Neural Architecture Search for printed Neural Networks

Jakub Marceli Trzciński
Supervisor: Haibin Zhao

Karlsruher Institut für Technologie, 76131 Karlsruhe, Germany
jakub.trzcinski@student.kit.edu

Abstract. Powerful Neural Networks are ubiquitous at present, yet the manual development of each individually is tedious, time consuming, and requires high levels of human expertise. From the desire, to automate this search process and to make it more broadly applicable, emerged the field of Neural Architecture Search. Neural Networks, automatically generated with Neural Architecture Search, established themselves as capable of outperforming human designs. Moreover they constantly push the state-of-the-art higher. Also, simultaneously to the advent of smart wearables and Internet of Things devices, progress has been made in Printed Electronics. These advancements promise to meet the needs of the mentioned, growing, market. Neural Architecture Search, with all of its powerful tools, commits to bringing those two concepts i.e., Neural Networks and Printed Electronics, together. This work will give an overview over the Neural Architecture Search techniques and discuss their suitability for designing solutions for printed Neural Networks.

Keywords: Neural Architecture Search · Search Strategy · printed Neural Networks.

1 Introduction

The concept of Neural Networks (NNs) isn't new, but in the recent years it has become obvious, that NNs are a suitable tool for many optimization problems. Their strength arises, in part, due to their sheer complexity, which, in many cases, is incomprehensible for humans and is treated as a black-box. Given a set architecture, or a hyperparameter pool, we know efficient methods e.g., gradient-based or Bayesian optimization [17], to train and adjust the NN for a specific application. This was not the case with NN architecture engineering. Neural Architecture Search (NAS) is a collection of different techniques with the main goal of automating architecture engineering [3]. As there are many domains and applications, in which Neural Networks are utilized, there is a proliferation of the strategies of Neural Architecture Search i.e., of ways of "learning how to learn".

Motivation Motivation for this work is twofold. On the one hand, although the NAS field is proliferating [11], there are still many open problems and avenues for improvement [10]. With that, the first goal is to gain better intuition for underlying principles of NAS, and understand the intricacies of different design choices on the way. On the other hand, due to the approaching end of Moore's Law, Neuromorphic Computing Systems (NCS) are picking up attention [16]. Specifically for many modern applications i.a., smart wearables, and Internet of Things devices, printed NN (pNN) seem to be a promising candidate [21]. Due to large feature sizes of digital logic circuits in Printed Electronics (PE), conventional digital computing is not feasible in PE yet. Thus, leveraging biology-inspired Neuromorphic Computing as a computing paradigm for PE is a thrilling prospect [21]. All three, NAS, NCS, and PE, meet for the objective of finding novel ways of computation.

History of NAS First signs of NAS could already be seen in the seminal work of Stanley et al. [19]. There, a NN was trained with Evolutionary Algorithms (EAs), and its architecture was simultaneously adjusted. EAs, together with one of the other precursors of NAS i.e. Bayesian-optimization methods, dominated the field in the early years [8]. The next breakthrough came with the NAS approach of Zoph et al. [26], which achieved state-of-the-art results and set off a wave of interest for NAS [3]. Even though this breakthrough required (at that time) around 3000 GPU days of computation to converge on the solution [3], it was a proof of concept which inspired further intensive research of NAS [6].

Problem description NAS can be regarded as part of Auto Machine Learning and has a lot in common with hyperparameter optimization, and meta-learning [3]. According to the common NAS categorization [3, 6, 23], its three core aspects are: search space, search strategy, and evaluation strategy. Figure 1 depicts the interplay of these NAS dimensions.

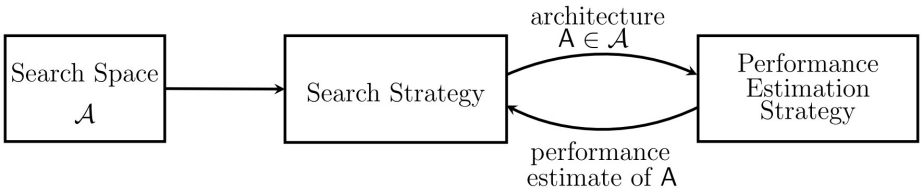

Fig. 1. In this Abstract overview, the core NAS components can be seen [3]

Search Space It refers to the space of all architectures that can be represented within the design [3]. It has fundamental influence on the results of the NAS [24].

Search Strategy The strategy is the profile of the approaches on how to tackle the exploration-exploitation trade-off intrinsic to the vast search spaces of NAS [3].

Model evaluation strategy For each NN architecture found, we have to be able to assess its performance. The simplest approach is time-consuming [6]. Therefore, much effort is being invested in the development of the means to reduce the cost for model evaluation [3].

pNN specific constraints Depending on the neuromorphic architecture, printing NCS components is bound by the technology. It includes a limited set of digital and analog features i.e., NCS building blocks in PE [21]. Moreover, there are several constraints put on the pNNs weights, their interactions, and overall pNN behaviour to consider during pNN architecture design and training. In detail, these are, according to Weller et al. [21] i.a.

- As resistances are in a certain way the NN weight values, there are ranges of resistances feasible in hardware.
- At the circuit level, weights can become coupled.
- Across NN Layers signal degrades.
- Guaranteeing the separability of the NN output signals involves minimum sensing resolution to consider during training.

Finally, because discussed here PE components process analogue signals, production dependent imperfections of the hardware itself must also be considered.

NAS for pNNs Although there are many publications in both NAS and NCS fields, there is little attention given to solutions to NAS for hardware (like printed NCS). This work will focus on the details of the most popular NAS techniques and, after summarizing them, discuss their suitability for pNNs considering the NCS-typical traits, and constraints.

2 Search Space

The search space defines which architectures can [and cannot] be represented in general [3]. As it grows exponentially [6] with the number of choices on the way, it directly determines how simple or complex the search process will be. In general, a neural architecture can be represented as a directed acyclic graph comprising of nodes indicating feature tensors and edges representing operations [6]. In convolutional NNs for instance, it may be considered as useful to restrict the building blocks to those found in other-well performing NNs of the same type. Further improvements can be achieved by varying the "granularity" of the search by using certain advanced human-designed modules as primitive operations [6]. The critical part of the search space design is the balance between novelty, search complexity, and introduced biases. The following sections will describe the classification of different search spaces.

2.1 Global Search Space

Global search space is one of the most intuitive and straight forward approaches [6]. It is parametrized by the maximum number of nodes, operation types, and the corresponding hyperparameters. These are e.g., kernel sizes, number of filters, and stride for a convolutional layer [3]. The nodes-stacked together, each performing specific operations, make up the NN architecture. The resulting models can range from simple "chain-structured" NNs to more complex multi-branch NNs e.g., when allowing skip connections. Because of its conceptual simplicity, the Global search space is comparatively easy to implement. This approach suffers, however, from being computationally expensive and lacking generalization ability. The reason for that is the known tendency of more complex models, i.e., deeper or wider, to outperform simpler models [3, 6]. Generalization means for a model to have the capacity to perform well on a big dataset, while being trained on a smaller one. The search spaces described in next subsection try to address this issue.

2.2 Cell-based Search Space

Studying the way human brains are created and their structure is encoded in the DNA reveals regularity in the neuron arrangement and their connections. Consequently the repeated structural motifs of the human brain are a part of its impressive abilities [18]. To no surprise, many human-designed models aim to leverage the repeating motifs, often to great success [5, 7]. It translates to the distinction between a micro architecture, i.e., the structure of the cells, and a macro architecture, which determines how the cells should be connected [3]. The focus here is to find quality cell candidates, depicted in the "level-two" in the figure 2, and preferably optimize the macro-structure on the way [3]. There is also the possibility of a hybrid approach, where, after finding such promising cells, one can manually engineer the macro architecture [3]. Apart from the simplification of the search space, there is also an advantage in being able to utilize transferability, i.e., training with a lower amount of cells and increasing that amount afterwards [6].

2.3 Hierarchical Search Space

The Hierarchical search space, who already encompasses the cell concept in itself, tries to simplify the joint micro-and-macro architecture optimization [3]. On the example of a three-level architecture, as depicted in the figure 2, the goal is to search for progressively more complex structures, treating each candidate "cell" from the level below as a primitive building block [6]. In comparison with the cell-based search space with a "hard-coded" macro structure, the hierarchical approach promises to allow more complex and flexible NN topologies [6].

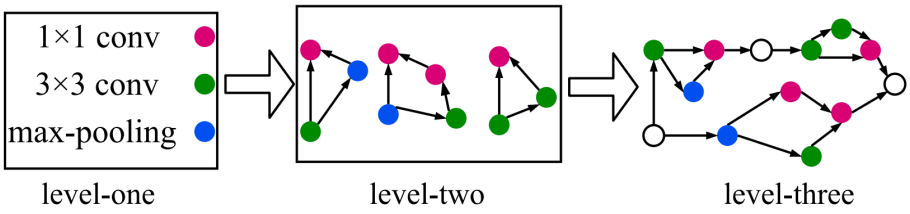

Fig. 2. Example of different levels of hierarchical NN architecture representations [6].

3 Search Strategy

There are two main difficulties in NAS, (1) the search space grows exponentially and (2) the evaluation is costly. Thus, how to explore this space becomes a legitimate question. It represents the classical exploration-exploitation trade-off, since finding a capable architecture quickly is preferable, while premature convergence on a locally-optimal solution is undesirable [3]. In the following sections, the principles of the most popular techniques will be discussed.

3.1 Evolutionary Algorithm (EA)

Researchers, inspired by biology, created the branch of EAs, i.e., population-based metaheuristic optimization algorithms, which simulate the evolution of a species in nature. In terms of the its application in NAS, EAs' root reach 3 decades back [14]. Researchers noticed, that EAs are robust and able to handle a wide range of complex problems effectively [6]. In its early days, as in [19], the NN architectures and the NN weights were optimized jointly. With the transition to deep, and more complex, NNs, the Gradient-based weights optimization proved better at scale. Therefore, more recent EAs for NAS use Gradient-based weights optimization, and EAs exclusively for NN architecture optimization [3, 13]. Also depicted in the figure 3, the optimization process consists of the following steps:

- Defining the initial population
- Selection
- Evolutionary operations
- Fitness evaluation and assessment of the stopping criterion

The subsequent paragraphs describe the crucial aspects of EAs for NAS.

Initial population Depending on what one tries to accomplish, the initial population can play a decisive role in the result of the NAS. If the goal is to minimize the human intervention aspect, random initialization can be chosen. On the other hand, so called "rich initialization" aims to improve upon other state-of-the-art architectures by e.g., compressing them. The drawback here is that the population can have a comparatively small progress potential and the

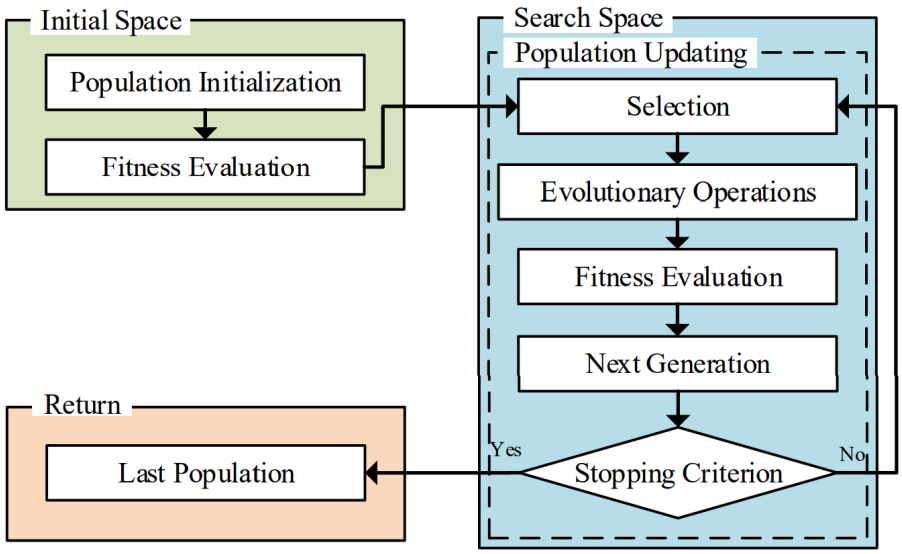

Fig. 3. The EA-based NAS principle consists of (1) Population Initialization, (2) Population Updating, (3) Last Population [13]

initialization can prohibit the development of novel ideas. To capitalize on the EAs capacity to discover innovative architectures, the primordial population can originate in a trivial space, i.e., a space of basic architectures containing only few layers [13].

Selection Analogous to natural evolution between generations, selection i.e., to decide the specimens that should generate offspring and make up the next generation, takes place. This step has the most surprising effect on the quality of the optimization outcome. While there are several existing strategies of selection [13] i.a., selecting the best, discarding the worst, discarding the oldest, or tournament selection, I find the following distinction to prevail. Focusing on the individuals with the highest fitness tends to cause a loss of diversity and it may consequently result in the optimization to converge on local optima [13]. Contrastingly, rewarding not only fitness, but also diversity in respect to the rest of the population, can be just enough to avoid local optima and explore the search space more efficiently. The latter approach is, however, more computationally intensive, which can offset the gains, which were made [18].

Evolutionary operations Commonly, these are the crossover and mutation operators [13]. Made to mimic the genetic recombination happening in the course of biological reproduction [6], they alter the individuals, i.e., the NN architectures, in the population. Although they constitute part of the EAs appeal, they also present a hurdle, as every operation must be constructed with great care. Apart from being compatible with the bespoke encoding, both

crossover and mutation must not damage the data and assert that the offspring / mutated individual is competitive with regard to its parents [13].

EA typical traits The evolutionary concept departs in numerous ways from the others. These are i.a.: (1) Instead of converging on a single solution, it maintains a population of solutions [13] i.e., solutions, that can be good in different ways [18]. (2) EAs are bounded by the restrictions of the defined evolutionary operations, moreover, these operations are performed randomly [6]. (3) Typically, simulating evolution in the NAS setting is intensive computationally and presents a bottleneck [13]. (4) In some sense, EAs possess a "open-endedness" property [18].

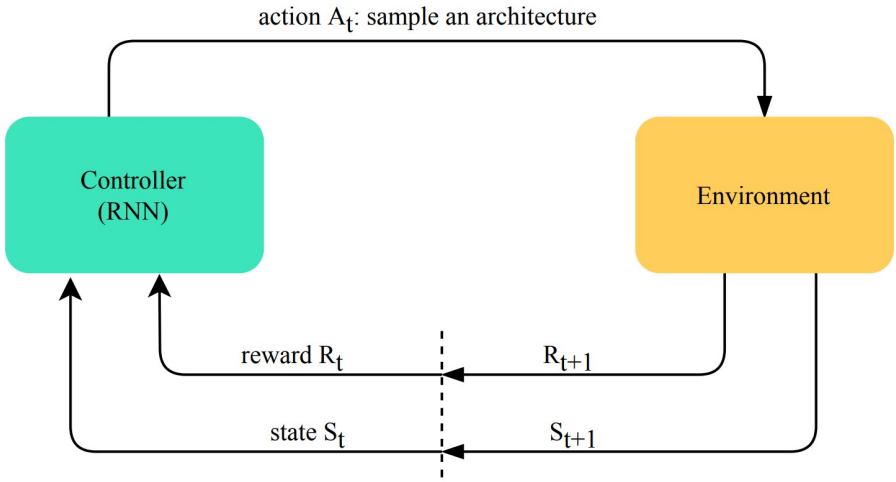

Fig. 4. Depiction of the RL for NAS principle [6]

3.2 Reinforcement Learning (RL)

There is an observable link between many of the foundational concepts of RL and biological learning systems. Thus, of all the methods used in Machine Learning, RL is the most akin to the manner, in which humans and other animals learn [20]. Currently, RL is an established paradigm for modelling the process of sequential decision making, whenever the problem can be translated to any setting of an agent interacting with an environment and a goal of maximizing its future reward [23]. In the case of NAS, as seen in the figure 4, the agent is the controller, usually a Recurrent Neural Network (RNN), which influences the environment, our search space, by taking actions. Its decision mechanism is based on sampling an architecture, evaluating its guess by some form of architecture performance estimation, and getting feedback, i.e., its reward. Environment changes action

is taken according to the agents policy, which is a mapping from the space of environment states to a probability distribution over possible actions [23]. As mentioned in the section 1, first RL for NAS works was resource-intensive. Since then, many improvements have been proposed [6], which indicate that RL isn't inherently slow and cannot be ruled out as a viable NAS technique. Nevertheless, there are some RL limitations, that need to be mentioned. Firstly, RL works on discrete [20]. Secondly, although RL can identify complex patterns, its search efficiency and stability is not guaranteed [6].

3.3 Bayesian Optimization (BO)

Bayesian Optimization is a known method from the optimization toolbox specialized on handling black-box functions, whose evaluations are expensive to obtain [9]. The core concept of this method is obtaining a probabilistic surrogate model of the objective function, and an acquisition function, which predicts the next data point to evaluate. The acquisition function accounts for both the predicted response, and the uncertainty of the prediction [23]. It has already been widely adopted in ML as a method of hyperparameter optimization. Because classically it is used on continuous and relatively low-dimensional problems, there was a certain reluctance to frame NAS as a BO problem prevalent [3]. Nevertheless, coincident to the other discussed NAS strategies, BO has its own unique advantages and shortcomings. A remark has to be made, that part of the BO strategy comes down to selecting the probability distribution prior, which here was chosen to be the Gaussian Process (GP). It is a popular pick, that has proven itself to be a fitting surrogate model for computer experiments [17].

Advantages Used as a mean of guiding the changes of the NN architecture, it can reduce the number of evaluations needed in the search [9]. Thus, it can accelerate the NAS. Also, given limited time, it has been shown, that BO is viable for finding competitive architectures quickly [9].

Issues The inherent limitation of GP-based BO methods is the inference time scaling cubically in the number of observations [6]. Moreover, designing a BO method for NAS is hard, as BO isn't built for handling variable-dimensional input (differently sized NN architectures) [6]. Therefore, one of the obstacles is finding a way to bypass this issue. Possible solutions are a fixed-length encoding scheme or e.g., reframing the problem by developing an edit-distance Kernel for the case of network morphisms as in [9]. In either way, considerable effort has to be put into creating a valid and generic BO-compatible representation of the NAS problem.

Similar methods A broader range of similar methods exist, namely the family of Surrogate Model-based Optimization (SMBO). Although not discussed here in detail they share some of the similar challenges (e.g., design of the surrogate model, which introduces complexity and potential bias) and benefit promises (e.g., reduced number of evaluations needed). For more in-depth discussion I refer to the work of He et al. [6].

3.4 Gradient-based Optimization(GO)

The Innovation of the GO paradigm lies in the departure from seeing the search space as necessarily discrete. The trick is based on the continuous relaxation of NN architecture representations enabling their optimization by gradient descent. Since the data efficiency of GO surpasses that of black-box search [12], it proved itself to be a path worth exploring. To my knowledge, and as the GO methods are summarized by He et al. [6]-and Elsken et al. [3], the core of all the attempts lies in the concept of a supernet, i.e., an architecture that embodies the whole of the search space, and from which child architectures can be derived. Figure 5 illustrates an instance of GO for NAS. To enable GO, among the variants of GO, different types of a differentiable softmax function are used [6].

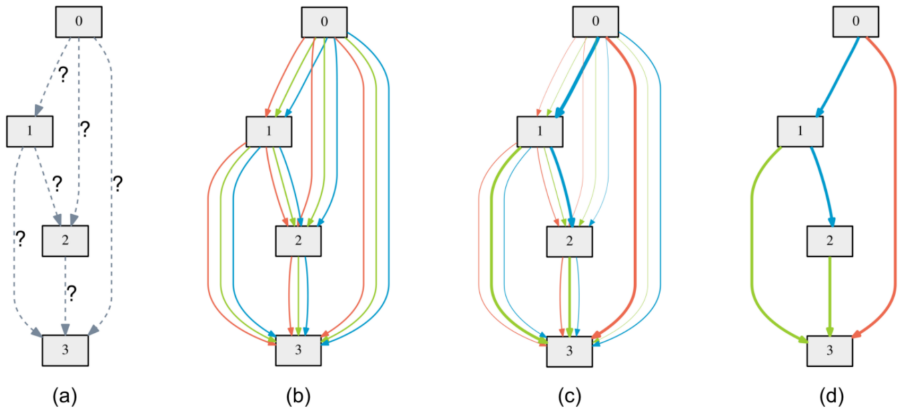

Fig. 5. Differentiable ARchitecture Search (DARTS) [12] is a representative GO example. Overview consists of: (a) Outline of the supernet, i.e., exact operations are unknown. (b) Continuous search space is obtained by placing all candidate operations for each edge in a weighted sum. (c) After the optimization mix of candidate probabilities is acquired. (d) Induced final architecture.

Issues In the other NAS strategies, the architecture part is discrete, whereas, for the validation set accuracy, NN weights are optimized in a continuous manner. In the GO case, both components are gradient-descent optimized. This indicates joint learning of architectures and their weights, and implies a bilevel optimization problem [12]. This class of problems combined with the high-dimensionality nature of NAS is a laborious one to be solved directly [6]. To overcome this difficulty, approximations may be introduced. However they pose a danger, as e.g., moving to a efficiently solvable

sinlge-level optimization introduces overfitting issues, rendering NN performance estimations undependable [6]. There are also mixed-level approaches, which promise to overcome the mentioned issue [4], but one has to be mindful, that they are still not direct solutions. Another aspect to consider is managing the memory requirement, as the number of candidate operations throughout the supernet might be to big, for it to fit into GPU memory at once [3]. This problem can be overcome, e.g., by reducing the search space size, or with use of differentiable samplers [6]. Important note: GO methods use some variation of the supernet idea, which has its characteristic drawbacks discussed later in the section 4.3.

3.5 Random Search (RS)

Random Search is already a competitive baseline in the NAS-related hyperparameter optimization problem [10]. Now, it is also widely accepted [3, 6, 10, 15, 23], that RS often just narrowly misses on the state-of-the-art performance. In the example of a search space of DARTS [12], Li and Talwalkar [10] showed, that a properly tuned RS with early stopping can come astoundingly close to the results of more refined and complex search strategies. In the grand scheme of things, connections, such as that RS working on a search space that is known to sample quality NN architectures [23], help uncovering misbeliefs, and foster research. With that, the role of such a baseline for scientific progress is invaluable. Moreover, the mentioned results prove, that there is still a lot of deep understanding in NAS to be gained.

4 Model Evaluation Strategy

With our prime goal of acquiring a performant NN architecture, the search must be guided by a metric, that is supportive of the goal. Depending on the use case scenario, the most straight forward method is to fully train the NN architecture, which is sampled from the search space, on a chosen dataset, then measure this NNs performance on the validation set. However this approach is prohibitively time-consuming [6] and as shown in the figure 1, models performance is often estimated. Another reason justifying approximations is the sole need of a ranking of the candidate architectures. For that purpose, obtaining relative fitness of the architectures suffices [3]. The main focus of this section will be describing the most popular means of accelerating the model evaluation.

4.1 Performance Estimation

With now years of experience in NN training, we know, that Human experts are capable of recognizing ill-performing models on the basis of their partial learning curves [1]. Therefore, much effort has been put into translating this human intuition into an algorithmic strategy.

Low-fidelity Estimation One of the avenues of achieving speed-up in the NN performance estimation is reducing the time budget for the NN training. According to Elsken et al. [3], the most popular methods are:

- Training for fewer epochs i.e. early stopping
- Training on a subset of the dataset
- Training a downscaled model of the NN
- Training on downscaled data

It is worth noting that these techniques usually underestimate the NN performance and work only while the discrepancy between the approximation and the "standard" NN training is relatively small [3].

4.2 Weight Inheritance

Another possible step towards reducing the computational cost of model evaluation is the concept of initializing the weights of new NN architectures with weights of other, already trained, NN architectures [3]. That way there is no need for discarding the NN once it has been evaluated [6]. In general, the Weight Inheritance approach necessitates a NN structure, which enables some kind of expression of similarity, so that the most related architectures share weights. It can be accomplished by utilizing i.a., network morphisms, which allow NNs to grow and change gradually. With that, the NNs can retain high performance without the need of training them from scratch [3]. Alternatively, as with evolutionary models, one can speak of Weight Inheritance between parent and offspring models [13].

4.3 One-shot NAS

One-shot NAS is an alternative perspective on weight sharing across models to amortize the cost of training [2]. Instead of sharing the weights between similar, but separate NN architectures [3], one can "condense" the whole search space into one supernet [2]. This means that, there exists only one single set of weights, from which, by zeroing out some of the operations, a NN architecture can be obtained [2]. This principle is visible in the figure 5, where (b) showcases the supernet and (d) displays one of the possible subnets i.e., a possible NN architecture, derived from the supernet. With that, the weights are shared implicitly between NNs that have edges of the supernet in common [2].

Advantages As underlined by Bender et al. [2], the NAS efficiency improvement comes not only from having to train only one, albeit large, supernet, but also from the fact, that while with the exponential growth of the search space, One-shot models size grows only linearly. This can reduce the resources required for NAS by orders of magnitude [2].

Issues Somewhat intuitively, due to deep coupling of the supernet weights, competition between candidate operations in the supernet can be observed [6]. In the section 4.1, it was pointed out that aggressive approximations can misjudge the actual performance [3]. Therefore, it works on the assumption, that the incurred bias, and the underestimation of the actual performance, don't hamper the NN architecture ranking too severely [3]. Also, as seen in the left part of the figure 6, by the nature of the a priori defined supernet, One-shot models restrict the search space to its subnets [3]. In contrast, the right part of the figure 6 represents more flexible classical search space [6].

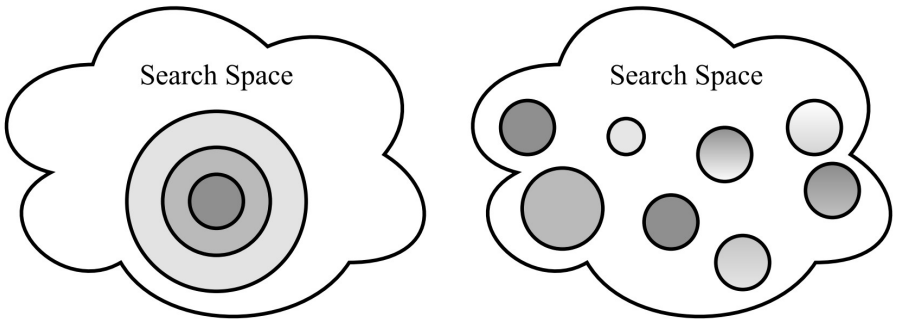

Fig. 6. Figurative distinction between: One-shot model search space (left) vs. non-one-shot search space (right) [6].

5 Discussion

While discussing different NAS strategies on the basis of particular implementations, separating different layers of design choices proves to be troublesome at best, and often impossible, without performing experiments myself. Also, some strategies e.g., DARTS [12], include "entangled" design choices i.e., in the mentioned example it is the GO method along with the One-shot Search Space, which render theoretical comparisons between strategies futile. As an example, one could run EA-based NAS on DARTS [12] search space, but not the other way around. The widespread i.a., [10, 11, 3, 22], notion in the field, that "NAS Evaluation is Frustratingly Hard" [24], confirms the observations made above. In this section, some of the issues NAS research is facing will be mentioned and we'll come back to the initial question of NAS for pNNs.

5.1 NAS Issues

Comparing NAS strategies Firstly, NAS research needs more stable comparison baselines [10], and although there is some pioneer work on NAS benchmarks with

the NAS-Bench-101 [25] and its successors, there still aren't any "gold-standard" evaluation protocols [11]. Furthermore, many existing benchmarks use relatively small datasets, which current NN architectures have nearly mastered, and which is increasing the risk of overfitting NAS to those datasets [11]. This might indicate, that the benchmarks are simply not hard enough.

Design choices With so many design choices in the process, there is a clear need for ablation studies to be conducted [10]. On the example of DARTS [12] and its derivative works using the same search space, it turns out that differently sampled architectures perform extremely alike [24]. This shows once again that the choice of a search space can predetermine the outcome of the search. The status quo is that while many modifications to existing techniques are being made, there is often no deep understanding why the results are better than before [11]. On that front, one of the first formal studies on NN encoding in the NAS context [22] was published only last year.

Other issues For further, unmentioned issues I refer to the analysis of He et al. [6], Lindauer et al. [11], and Li et al. [10].

5.2 NAS for pNNs

Disclaimer: this, and the following, sections represent solely opinions and hypothetical reasoning, that was induced by impressions from the referenced work. The work of Weller et al. [21] will serve as an example of a pNN design process for the discussion. The figure 7 exemplifies a pNN design flow. The steps of Mapping components and Fabrication are fixed, and the problem has following critical design constraints [21]:

- The set of digital and analog components in PE is limited
- Analog designs are preferable
- Weights need to conform to the ranges of resistances feasible in hardware
- Weight coupling can occur in a pNN, with decoupled weights, the pNN is susceptible to signal noise
- Signal degradation across layers occurs in a pNN
- Analog circuits are influenced by hardware imperfections
- NCS components used are in the mm range

In this case (figure 7), the role for NAS falls between the steps of Specification and Training (Design). Therefore, the NAS method in question has to have following characteristics: (1) good handling of multiple NN constraints, (2) insensitivity to hardware imperfections, (3) use an optimal number of hardware components to reduce the pNN footprint. With these in mind, we can come to conclusion, that:

- RL is well suited with respect to the (1), and (3) can be formulated as a part of the agents reward function.

– EA is well suited with respect to the (1) and (3) is achieved thanks to gradual growth during mutations
– BO is intrinsically able to cope with the (2). It might be challenging to account for all the constraints of (1).
– GO has a search space, that relies heavily on performance estimations. If these predictions would line up with the actual pNN performance, then GO could be the best. If not, GO wouldn't be suitable for this task.

Consequently, EA and RL would seem to be reasonable choices to try out first.

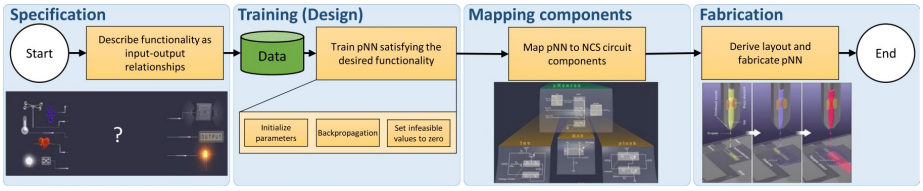

Fig. 7. Example of design flow in the case of pNNs [21]

Looking forward There is also a prospect of extending the NAS to the third phase depicted in the figure 7. With that, custom arrangement of the basic NCS building blocks could be developed. Such alternatives to standard neurons would make up whole new kinds of pNN architectures. Finally, although back-propagation is a well established algorithm leading to fast convergence in learning and has broad applicability, it is restrictive and notably, it was developed with the von Neumann architecture in mind [16]. Depending on the application it may require a non-von Neumann architecture [16].

References

1. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating Neural Architecture Search using Performance Prediction. arXiv:1705.10823 [cs] (Nov 2017), http://arxiv.org/abs/1705.10823, arXiv: 1705.10823
2. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and Simplifying One-Shot Architecture Search. pp. 550–559. PMLR (Jul 2018), https://proceedings.mlr.press/v80/bender18a.html
3. Elsken, T., Metzen, J.H., Hutter, F.: Neural Architecture Search: A Survey. arXiv:1808.05377 [cs, stat] (Apr 2019), http://arxiv.org/abs/1808.05377, arXiv: 1808.05377
4. He, C., Ye, H., Shen, L., Zhang, T.: Milenas: Efficient neural architecture search via mixed-level reformulation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11993–12002 (2020)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs] (Dec 2015), http://arxiv.org/abs/1512.03385, arXiv: 1512.03385

6. He, X., Zhao, K., Chu, X.: AutoML: A Survey of the State-of-the-Art. Knowl. Based Syst. (2021). https://doi.org/10.1016/j.knosys.2020.106622
7. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely Connected Convolutional Networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2261–2269 (Jul 2017). https://doi.org/10.1109/CVPR.2017.243, iSSN: 1063-6919
8. Jaafra, Y., Luc Laurent, J., Deruyver, A., Saber Naceur, M.: Reinforcement learning for neural architecture search: A review. Image and Vision Computing **89**, 57–66 (Sep 2019). https://doi.org/10.1016/j.imavis.2019.06.005, https://www.sciencedirect.com/science/article/pii/S0262885619300885
9. Jin, H., Song, Q., Hu, X.: Auto-keras: An efficient neural architecture search system. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 1946–1956 (2019)
10. Li, L., Talwalkar, A.: Random Search and Reproducibility for Neural Architecture Search. pp. 367–377. PMLR (Aug 2020), https://proceedings.mlr.press/v115/li20c.html
11. Lindauer, M., Hutter, F.: Best practices for scientific research on neural architecture search. Journal of Machine Learning Research **21**(243), 1–18 (2020)
12. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable Architecture Search. ICLR (2019)
13. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. IEEE Transactions on Neural Networks and Learning Systems (2021)
14. Miller, G., Todd, P., Hegde, S.: Designing Neural Networks using Genetic Algorithms. (Jan 1989)
15. Ren, P., Xiao, Y., Chang, X., Huang, P.Y., Li, Z., Chen, X., Wang, X.: A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. arXiv:2006.02903 [cs, stat] (Mar 2021), http://arxiv.org/abs/2006.02903, arXiv: 2006.02903
16. Schuman, C.D., Potok, T.E., Patton, R.M., Birdwell, J.D., Dean, M.E., Rose, G.S., Plank, J.S.: A Survey of Neuromorphic Computing and Neural Networks in Hardware. arXiv:1705.06963 [cs] (May 2017), http://arxiv.org/abs/1705.06963, arXiv: 1705.06963
17. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian Optimization of Machine Learning Algorithms. In: Advances in Neural Information Processing Systems. vol. 25. Curran Associates, Inc. (2012)
18. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. Nature Machine Intelligence **1**(1), 24–35 (Jan 2019). https://doi.org/10.1038/s42256-018-0006-z, https://www.nature.com/articles/s42256-018-0006-z
19. Stanley, K.O., Miikkulainen, R.: Evolving Neural Networks through Augmenting Topologies. Evolutionary Computation **10**, 2002 (2001)
20. Sutton, R.S., Barto, A.G.: Reinforcement Learning, second edition: An Introduction. MIT Press (Nov 2018), google-Books-ID: uWV0DwAAQBAJ
21. Weller, D.D., Hefenbrock, M., Beigl, M., Aghassi-Hagmann, J., Tahoori, M.B.: Realization and training of an inverter-based printed neuromorphic computing system. Scientific Reports **11**(1), 9554 (May 2021). https://doi.org/10.1038/s41598-021-88396-0, https://www.nature.com/articles/s41598-021-88396-0

22. White, C., Neiswanger, W., Nolen, S., Savani, Y.: A Study on Encodings for Neural Architecture Search. arXiv:2007.04965 [cs, stat] (Feb 2021), http://arxiv.org/abs/2007.04965, arXiv: 2007.04965
23. Wistuba, M., Rawat, A., Pedapati, T.: A Survey on Neural Architecture Search. arXiv:1905.01392 [cs, stat] (Jun 2019), http://arxiv.org/abs/1905.01392, arXiv: 1905.01392
24. Yang, A., Esperança, P.M., Carlucci, F.M.: NAS evaluation is frustratingly hard (Sep 2019), https://openreview.net/forum?id=HygrdpVKvr
25. Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., Hutter, F.: NAS-Bench-101: Towards Reproducible Neural Architecture Search. arXiv:1902.09635 [cs, stat] (May 2019), http://arxiv.org/abs/1902.09635, arXiv: 1902.09635
26. Zoph, B., Le, Q.V.: Neural Architecture Search with Reinforcement Learning. arXiv:1611.01578 [cs] (Feb 2017), http://arxiv.org/abs/1611.01578, arXiv: 1611.01578