to speed up the search by using less cells and then increase their number for the final evaluation of the discovered architecture. However, the unavoidable natural drawback of this approach is the fact that only a small subset of network design space is reachable with such constraints, and potentially better architectures may not be discoverable. Therefore, we opt for optimizing the whole network simultaneously.

We model a deep neural network as a multivariate random variable coming from a known probability distribution. For the sake of tractability, we consider only the choices of layer types for optimization, resulting in a discrete distribution, while other hyperparameters are not directly tuned by the search procedure. Specifically, we bind the values of filter sizes and strides with the layer type choices and set the channel count to an externally defined constant for all the layers. We denote the set of possible layer types as L and call it *the layer library*. For the purpose of this work, we include the following ten common operations in the library (each with the corresponding shorthand notation):

id	identity (output is equal to input),
c1	1x1 convolution,
c3	3x3 convolution,
c5	5x5 convolution,
c7	7x7 convolution,
d3	3x3 dilated convolution (with dilation rate of 2),
d5	5x5 dilated convolution (with dilation rate of 2),
m2	2x2 max pooling,
m3	3x3 max pooling (with stride 2),
a3	3x3 average pooling (with stride 2).

We assume that the choice of each layer in a CNN is independently distributed. While this assumption is unlikely to hold in practice, it simplifies the formulation, and inter-layer interactions are implicitly taken into account during the search. Multivariate generalizations of the proposed method can potentially offer improvements and are a promising future work direction. Given our assumption, a discrete distribution of network structures can be represented as a matrix of probabilities P, where each row describes a layer and $P_{ij} \in [0, 1]$ is the probability of i-th layer being the j-th layer type from L. Matrix P is henceforth called *prototype*. The dimensions of P are $N \times |L|$, where N is the current number of layers in the network.

The probabilistic representation has a number of advantages over the population of networks. Matrices have a wide range of available optimization approaches; many existing optimization heuristics outside of the scope of this work are straightforwardly applicable to the proposed representation. The prototype offers intuitive insight into the anytime state of the search, as the probability mass is always explicitly assigned for every point of the design space. The convergence of the search is easy to determine by how close the layer probabilities are to their extremes. Finally, the proposed representation can offer significant implementation advantages in a distributed setting, as only a small prototype matrix needs to be transferred between computational nodes, rather than full-scale models.

A limitation of the proposed representation is the fact that evaluating the prototype can be done only by sampling networks from it and training them from scratch. To minimize the sampling error, the number of samples has to be large, which can incur particularly high computational costs (especially for large values of N). While a number of techniques to minimize the evaluation costs exists, few of them are suitable for the prototype representation; for instance, due to the large variety of possible structures (of differing sizes), sharing weights between them is not practical. We consider options to address this in the analysis section.

B. Search Algorithm

To construct an iterative architecture search algorithm with the above representation, three elements need to be defined - initialization, update and stopping condition. The proposed algorithm, denoted ASED, operates on a single prototype for the sake of simplicity. The depth of all networks on a given search step is the same due to the fixed prototype dimensions; to search across architectures of different sizes, we gradually increase the depth after each update step. While the prototype rows are never removed, the inclusion of identity in our layer library means that, in practice, networks with less than N layers can be represented and discovered at any search stage.

The prototype P is initialized as a $N_{init} \times |L|$ matrix with every element set to $1/|L|$, where N_{init} is a starting layer count. While a more specific prior can be given, the uniform initial distribution ensures that every reachable architecture is equally likely to be considered, which helps to emphasize early exploration. The choice of N_{init} should be carefully considered, as a small value can result in premature convergence without sufficiently exploring the larger portion of the design space, but a large value can cause the search to be ”lost” unless an impractically large number of samples is evaluated (due to the curse of dimensionality).

To update the prototype, sampling of K candidate networks is performed first, with each layer independently selected from the discrete distribution given by the corresponding row of the prototype matrix. Each candidate model is then trained and evaluated on the target problem, and the temporary population is sorted by validation performance. While we evaluate and track the classification accuracy, we opt for another, additional measure to compute the candidate ranking. We adopt the multi-class Matthews coefficient [62], which is designed to be robust to the class imbalance in the data, allowing the search to operate reliably in such cases. The formula of the multi-class Matthews coefficient is as follows:

$$m = \frac{\sum\limits_{klm} (C_{kk}C_{lm} - C_{kl}C_{mk})}{\sqrt{\sum\limits_{k}(\sum\limits_{l} C_{kl})(\sum\limits_{\substack{l' \\ k' \neq k}} C_{k'l'})}\sqrt{\sum\limits_{k}(\sum\limits_{l} C_{lk})(\sum\limits_{\substack{l' \\ k' \neq k}} C_{l'k'})}}, \quad (1)$$

where C is a confusion matrix. The value of the multi-class Matthews coefficient ranges between a data-dependent negative value (≥ -1) and $+1$. Whenever the denominator of the fraction in Equation 1 is zero, we set the output value to the lowest possible: -1.

Given the ranking, the best $K_s < K$ models are then selected to directly induce the new prototype, which, due to the independence assumption, takes the following form:

$$P_{ij} = \frac{1}{|K_s|} \sum_{k=1}^{K_s} x_{kij}, \quad (2)$$

Algorithm 1 Architecture Search by Estimation of Network Structure Distribution (ASED)

```
1: Input: $L$, $N_{init}$, $t_{max}$, $K$, $K_s$, $n(t)$
2: $N \leftarrow N_{init}$
3: for $i \in \{1, \dots, N_{init}\}$, $j \in \{1, \dots, |L|\}$ do
4:     $P_{ij} \leftarrow 1/|L|$
5: end for
6: for $t \in \{1, \dots, t_{max}\}$ do
7:     Sample $K$ candidate networks from $P$
8:     Train and evaluate candidate networks
9:     Sort candidate networks by validation performance
10:    $S \leftarrow K_s$ best performing candidate networks
11:    Recompute $P$ based on $S$ (Eq. 2)
12:    Add $n(t)$ new rows to $P$
13:    for $i \in \{N+1, \dots, N+n(t)\}$, $j \in \{1, \dots, |L|\}$ do
14:        $P_{ij} \leftarrow 1/|L|$
15:    end for
16:    $N \leftarrow N + n(t)$
17:    if $\forall i,j\ P_{ij} \in \{0, 1\}$ then
18:        break
19:    end if
20: end for
21: return $P$
```

where x_{kij} is an indicator variable that is equal to 1 if k-th selected candidate network has j-th library item as i-th layer, and 0 otherwise. This update step is equivalent to the one used in the UMDA algorithm [41]. Every update is followed by an addition of one or more rows to the prototype, according to the predefined schedule (denoted $n(t)$). These new layers are initialized with a uniform distribution. Note that we avoid explicitly preserving the best candidates between updates (the technique known as elitism). As our approach starts from the solution space of low dimensionality and gradually increases it, the bias towards early dominant solutions will result in premature convergence. The complete description of the ASED procedure is given in Algorithm 1. The search stops when the specified iteration limit t_{max} is reached or all the values of the prototype matrix become strictly 0 or 1, which indicates complete convergence. A single network architecture with the highest probability is selected to be the final output.

C. Convergence Control Techniques

While the described search procedure navigates the search space by progressively narrowing down the region under consideration and should be capable of avoiding local optima, it can still get stuck in a local optimum and hence exhibit premature convergence. As the search progresses, individual layer type probabilities tend to approach either 0 or 1 regardless of their immediate impact on the network performance, as is established in the theory of EDAs [63], [64]. The proposed algorithm does not allow for any mechanisms to limit this; in fact, such an effect is desirable for the search convergence. Moreover, once a probability has achieved the value of exactly

0 or 1, it becomes fixed and will not change thereafter, as all of the sampled networks will be the same with respect to the type of the corresponding layer. In case of 0 the corresponding operation is no longer considered, while in case of 1 the layer choice is made permanent, meaning that the dimensionality of the problem is essentially reduced from that point on. A subset of network structures becomes unreachable, which can be beneficial for navigating the design space, but can also mean the loss of potentially better solutions. We consider two different techniques to address this issue.

A common technique in EDAs involves capping the probabilities, such that extreme values are not achievable and each element instead spans the predefined range $[p_{min}, p_{max}]$, where $p_{min} > 0$, $p_{max} < 1$. In our setting, this means that there is always at least the probability of p_{min} for each layer type to be selected in any position, removing irreversible choices. Probability capping is implemented by simple row-wise proportional normalization of the prototype matrix after every prototype update step. We adopt the approach where the upper cap p_{max} is explicitly given as a parameter and the lower cap is then computed as

$$p_{min} = \frac{1 - p_{max}}{|L| - 1}. \quad (3)$$

The normalization itself is then performed as follows:

$$p'_{ij} = \begin{cases} p_{min} & \text{if } p_{ij} \leq p_{min} \\ p_{ij} \cdot m_i & \text{if } p_{ij} > p_{min} \end{cases}$$
$$m_i = \frac{\sum B_i + \sum S_i - |S_i| \cdot p_{min}}{\sum B_i}, \quad (4)$$

where $S_i = \{p_{ij} | p_{ij} \leq p_{min}\}$ and $B_i = \{p_{ij} | p_{ij} > p_{min}\}$ for $i \in \{1, \ldots, N\}$. Note that B_i cannot be 0 as the layer probabilities sum to 1.

Another way to prevent the search from prematurely converging is to additionally modify the prototype between iterations. One can, for instance, apply a small random perturbation, similar to how mutation is used in evolutionary algorithms. However, due to the search being driven by sampling, the effect of such mutation would be either insignificant or highly unpredictable. Instead, we opt for another operation, which we call *prototype inversion*, that replaces high probabilities with low values and vice versa. This prompts the search to explore exactly the previously discarded regions of the search space, while the currently dominant choices become extremely unlikely (the latter aspect evokes similarities to the well-known tabu search, which is an optimization technique that explicitly forbids the reuse of already seen solutions [65]). Naturally, such an inversion operation is highly destructive and can prevent the search from progressing, so it needs to be executed only at some iterations of the algorithm. Additionally, we save the current prototype just before inverting it to make sure the information is not lost, which essentially means the ASED algorithm can produce multiple solutions before the stopping condition is met.

To establish an inversion condition, we need to find the measure of convergence, as performing the inversion too early and/or too often can hinder the search process. L_2-norm of the probability vectors is suitable for this purpose, as it spans the interval $[1/\sqrt{|L|}, 1]$. Here, the lower bound corresponds to the uniform distribution and the upper bound is achievable only when a single element (layer type) takes value 1 with every other being 0. The L_2-norm of each prototype row is thus a measure of the certainty of the layer choice and increases as the search progresses. The condition for triggering the prototype inversion can then be a threshold on the L_2-norm of the prototype, averaged over all the rows (as they are assumed independent). If the inversion is used, this condition is checked at every iteration after the prototype is updated. The newly added uniformly distributed rows are ignored for the purposes of the mean norm calculation.

The inversion operation is implemented by subtracting each probability value from 1 (e.g., 0.85 becomes 0.15). For probabilities that have taken values of 0 or 1 this operation does not have the intended effect, as they still limit the exploration space. To suppress these extreme values, the inversion is followed by the probability capping normalization defined in Eq. (4), using the same parameters p_{max} and p_{min}. As no probability goes to zero as a result of inversion, previously reachable solutions remain reachable, allowing for potential backtracking. We consider two types of inversion operation: the *full inversion* and the *partial inversion*. The former is applied row by row to the whole prototype. The latter is less destructive as it only applies to the subset of prototype rows which have the highest L_2-norm. The specific number of such rows is empirically set to $\lfloor \sqrt{N} \rfloor$. The partial inversion thus applies only to some of the most converged layers, preserving less confident choices as they are.

As both described techniques are simple mathematical operations on the prototype matrices, they do not incur significant computational costs by themselves. However, as they influence the search behaviour of the algorithm towards slowing down the convergence, more iterations may be required to reach the solution of the same level of complexity as with the baseline variant. With respect to the given stopping condition, the proposed techniques can indirectly increase the overall running time of the algorithm, although the specific impact can be evaluated only empirically on a case-by-case basis.

D. Non-Linear Layer Connectivity

The default prototype specification supports only the simplest architectures, where the flow of the input data through the layers is strictly sequential. However, connections between non-adjacent layers, also known as *shortcuts*, play an important role in the powerful CNN models, such as ResNet [2] and DenseNet [3]. Shortcuts counteract the problem of vanishing gradients by ensuring the efficient flow of the backwards propagated signal, which allows models of much larger depth to be reliably trained. While we constrain layer counts during the architecture search due to computational limitations, the use of shortcuts still simplifies and speeds up the training of candidate models as the depth increases. By making deeper models more competitive against shallow ones already in the early training stages, shortcuts discourage premature convergence of the prototype. Thus, while implicit, they grant similar benefits as the techniques described in the previous subsection. Of course, our framework also benefits from shortcuts via increased expressiveness and generality.

There are several options for introducing shortcuts to the proposed formulation. One possibility is to define another prototype matrix, which would store the connectivity patterns, i.e., presence or absence of a connection, between layers. This additional prototype can be operated on with any procedure suitable for the original, such as the one given in Algorithm 1. The two matrices can be concatenated and optimized jointly, or they can be iterated between, introducing an additional step to the search procedure. It is worth noting, however, that such an approach significantly increases the problem complexity. Additional dimensions of the search space would require either exponentially more samples (in the joint case) or exponentially more iterations (in the iterative case) to ensure sufficient exploration. As a compromise between complexity and expressivity, we consider another approach — *fixed shortcut patterns*. We design simple shortcut-generating rules, inspired by existing deep CNN models, and apply them to any sampled candidates throughout the search, as well as to the ultimately discovered architectures (while ensuring the solutions remain valid). As the rule does not change over time, the search procedure can be expected to select structures which take the most advantage of the given shortcut pattern. This simple approach does not incur any additional computational costs. In fact, it can potentially speed up the training of the deeper networks, but this is not currently taken advantage of, as we use fixed training schedules for simplicity.

We consider two types of shortcut patterns — the *residual pattern* and the *semi-dense pattern*. Both can be characterized by a single parameter $D \geq 1$. *Residual pattern* is inspired by the residual connections of ResNet [2]. In this case, the shortcut connects the output of the current layer to the output of the layer which is D positions after (e.g., the following layer if $D = 1$). Residual shortcuts cannot intersect or overlap, i.e., there can be no starting point between another starting point and its corresponding endpoint. In the case of the *semi-dense pattern*, every group of D consequent layers have their outputs connected to the single endpoint at the output of the $(D + 1)$th layer. It is inspired by the DenseNet [3] with some simplification (the original pattern would have all the layers within the group connected). These shortcut patterns are illustrated in Fig. 1.

As in other deep models, the signal arriving via a shortcut is combined with the primary signal at the endpoint by simple addition. As in the early ResNet, we follow the post-activation pattern, i.e., the activation function is applied after signals are combined (but batch normalization, if applicable, precedes the signal combination). A notable issue for the proposed approach is the compatibility of the signal dimensions. In handcrafted architectures, shortcuts are typically applied within layer blocks that do not perform downsampling, and, therefore, the shapes of the primary and the shortcut signals always match. This does not necessarily hold within our framework: while convolutions are zero-padded to preserve dimensions between input and output, any layer can potentially be a pooling layer, breaking the compatibility. We resolve

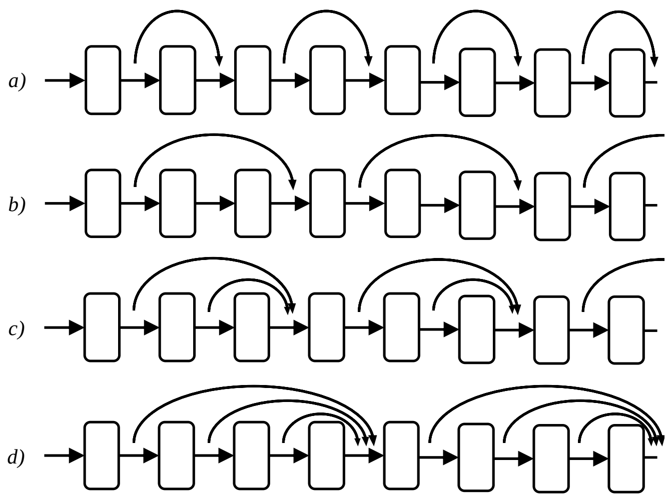

Fig. 1. Illustrated shortcut patterns: a) both, $D = 1$; b) residual, $D = 2$; c) semi-dense, $D = 2$; d) semi-dense, $D = 3$.

this issue by keeping track of the pooling operations that the primary signal is undergoing and applying them to the shortcut signal before adding the two together. Compatibility with regards to the channel number is guaranteed, since it remains constant throughout the network, with the exception of the original input, which is not allowed to be the starting point of a shortcut. It is also worth noting that, since our framework allows every layer to potentially represent an identity function, the span of some of the shortcuts may in practice be less than D.

IV. Experimental Results and Analysis

In this section, we describe the experimental setting and obtained results and discuss their implications for this and future work.

A. Experimental Setting

Following the previous works of the field, the proposed ASED algorithm is validated in the image classification setting. We consider two datasets of differing difficulty and scale. As a simpler problem we utilize USPS dataset [66], which contains grayscale samples of handwritten digits of 16x16 pixels each. USPS has 10 classes, 7291 training examples, and 2007 test examples. The larger problem is CIFAR-100 dataset [67], which is one of the standard NAS benchmarks. It consists of 3-channel 32x32 RGB images. It contains 50K training examples and 10K test examples of 100 different classes. To allow for evaluation of candidate networks without leaking the test data, for both datasets we sample 20% of the original training images, maintaining class balance, to obtain validation sets. We use the original test sets only to report the performance of the final discovered model. Classification accuracy is used as the performance metric. We do not use any preprocessing for USPS. For CIFAR-100 the standard preprocessing procedure is applied: images are zero-padded by 4 pixels on each side, followed by random cropping down to 32x32 and horizontal flipping with probability 0.5, as well as normalization to zero mean and unit variance.

The algorithm parameters are set as follows. The search is initialized by sampling 10K networks from the uniform 5-layer prototype, which covers 10% of the design space (as the current library permits 10^5 possible 5-layer networks). At every iteration of the search, $K = 1000$ networks are sampled, trained, and ranked, with the top $K_s = 100$ forming the next prototype. As adding layers is initially affordable, but becomes more expensive later, the following growth schedule is adopted:

$$n_t = \begin{cases} 2 & \text{if } t \in \{1, 2\} \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

For USPS dataset we additionally consider a modified set of settings to start from smaller networks, which allows for more gradual navigation of the search space in the early stages. With the modified search settings a 2-layer uniform prototype is used as a starting point (instead of a 5-layer one). The sampling between iterations is also reduced: 100 networks are generated and top 10 are selected to induce the next prototype. This results in the smaller number of candidates being evaluated, speeding up the search by approximately a factor of ten.

During the search, the channel count of all convolutional layers is set to 32. To avoid the mismatch of signal dimensions, every convolutional layer has its output padded to match the input; therefore, only the pooling layers can perform down-sampling. We use PReLu as an activation function and apply the corresponding initialization policy of He *et al.* [68]. To make the discovered architectures output the class predictions, we perform global average pooling after the last sampled layer, followed by a fully connected layer of 100 PReLu-activated neurons, dropout with rate 0.5, and a softmax layer. Batch normalization is not used within the candidate networks during the search; however, the final discovered architecture has batch normalization applied after every convolutional layer to maximize evaluation performance.

Training numerous deep networks from scratch incurs the majority of computational expenses of the search procedure. We therefore use a different, less intensive training regime, denoted as *brief training*, to produce metrics for candidate ranking during the search. *Full training* is reserved only for the evaluation of the final solution discovered by the architecture search. Both settings use stochastic gradient descent (SGD) with momentum of 0.9 to minimize the cross entropy loss. Brief training runs for 20 epochs, using the learning rate of 10^{-2} for the first 10 epochs and 10^{-3} thereafter. Full training runs for 200 epochs and uses the following schedule of learning rates: 0.01 for epochs 1-60, 0.02 for epochs 61-120, 0.004 for epochs 121-160, and 0.001 for epochs 161-200. Full training also uses L_2-norm weight regularization with the coefficient of 10^{-4} (excluding PReLU weights, as recommended in [68]) and imposes the maximum L_2-norm constraint of 0.5 on weights. The batch size is set to 64 for USPS and 128 for CIFAR-100.

The settings for convergence control and non-linear connectivity variants are given as follows. For the variants that use probability capping we set p_{max} to 0.9. The L_2-norm threshold for both full inversion and partial inversion is set to 0.65, as that corresponds to the middle of the interval of the possible values. For the evaluation of shortcut-generating rules we consider $D = 2$ and $D = 3$, as $D = 1$ has limited impact on the gradient flow and $D > 3$ results in too few shortcuts created, given the network depth limitation. We do not consider the convergence control and the shortcut patterns jointly: since both of them act like regularizers on the search procedure, their significance is easier to analyze separately. The total number of iterations is $t_{max} = 9$ for the baseline and probability capped variants, which, under the adopted schedule, corresponds to the maximum layer count of 16. However, for the inversion experiments the search is run further to allow for observing the results after multiple instances of inversion triggering. The achievable layer count is 22 for these variants.

All of our experiments are implemented in PyTorch and performed on a workstation with 4 GeForce 1080Ti GPU units. The running times are included with the reported results. The source code will be made publicly available on GitHub upon acceptance of the paper.

B. Architecture Search Performance on USPS

To the best of our knowledge, the current state of the art results on USPS are achieved by Oyedotun et. al. [69]. Their solution is based on a well-known ResNet architecture, modified by a number of techniques: maxout activation function, elastic net regularization, and feature standardization. Furthermore, the final classification is obtained via an SVM that is trained on the features from the final convolutional layer. We compare the reported results from [69] with the solutions discovered by the baseline ASED in its default and modified configurations. The comparison between ResNet and both ASED setups is given in Table I. The discovered architectures are shown in Fig. 2, using the notation defined in Section III.

The networks discovered by ASED clearly produce competitive results to modified S-ResNet. While the default search configuration shows a clear tradeoff between classification accuracy and the number of parameters, the modified search configuration can produce very competitive architectures with just 16 channels. The discovered architecture is significantly smaller than S-ResNet in both depth and overall memory requirement, while achieving a similar test error rate. ASED is, therefore, capable of discovering very efficient architectures in smaller problem domains. The discovered architecture notably differs from common patterns of handcrafted designs, lacking pronounced block-like repeating patterns and taking advantage of convolutions with larger kernels. This shows the potential of the neural architecture search to automatically discover novel and potentially superior network structures, which may lie outside the standard design spaces.

C. Architecture Search Performance on CIFAR-100

We report the results of running the proposed ASED algorithm on CIFAR-100 with the default search configuration

TABLE I
COMPARISON OF THE ERROR RATES (FULL TRAINING MODE) ON USPS DATASET

Model	Test error (%)	Parameter count	Model depth	Search time (GPU hours)
Maxout S-ResNet+ENR+SVM [69]	2.34	169K	54	N/A
Maxout S-ResNet+ENR+FS+SVM [69]	2.19	169K	54	N/A
ASED default, 16 channels	2.49	77K	9	64.0
ASED default, 32 channels	2.25	305K	9	64.0
ASED modified, 8 channels	2.64	12K	10	6.4
ASED modified, 16 channels	2.25	46K	10	6.4
ASED modified, 32 channels	2.25	182K	10	6.4

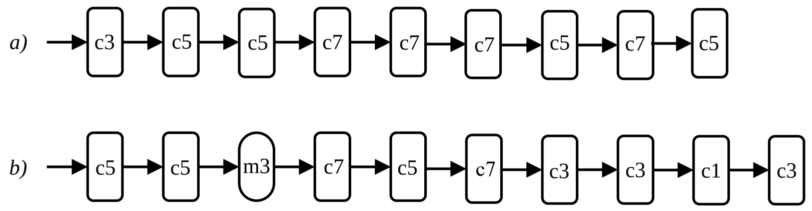

Fig. 2. Best discovered structures on USPS for a) default settings, b) modified settings. Shorthand notation from Section III. Pooling operations denoted by oval shapes.

described above. In addition to the baseline given by Alg. 1, we also evaluate the proposed modifications of the search: probability capping (denoted ProbCap for clarity), the full inversion variant, the partial inversion variant, and the shortcut-generating rules of both types.

Fig. 3 shows the distributions of candidate models' performances on the validation set throughout the search iterations, under the brief training regime. While limiting the number of training epochs does allow for much faster evaluations during the search process, the performance estimates obtained this way are pessimistic and not fully representative of the underlying models' capabilities. Therefore, we also conduct an extended evaluation of the best discovered networks by adding batch normalization modules and using the full training regime (see Section IV-A). In this comparison, we also include the best architecture from the initialization sample (generated from the 5-layer uniform prototype), as well as the best 16-layer architecture from 1000 networks generated from a uniformly random prototype. Every configuration is trained from scratch with the convolution channel counts of 32, 64, 128, and 256. The results are reported in Table II. The indicated layer count excludes any identity layers. The best-performing network structures from each algorithm variant are shown in Fig. 4.

Modified variants of ASED prove superior to the baseline version, with the notable exception of ProbCap, which appears to stagnate at an earlier point of the search, leaving it with the smallest depth. The shortcut-based variants overall outperform other solutions for lower channel counts, but the inversion-based variants are the most efficient for larger models (in terms of both accuracy and the number of parameters). This result shows that perturbing the prototype, even very aggressively, can lead to discovering better solutions with the same or smaller depth. The shortcuts are essential in reaching deeper models without premature convergence; such models are immediately more powerful and offer short-term performance advantages, but their larger sizes imply higher memory and computation requirements.

The experiments also confirm that the brief training regime significantly underestimates the model performance. This disparity in itself poses no problems for the algorithm, as the search depends on a relative performance estimate, not the specific numeric values. However, comparing the data from Fig. 3 and Table II makes it clear that the candidate ranking is to some extent distorted by the brief training regime. Baseline algorithm obtains the highest validation performance during the search, but is notably inferior in the final evaluation on the test set. The validation performance produced by the brief training appears to be misleading when comparing different search variants, as the ones that appear superior (in both search speed and solution accuracy) are exactly those which are weaker in the test set evaluation. While within the same variant the algorithm does steadily improve the quality of its solutions over time, the possibility remains that some promising intermediate structures are ranked too low to influence the prototype update and are discarded. Tackling this issue in the future work would likely result in better model discovery. While full training is not computationally viable in the late stages of the search when the models become more complex, low-fidelity training allows for alternative, potentially more stable implementations. These include, for example, training on the subset of the data or training with a smaller channel count. More sophisticated solutions include dynamic allocation of the training budget to prioritize more promising architectures (see e.g. [75]) or predicting the candidate performance from its properties without training it (similar to [36]). All of these options have also an additional benefit in speeding up the evaluation process, which is the slowest step in the ASED algorithm.

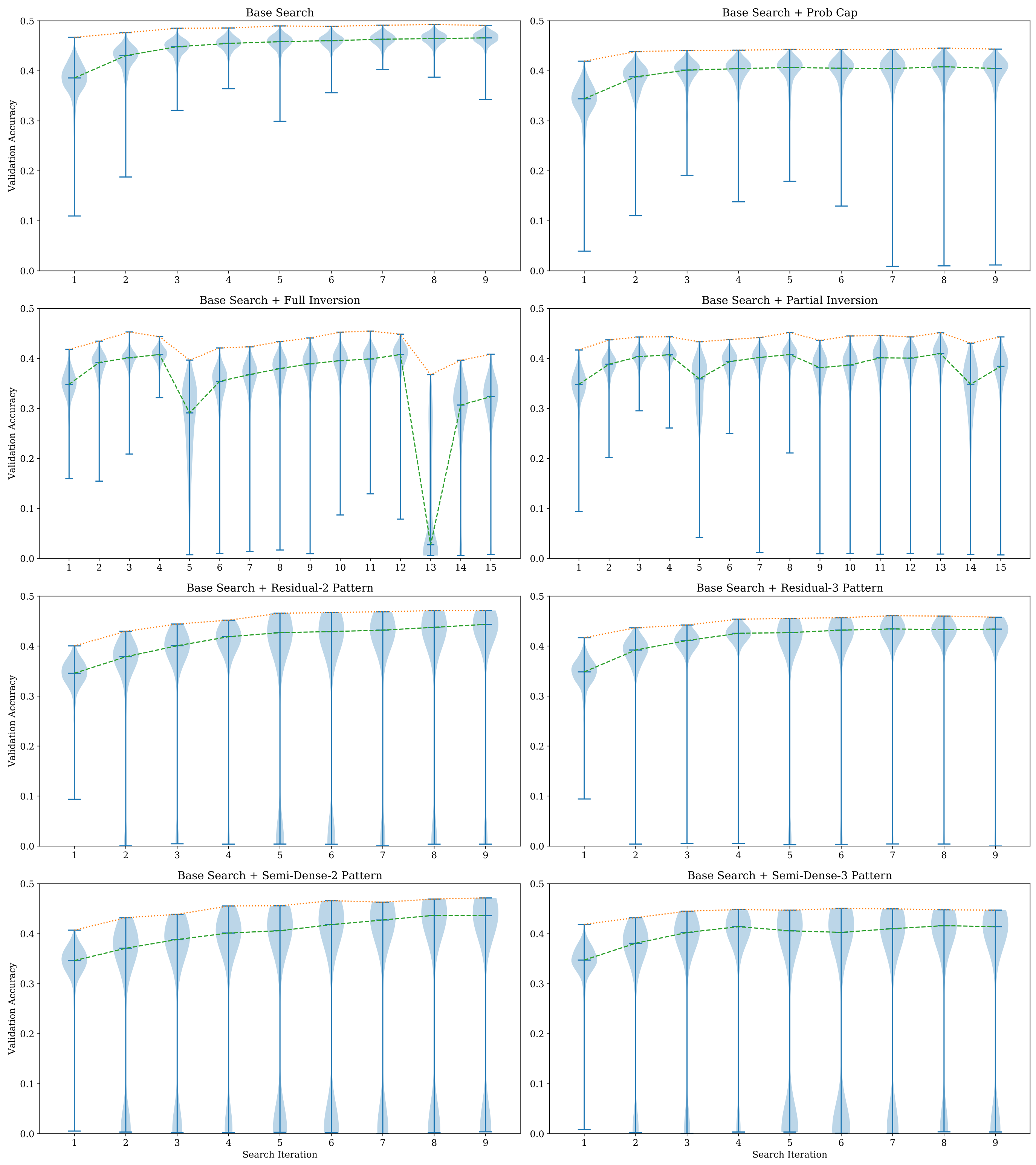

Fig. 3. Distributions of validation accuracies (brief training mode) on CIFAR-100 across search iterations. The line plots show the maximum and median accuracies of the sampled networks.

Another important observation can be made about the practical model depth achieved by different algorithm variants. If the evolution of the prototype reaches a point where the newly added layer assigns the largest probability to an identity operation, adding further layers is unlikely to introduce structural novelty, as they would also tend to converge to identities. The search essentially stops at that point, as the depth does not increase further. All of the algorithm variants that do not use shortcuts demonstrate this behaviour, although the inversion variants are capable of avoiding it to a limited

TABLE II
COMPARISON OF BEST DISCOVERED ARCHITECTURES BY THEIR TEST ACCURACY (FULL TRAINING MODE) ON CIFAR-100 DATASET

Model Source	Layer Count	32 channels		64 channels		128 channels		256 channels	
		Acc.	Par.	Acc.	Par.	Acc.	Par.	Acc.	Par.
Initialization	5	0.4898	131K	0.5835	513K	0.6419	2.0M	0.6846	8.1M
Random uniform	15	0.5794	223K	0.6621	879K	0.7200	3.5M	0.7499	13.9M
ASED	10	0.5659	224K	0.6582	886K	0.7102	3.5M	0.7483	14.1M
ASED + Prob Cap	8	0.5483	190K	0.6330	751K	0.6963	3.0M	0.7442	11.9M
ASED + Full Inversion	12	0.5827	268K	0.6728	1.1M	0.7297	4.2M	**0.7729**	16.9M
ASED + Partial Inversion	11	0.5748	236K	0.6641	932K	0.7249	3.7M	0.7652	14.8M
ASED + Residual-2	16	0.6396	419K	0.6872	1.7M	0.7282	6.6M	0.7485	26.5M
ASED + Residual-3	13	0.6194	367K	0.6689	1.5M	0.7068	5.8M	0.7315	23.2M
ASED + Dense-2	15	**0.6461**	392K	**0.6984**	1.6M	**0.7398**	6.2M	0.7610	24.8M
ASED + Dense-3	11	0.6092	251K	0.6678	1.0M	0.7084	3.9M	0.7348	15.8M

TABLE III
TEST PERFORMANCE COMPARISON OF ESTABLISHED AND AUTOMATICALLY DISCOVERED ARCHITECTURES ON CIFAR-100 DATASET

Method	Accuracy (%)	Parameter count	Model depth	Search cost (GPU days)
FractalNet [70]	76.7	38.6M	21	N/A
Shake-Shake [71]	**84.2**	26.2M	26	N/A
Wide ResNet 28-10 [72]	80.4	36.5M	28	N/A
DenseNet-BC [3]	82.8	25.6M	190	N/A
Genetic CNN [28]	70.9	–	17	17
MetaQNN [73]	72.9	11.2M	9	100
Large Scale Evolution [29]	77.0	40.4M	≥ 13	≥ 2600
SMASH [55]	79.4	16M	211	1.5
Hill Climbing [56]	76.6	22.3M	30	1
NSGA-NET-128 [74]	79.3	3.3M	21	8
NSGA-NET-256 [74]	80.2	11.6M	21	8
ASED (best)	77.29	16.9M	12	20

extent by resetting the probability of identity to a low value for previously discovered layers. This effect can once again be traced back to the brief training regime and the performance estimates it produces. Fig. 3 shows how compressed the range of the estimated accuracy is. The short training schedule biases the search towards architectures that show the fastest improvement in early epochs. As a result, ASED exhibits *low complexity bias*. This can be advantageous in specific cases, but it makes the architectures of higher complexity, which are naturally slower to train, much less competitive and less likely to be retained. Shortcut-based ASED circumvents this problem by making the deeper models easier to train and, therefore, more competitive. The search can thus explore more complex solutions without the induced ”limit” to the layer count. However, shortcut-based ASED is still to some extent affected by the bias towards simpler architectures: Table II shows that the superiority of the models with shortcuts is limited to the smaller channel counts, which are closer to the low-fidelity evaluation setting. While the low complexity bias limits the exploration of high-dimensional structures, it can be turned into an advantage in the appropriate problem setting, e.g., if the goal is to find specifically the simplest, fastest to train models.

The best discovered architectures (see Fig. 4) can be compared with common handcrafted designs, as well as with networks using block-based representation. The solutions found by ASED have pooling layers distributed roughly regularly along the network depth. This draws parallels to other common designs, where pooling (more generally, downsampling) typically separates different network submodules/blocks. However, it is important to note that the architectures discovered by ASED do not contain repeating sequences of operations and, therefore, cannot be represented with just a few recurring patterns. These final solutions lie outside of the search space of many other architecture search methods, which demonstrates the value of generality of the proposed representation. Another notable difference between the proposed and existing solutions is the tendency for the convolution kernel size to increase along the network depth. Handcrafted designs most commonly feature larger convolutions in the earlier stages, or simply use the same kernel size for all the layers. In the ASED-produced solutions, on the other hand, 7x7 convolutions and dilated 5x5 convolutions (with an effective receptive field of 9x9) are dominant in the later layers.

Table III presents the comparison between the ASED algorithm (the full inversion variant, due to the highest overall performance) and existing solutions with reported results on CIFAR-100. Handcrafted architectures are listed in the first

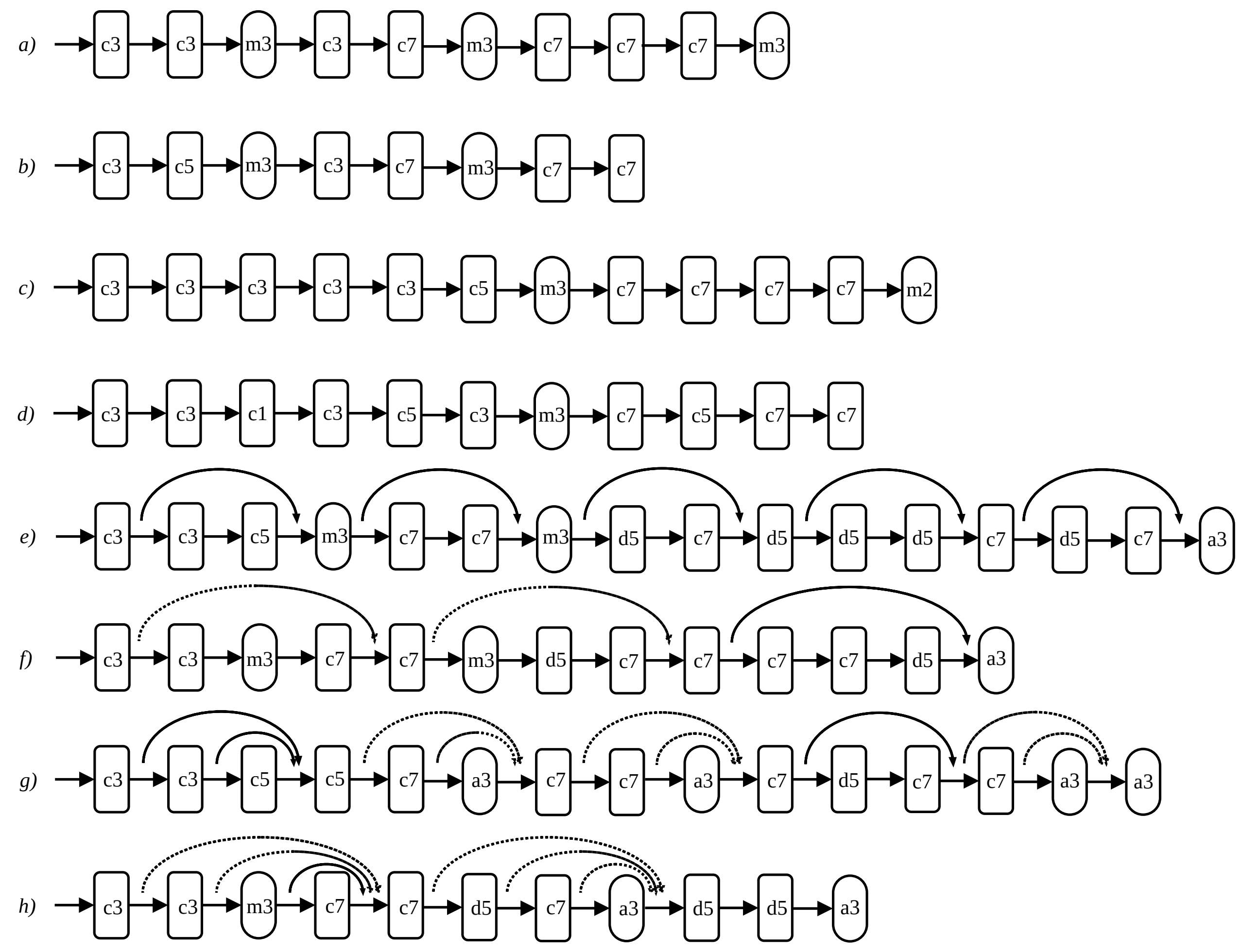

Fig. 4. Best discovered structures for a) base procedure, b) probability capping, c) full inversion, d) partial inversion, e) residual-2 pattern, f) residual-3 pattern, g) semi-dense-2 pattern, h) semi-dense-3 pattern. Shorthand notation from Section III. Pooling operations denoted by oval shapes. Dotted lines indicate skip connections with pooling.

4 rows of the table. The achieved accuracy is competitive, given the simplicity of the formulation, its interpretability, and many possibilities for extension. Moreover, further increasing the maximal depth of the search is certainly promising; it can be achieved by taking advantage of the shortcuts and, possibly, inversion at the same time. While the computational costs of ASED are relatively high due to sampling and evaluation stages, it is worth noting that the algorithm's implementation can scale almost linearly in the parallel computation setting. This is due to the fact that each candidate model is processed independently from others. In such a scenario, the model parameters (weights) are also never transferred between distributed workers; only the prototype matrix and the evaluation outcomes need to be exchanged.

V. Conclusion

The automated neural architecture design is growing in importance as the application-driven demand outpaces the available expertise and resources. In this paper, we proposed a probabilistic representation of the deep network structure and defined an architecture search algorithm, named ASED, that has the advantages of being intuitive, easy to interpret and analyze, as well as readily extensible to incorporate many well-known elements of the field. While the proposed optimization approach is simple and limited in scale with respect to depth and computation, it is already capable of discovering competitive and novel architectures, compared to existing methods with much higher complexity. Computationally ASED benefits from the ease of parallel implementation, due to the candidate networks being processed independently. ASED is limited by the reliance on the low-fidelity training, which introduces a low complexity bias. While it is considered an undesirable property, it can also prove beneficial in preventing the solutions from overfitting, as well as some other cases, such as optimizing under heavy computational constraints.

The prototype-based approach allows for many promising directions for future work. The underlying search mechanism can incorporate many of the developments in the area of EDAs, such as the multiobjective formulation [76] or the use of historical information [77]. The network representation can be further enriched by introducing novel layer operators, as well as explicitly encoding more hyperparameters and structural patterns. Running the proposed algorithm on established architecture search spaces could also provide novel solutions.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 5 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[3] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *Advances in Neural Information Processing Systems 28*, pp. 91–99, 2015.

[6] G. Hinton *et al.*, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, 2012.

[7] B. Lee, S. Min, and S. Yoon, "Deep learning in bioinformatics," *Briefings in Bioinformatics*, vol. 18, no. 5, pp. 851–869, 2016.

[8] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research (JMLR)*, vol. 15, pp. 1929–1958, 2014.

[9] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[10] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," in *Advances in Neural Information Processing Systems 29*, 2016, pp. 901–909.

[11] C. Yun, S. Sra, and A. Jadbabaie, "Global Optimality Conditions for Deep Neural Networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[12] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 6389–6399.

[13] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, "Understanding Batch Normalization," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 7694–7705.

[14] Z. C. Lipton, "The Mythos of Model Interpretability," in *ICML 2016 Workshop on Human Interpretability in Machine Learning (WHI 2016)*, 2016.

[15] A. S. Charles, "Interpreting Deep Learning: The Machine Learning Rorschach Test?" *arXiv preprint arXiv:1806.00148*, 2018.

[16] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[17] A. Nguyen, J. Yosinski, and J. Clune, "Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 427–436.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 630–645.

[19] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, 2016.

[20] A. Muravev, J. Raitoharju, and M. Gabbouj, "On the Layer Selection in Small-Scale Deep Networks," in *7th European Workshop on Visual Information Processing (EUVIP)*, 2018.

[21] T. Tommasi, N. Patricia, B. Caputo, and T. Tuytelaars, "A Deeper Look at Dataset Bias," in *Domain Adaptation in Computer Vision Applications*, G. Csurka, Ed. Cham: Springer International Publishing, 2017, pp. 37–55.

[22] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, 1990.

[23] Xin Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[24] K. O. Stanley and R. Miikkulainen, "Evolving Neural Network through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002.

[25] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks," *Artificial Life*, vol. 15, no. 2, pp. 185–212, 2009.

[26] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Evolutionary artificial neural networks by multi-dimensional particle swarm optimization," *Neural Networks*, vol. 22, no. 10, pp. 1448–1462, 2009.

[27] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[28] L. Xie and A. Yuille, "Genetic CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1388–1397.

[29] E. Real *et al.*, "Large-Scale Evolution of Image Classifiers," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2902–2911.

[30] H. Zhang, S. Kiranyaz, and M. Gabbouj, "Finding Better Topologies for Deep Convolutional Neural Networks by Evolution," *arXiv preprint arXiv:1809.03242*, 2018.

[31] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 4780–4789, 2019.

[32] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving Deep Convolutional Neural Networks for Image Classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, pp. 394–407, 2019.

[33] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[34] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[35] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient Neural Architecture Search via Parameter Sharing," in *Proceedings of the 35th International Conference on Machine Learning (PMLR)*, 2018, pp. 4095–4104.

[36] C. Liu *et al.*, "Progressive Neural Architecture Search," in *European Conference on Computer Vision (ECCV)*. Cham: Springer International Publishing, 2018, pp. 19–35.

[37] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient Multi-objective Neural Architecture Search via Lamarckian Evolution," in *International Conference on Learning Representations (ICLR)*, 2019.

[38] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable Architecture Search," in *International Conference on Learning Representations (ICLR)*, 2019.

[39] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the Search Phase of Neural Architecture Search," *arXiv preprint arXiv:1902.08142*, 2019.

[40] S. Baluja, "Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning," Pittsburgh, PA, USA, 1994.

[41] H. Mühlenbein, "The Equation for Response to Selection and Its Use for Prediction," *Evolutionary Computation*, vol. 5, no. 3, pp. 303–346, 1997.

[42] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," in *Neurocomputing: Foundations of Research*, J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, pp. 696–699.

[43] T. Back, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: comments on the history and current state," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 3–17, 1997.

[44] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013.

[45] F. Gomez and R. Miikkulainen, "Incremental Evolution of Complex General Behavior," *Adaptive Behavior*, vol. 5, no. 3-4, pp. 317–342, 1997.

[46] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 694–713, 5 1997.

[47] F. J. Gomez and R. Miikkulainen, "Solving non-Markovian Control Tasks with Neuroevolution," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, 1999, pp. 1356–1361.

[48] K. O. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 131–162, 2007.

[49] S. Risi and K. O. Stanley, "An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons," *Artificial Life*, vol. 18, no. 4, pp. 331–363, 2012.

[50] D. B. DAmbrosio, J. Gauci, and K. O. Stanley, "HyperNEAT: The First Five Years," in *Growing Adaptive Machines*. Springer, Berlin, Heidelberg, 2014, pp. 159–185.

[51] R. Miikkulainen *et al.*, "Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Academic Press, 2019, pp. 293 – 312.

[52] S. Kiranyaz, T. Ince, A. Iosifidis, and M. Gabbouj, "Progressive Operational Perceptrons," *Neurocomputing*, vol. 224, pp. 142–154, 2017.

[53] ——, "Operational Neural Networks," *Neural Computing and Applications (in print)*, 2020.

[54] D. T. Tran, S. Kiranyaz, M. Gabbouj, and A. Iosifidis, "Heterogeneous Multilayer Generalized Operational Perceptron," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–15, 2019.

[55] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-Shot Model Architecture Search through HyperNetworks," in *Workshop on Meta-Learning (MetaLearn 2017) at NIPS*, 2017.

[56] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and Efficient Architecture Search for Convolutional Neural Networks," in *6th International Conference on Learning Representations (ICLR)*, 2018.

[57] A.-C. Cheng, C. H. Lin, D.-C. Juan, W. Wei, and M. Sun, "InstaNAS: Instance-aware Neural Architecture Search," *arXiv preprint arXiv:1811.10201*, 2018.

[58] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural Architecture Search with Bayesian Optimisation and Optimal Transport," in *Advances in Neural Information Processing Systems 31*, 2018, pp. 2016–2025.

[59] F. P. Casale, J. Gordon, and N. Fusi, "Probabilistic Neural Architecture Search," *arXiv preprint arXiv:1902.05116*, 2019.

[60] M. Pelikan, D. E. Goldberg, and F. G. Lobo, "A Survey of Optimization by Building and Using Probabilistic Models," *Computational Optimization and Applications*, vol. 21, no. 1, pp. 5–20, 2002.

[61] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[62] J. Gorodkin, "Comparing two K-category assignments by a K-category correlation coefficient," *Computational Biology and Chemistry*, vol. 28, no. 5-6, pp. 367–374, 12 2004.

[63] Q. Zhang, "On Stability of Fixed Points of Limit Models of Univariate Marginal Distribution Algorithm and Factorized Distribution Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 80–93, 2004.

[64] T. Friedrich, T. Kötzing, and M. S. Krejca, "EDAs Cannot Be Balanced and Stable," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2016, pp. 1139–1146.

[65] F. Glover and M. Laguna, "Tabu Search," in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds. Boston, MA: Springer US, 1998, pp. 2093–2229.

[66] J. J. Hull, "A Database for Handwritten Text Recognition Research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1994.

[67] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[69] O. K. Oyedotun, A. E. R. Shabayek, D. Aouada, and B. Ottersten, "Improving the Capacity of Very Deep Networks with Maxout Units," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018.

[70] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-Deep Neural Networks without Residuals," in *International Conference on Learning Representations (ICLR)*, 2017.

[71] X. Gastaldi, "Shake-Shake regularization of 3-branch residual networks," in *International Conference on Learning Representations (ICLR) Workshop*, 2017.

[72] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," in *British Machine Vision Conference (BMVC)*, 2016.

[73] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures Using Reinforcement Learning," in *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017, pp. 1–18.

[74] Z. Lu *et al.*, "NSGA-NET: A Multi-Objective Genetic Algorithm for Neural Architecture Search," in *The Genetic and Evolutionary Computation Conference (GECCO)*, 2019.

[75] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[76] Y. Sun, G. G. Yen, and Z. Yi, "Improved Regularity Model-Based EDA for Many-Objective Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 662–678, 2018.

[77] B. Doerr and M. S. Krejca, "Significance-based Estimation-of-Distribution Algorithms," *IEEE Transactions on Evolutionary Computation*, 2019.

Anton Muravev received his B.Sc. and M.Sc. degrees in computer science from the Tomsk Polytechnic University, Tomsk, Russia, followed by the M.Sc. degree in information technology from the Tampere University of Technology in 2013, 2015 and 2016 respectively. He is currently working towards the Ph.D. degree at the Tampere University, Tampere, Finland. His research interests include deep learning, pattern recognition and evolutionary computation.

Jenni Raitoharju received her Ph.D. degree at Tampere University of Technology, Finland in 2017. Since then, she has worked as a Postdoctoral Research Fellow at the Faculty of Information Technology and Communication Sciences, Tampere University, Finland. In 2019, she started working as a Senior Research Scientist at the Finnish Environment Institute, Jyvskyl, Finland after receiving Academy of Finland Postdoctoral Researcher funding for 2019-2022. She has co-authored 12 journal papers and 24 papers in international conferences. She is the chair of Young Academy Finland 2019-2020. Her research interests include machine learning and pattern recognition methods along with applications in biomonitoring and autonomous systems.

Moncef Gabbouj received his MS and PhD degrees in electrical engineering from Purdue University, in 1986 and 1989, respectively. Dr. Gabbouj is a Professor of Signal Processing at the Department of Computing Sciences, Tampere University, Tampere, Finland. He was Academy of Finland Professor during 2011-2015. His research interests include Big Data analytics, multimedia content-based analysis, indexing and retrieval, artificial intelligence, machine learning, pattern recognition, nonlinear signal and image processing and analysis, voice conversion, and video processing and coding. Dr. Gabbouj is a Fellow of the IEEE and member of the Academia Europaea and the Finnish Academy of Science and Letters. He served as associate editor and guest editor of many IEEE, and international journals.