SciTeller: An LLM-Based Framework for Persona-Adaptive Scientific Storytelling

frontiers | Frontiers in Artificial Intelligence

TYPE Original Research
PUBLISHED 19 June 2024
DOI 10.3389/frai.2024.1414707

OPEN ACCESS

EDITED BY
Valentino Santucci,
University for Foreigners Perugia, Italy

REVIEWED BY
Wei Wang,
Xi'an Jiaotong University, China
Wenyu Yang,
Huazhong Agricultural University, China

*CORRESPONDENCE
Alice Bizzarri
✉ alice.bizzarri@unife.it

RECEIVED 09 April 2024
ACCEPTED 28 May 2024
PUBLISHED 19 June 2024

CITATION
Bizzarri A, Fraccaroli M, Lamma E and Riguzzi F (2024) Integration between constrained optimization and deep networks: a survey. *Front. Artif. Intell.* 7:1414707. doi: 10.3389/frai.2024.1414707

Integration between constrained optimization and deep networks: a survey

Alice Bizzarri[1]*, Michele Fraccaroli[1], Evelina Lamma[1] and Fabrizio Riguzzi[2]

[1]Department of Engineering, University of Ferrara, Ferrara, Italy, [2]Department of Mathematics and Computer Science, University of Ferrara, Ferrara, Italy

Integration between constrained optimization and deep networks has garnered significant interest from both research and industrial laboratories. Optimization techniques can be employed to optimize the choice of network structure based not only on loss and accuracy but also on physical constraints. Additionally, constraints can be imposed during training to enhance the performance of networks in specific contexts. This study surveys the literature on the integration of constrained optimization with deep networks. Specifically, we examine the integration of hyper-parameter tuning with physical constraints, such as the number of FLOPS (FLoating point Operations Per Second), a measure of computational capacity, latency, and other factors. This study also considers the use of context-specific knowledge constraints to improve network performance. We discuss the integration of constraints in neural architecture search (NAS), considering the problem as both a multi-objective optimization (MOO) challenge and through the imposition of penalties in the loss function. Furthermore, we explore various approaches that integrate logic with deep neural networks (DNNs). In particular, we examine logic-neural integration through constrained optimization applied during the training of NNs and the use of semantic loss, which employs the probabilistic output of the networks to enforce constraints on the output.

KEYWORDS

deep learning, symbolic artificial intelligence, constrained training, constrained neural architecture search, neural-symbolic integration

1 Introduction

Artificial intelligence (AI) has long been characterized by two primary paradigms: symbolic and neural approaches. The field of AI has witnessed notable advancements since its inception in the 1950s. Seminal works by McCulloch and Pitts (McCulloch and Pitts, 1943) laid the groundwork for neural networks, while Turing's seminal contributions in the 1950s introduced the concept of machine intelligence (Turing, 2009). Symbolic AI dominated the landscape until the 1980s, after which neural AI began to grow and attract considerable attention. The ongoing debate about these two approaches remains extensive. However, in recent years, the convergence of symbolic and neural AI methodologies has gained traction. In addition to fundamental symbolic or neural techniques, hybrid applications incorporating both symbolic and neural features have emerged.

The main differences between the neural and symbolic fields of AI are as follows:

1. Neural approaches provide associative results, while symbolic approaches produce logical conclusions
2. Neural methods learn and adapt to the data provided, whereas human intervention is common in symbolic methods
3. Neural methods, such as deep learning (DL) that use multiple layers to progressively extract higher-level features from the raw input Deng et al. (2014), can handle large and noisy datasets, while symbolic methods perform better when dealing with relatively small and precise data.

The popularity of machine learning (ML) in recent years has raised questions about which tasks are suitable for deep learning, which ones require model-based symbolic reasoning, and what advantages an integration between the two approaches might bring.

Several algorithms that bridge symbolic and neural methodologies have been developed, including knowledge-based neural networks (KBNN or KBANN) (Agre and Koprinska, 1996), graph neural networks (GNNs) (Lamb et al., 2020), connectionist logic programming and inductive learning (C-IL^{2}P) (Avila Garcez and Zaverucha, 1999), connectionist knowledge time logic (CTLK) (Garcez et al., 2019), the hybrid expert system (HES) (Sahin et al., 2012), and the tensor product representation (Smolensky, 1990), which features a neural network core coupled with a symbolic problem solver.

This study aims to analyze the integration of constraints in a neural context from two perspectives, which, although seemingly different, share common features: constrained neural architecture search (NAS) and the integration of constraints in neural networks. Both paradigms focus on incorporating restrictions and prior knowledge into the process of building and training neural networks to improve efficiency, interpretability, and compliance with specific application requirements. Both require the incorporation of knowledge and the use of constrained optimization, although for different purposes. In constrained NAS, constraints are applied during the architecture search phase to create neural networks optimized for specific contexts, generally within the realm of TinyML, which involves physical constraints such as computational limitations. In the second case, constraints are applied during the training of the network itself to achieve networks that perform better in specific contexts, such as handling imbalanced data, or to enhance network performance by injecting domain knowledge.

In this study, after providing necessary background information, we delve into the state-of-the-art integration of NAS with constraints dictated by the physical limitations inherent in embedded systems, a domain commonly referred to as Tiny Machine Learning (TinyML). NAS methodologies have gained increasing popularity (Benmeziane et al., 2021) and have become indispensable for expediting and automating the arduous and error-prone process of synthesizing novel DL architectures. While NAS has been extensively researched in recent years and has exhibited remarkable success, its practical applicability to real-world challenges still poses significant hurdles. Notably, the complexity of convolutional neural network architectures makes them unsuitable for deployment on resource-constrained platforms typical of TinyML, such as mobile and embedded systems.

Within this survey, we illustrate various solutions from the literature aimed at adapting NAS systems for TinyML. Specifically, we examine several NAS frameworks (Dong et al., 2018; Zhou et al., 2018; Jin et al., 2019; Tan et al., 2019; Fraccaroli et al., 2021, 2022; Liberis et al., 2021) and explore potential methodologies for incorporating physical constraints into synthesized networks.

In the second part of the survey, we scrutinize works where contextually inferred constraints are leveraged to enhance neural network performance. We present examples of constrained neural networks (NNs) employing penalty methods, such as the work by Sangalli et al. (2021), where a deep neural network (DNN) is formulated for binary classification under class imbalance conditions as a constrained optimization problem, alongside novel frameworks for out-of-distribution (OOD) detection (Katz-Samuels et al., 2022). Then, we exemplify probabilistic integration with the study conducted by Xu et al. (2018), where a novel methodology is proposed for integrating symbolic knowledge into deep learning. They derive a semantic loss function that establishes a connection between neural output vectors and logical constraints, taking into account the extent to which the neural network adheres to the constraints imposed on its output.

This study is structured as follows: Section 2 offers an overview of fundamental concepts encompassing deep neural networks (DNN), automated machine learning (AutoML), optimization, and constraints. Section 3 delves into different approaches to NAS, multi-objective optimization (MMO), and their integration. Section 4 explores the integration of logic and deep learning through the Lagrange Multiplier Method and probabilistic interpretation, along with their practical applications. Finally, in Section 5, we draw conclusions based on the findings presented.

2 Main concepts

This section provides an overview of the basic components of the most popular DNNs. Then, it presents the main research concepts of autoML, optimization, and constraints, to provide a general overview of the topics covered in the following sections.

2.1 DNNs

Recently, DNNs are one of the hottest areas of ML. Their application spans across diverse domains, including Computer Vision (CV), Natural Language Processing (NLP), and robotics. DNNs, or neural networks with multiple hidden layers, possess the capability to automatically extract features from extensive unstructured datasets, such as text, images, and audio, or from large tabular data (e.g., clinical data). Through their multi-layered architecture, DNNs can iteratively learn the mapping between input features and predicted classes, achieving high levels of accuracy.

Given their versatility, DNNs find extensive applications in various domains. Notably, in CV, convolutional neural networks (CNNs) emerge as the primary tool, leveraging convolutions to extract crucial feature vectors from input images. Prominent examples include AlexNet (Krizhevsky et al., 2017), VGG (Simonyan and Zisserman, 2014), GoogLeNet (Szegedy et al., 2015), ResNet (He et al., 2016), and DenseNet (Huang et al., 2017).

In the realm of CV, networks are typically classified into three main categories based on the task they perform: image classification (He et al., 2016; Howard et al., 2017; Tan and Le, 2019; Dai et al., 2021), object detection (Redmon et al., 2016; Lin et al., 2017), and semantic segmentation (Badrinarayanan et al., 2017; He et al., 2017).

Another significant application domain is NLP, renowned for its complexity owing to the inherent ambiguity characteristic of human language. In this study, recurrent neural networks (RNNs) such as Long Short-Term Memory (LSTM) (Staudemeyer and Morris, 2019) and Gated Recurrent Unit (GRU) (Cho et al., 2014), alongside NNs with memory, are employed to learn context and word connections. In recent years, the rise of transformers (Tenney et al., 2019; Lin et al., 2021) has revolutionized NLP, with their self-attention mechanism enabling the capture of relationships among words in a sentence. Notably, transformers have also found application in CV, demonstrating state-of-the-art performance (Dai et al., 2021).

2.2 AutoML

There are many different ML algorithms, each characterized by a distinct set of hyperparameters, resulting in an overwhelming array of potential alternatives. Consequently, their application typically entails a complex endeavor necessitating experience, time, and labor. AutoML (He et al., 2021), an emerging scientific discipline, addresses this challenge by exploring methodologies for the efficient, objective, and data-driven construction of ML models.

In recent years, many approaches have been proposed for both the construction and optimization of model learning pipelines and the development of DNNs. AutoML methods can be categorized on the basis of various criteria, including the optimization method employed (e.g., Bayesian optimization, genetic programming, and random search), the structure of the generated pipelines (e.g., with or without fixed structure), and the utilization of meta-learning for leveraging insights from prior datasets or post-processing tasks such as ensemble construction (Gijsbers et al., 2019). Noteworthy examples of AutoML frameworks include Auto-WEKA (Thornton et al., 2013), auto-sklearn (Feurer et al., 2015), and AutoKeras (Jin et al., 2019).

The following section provides a succinct overview of the foundational principles underlying all AutoML systems.

2.2.1 Automated HPO

Automated hyperparameter optimization (HPO) is a crucial task within AutoML systems, particularly for optimizing the hyperparameters (HPs) of ML algorithms, including DNNs, that are particularly sensitive to the choice of hyperparameters, making automated HPO essential for achieving optimal performance. The significance of automated HPO is underscored by its multifaceted utility:

- **Reduction of human effort:** Automated HPO alleviates the burden on human practitioners by automating the tedious and time-consuming process of manually tuning hyperparameters for ML applications.
- **Performance enhancement:** Through automated optimization, ML algorithms can achieve improved performance, leading to new state-of-the-art results across various machine learning tasks (Snoek et al., 2012; Melis et al., 2017).
- **Enhanced reproducibility and fairness:** Automated HPO facilitates fair comparisons between different ML methods by ensuring that they are all evaluated under the same hyperparameter configurations. This enhances the reproducibility and fairness of scientific studies (Bergstra et al., 2013).

However, HPO presents several challenges that make it a difficult problem. The main challenges includes: the cost of evaluating functions for large models, complex machine learning pipelines, or large datasets; the complexity and high density of the configuration space (encompassing a mix of continuous, categorical, and conditional hyperparameters); the lack of access to the gradient of the loss function with respect to HPs; and the inability to directly optimize generalization performance due to the limited size of the training datasets (Feurer and Hutter, 2019).

The Automated HPO problem can be formally defined as follows:

Let $\mathcal{A}$ denote a machine learning algorithm with N HPs. We denote the domain of the n-th HP by Λ_n and the overall HP configuration space as $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_n$. A vector of hyperparameters is denoted by $\lambda \in \mathbf{\Lambda}$, and $\mathcal{A}$ with its hyperparameters instantiated to λ is denoted by $\mathcal{A}_\lambda$. Given a dataset $\mathcal{D}$, our goal is to find:

$$\boldsymbol{\lambda}^* = \arg\min_{\lambda \in \mathbf{\Lambda}} \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_\lambda, D_{train}, D_{valid}), \quad (1)$$

where $\mathbf{V}(\mathcal{L}, \mathcal{A}_\lambda, D_{train}, D_{valid})$ measures the loss of a model generated by algorithm $\mathcal{A}$ with hyperparameters λ on training data D_{train} and evaluated on validation data D_{valid}. In real applications, we have access to a limited number of $D \sim \mathcal{D}$ data; therefore, it is necessary to approximate the expectation in Equation (1).

Popular choices for the validation protocol $\mathbf{V}(\cdot, \cdot, \cdot, \cdot)$ are holdout error and cross-validation for a user-given loss function (such as the misclassification rate). Several strategies have been proposed to reduce evaluation time: it is possible to test ML algorithms only on a subset of the folds (Thornton et al., 2013), a subset of data (Swersky et al., 2013; Klein et al., 2017), or for a limited number of iterations.

2.2.2 Meta-learning

Meta-learning, or learning to learn, is the science of systematically studying how different ML systems perform on a wide range of tasks to learn from this experience (*meta-data*) and perform a new task as quickly as possible. This not only allows the user to improve the performance of the ML design but also replaces hand-tuned algorithms with new algorithms learned in a data-driven way (Vanschoren, 2019). First, it is necessary to record the exact algorithm configurations used for training the models (e.g. HPs, pipelines, network architectures, and training time), that is, the *meta-data* describing previous training activities. Then, we have to learn from these *meta-data* to extract and transfer knowledge to

new tasks. The term *meta-learning* refers to any type of learning based on previous experiences with other tasks. The more similar the previous tasks are, the more types of *meta-data* we can exploit. When a new task represents completely unrelated phenomena, exploiting previous experiences will not be effective.

2.2.3 NAS framework

In recent years, NAS frameworks have become fundamental for optimizing neural network HP. NAS, a subfield of AutoML, exhibits significant overlap with HPO and meta-learning methodologies.

NAS aims at discovering the best neural network architecture for specific requirements. It encompasses a suite of tools and methodologies that systematically explore and evaluate many architectures within a predefined search space, employing various search strategies such as random search, grid search, genetic algorithms, or Bayesian search (Elsken et al., 2019; Liashchynskyi and Liashchynskyi, 2019) to select the most promising candidate.

A NAS framework can be divided into three components. The first component is the search space that defines which architectures are allowed; it can be reduced by incorporating prior knowledge about the properties that architectures must have to be suitable for a task. Second, the search strategy indicates how the NAS algorithm explores the search space to find optimal or near-optimal architectures. Finally, the performance estimation strategy refers to the process of estimating the performance of the generated networks: the simplest option is to perform training and validation of the architecture on the data.

NAS generally aims to achieve the best test accuracy without considering the computational cost of inference thus, the generated networks might not be suitable for embedded systems.

In Talbi (2021), the authors propose a unified method for describing various optimization algorithms that focus on the common and important search components of optimization algorithms. They also extend this unified methodology to advanced optimization approaches, such as surrogate, multi-objective, and parallel optimization. The authors propose several constraint management strategies distinguishing them into categories as follows:

1. Rejection: only feasible solutions are retained during the optimization process, and infeasible solutions are automatically discarded (Dong et al., 2018; Hsu et al., 2018; Liberis et al., 2021).
2. Penalization: all solutions are considered, but those that are not feasible are penalized. The objective function is extended by a penalty function. This is the most popular approach, in which many alternatives have been used to define penalties (Veniat and Denoyer, 2018; Zhou et al., 2018; Tan et al., 2019; Liberis et al., 2021).
3. Repair: heuristic algorithms that transform an unfeasible solution into a feasible solution (He and Sun, 2015; Tan and Le, 2019).
4. Preserving: strategies that incorporate problem-specific knowledge into encoding and search operators to generate only feasible solutions. This can reduce the size of the search space and thus simplify the search process (Lu et al., 2018; Wang et al., 2018).

2.3 Optimization

Optimization is a branch of applied mathematics that studies the theory and methods for finding the maximum and minimum points of a mathematical function within a specified domain.

A simple example of an optimization problem is maximizing or minimizing a real function of a real variable over a given interval. More generally, optimization involves finding a sufficiently optimal value for some objective function in a given domain (or input). Adding more than one objective to an optimization problem increases its complexity. In this case, we have a multi-objective optimization (MOO).

2.3.1 MOO

A MOO problem is defined as follows (Equation 2):

$$\underset{x\in X}{\text{minimize}}\, F(x) = [f_1(x), f_2(x), \ldots, f_n(x)]^T \tag{2}$$

where the integer $n \geq 2$ is the number of objectives, and $X \subseteq \mathbb{R}^n$ is the feasible set (Hwang and Masud, 2012).

Due to their conflicting nature, all objectives cannot be optimized simultaneously. Consequently, most MOO approaches aim at recovering the Pareto front, which can be defined as the set of Pareto optimal points. A point is considered Pareto optimal if it cannot be improved in any of the objectives without degrading another objective.

MOO methods can be essentially divided into two classes:

1. Methods that generate single points that are candidates to be Pareto points;
2. Methods that generate optimal approximation of the Pareto front,

In the first class, a reference vector $z^{ref} \in \mathbb{R}^n$ in the space of objectives is defined, and a solution in the space of variables is determined to minimize the distance between the vector of objective functions and the reference (Miettinen, 1999). Typically, one uses the ideal vector of goals Z^* defined by its components as a reference vector (Equation 3):

$$Z_i^* = \min_{x\in X} f_i(x) \tag{3}$$

Therefore, we can define the function that minimizes the distance between the value of the multi-objective function $F(x)$ and the ideal vector Z^* as the goal function (Equation 4):

$$\min_{x\in X} ||F(x) - Z^*|| \tag{4}$$

In the second class, a representative approach is known as the method of weights. Consider the following problem (Equation 5):

$$\min_{x\in X} \sum_{i=1}^{N} w_i f_i(x) \tag{5}$$

where the weights w_i are such that $w_i > 0$ and $\sum_{i=1}^{N} w_i = 1$. As the vector of weights w varies, different Pareto points can be obtained.

2.4 Integration of constraints and DL

One of the main challenges of ML models is the satisfaction of physical constraints. Without these constraints, ML models, while optimizing the loss function, may deviate from known physical principles. Constraints can be applied in two ways: soft constraints and hard constraints. The latter must be satisfied by any feasible solution model. On the other hand, a soft constraint can be violated, but violation of the constraint results in a penalty in the objective function (often, the greater the amount by which the constraint is violated, the greater the penalty). ML models with hard constraints have some advantages over soft constraints, such as more robust and accurate predictions, but are usually difficult to optimize due to the strict adherence to constraints.

Integrating constraints with DL presents a challenge because logic is discrete, symbolic, and semantic, while DL is continuous, smooth, and differentiable. However, there exist different methods for the integration, such as penalty, Lagrange multipliers, and probabilistic interpretation.

2.4.1 Penalty

Penalty methods are algorithms for finding local minima or maxima of a function subject to constraints. They transform the problem with constraints into a problem or problems of unconstrained optimization. The unconstrained problems are formed by adding a term, the *penalty function*, to the objective function that consists of a *penalty factor* multiplied by a measure of violation of the constraints. The measure of violation is non-zero when the constraints are violated and becomes zero in the region where the constraints are not violated.

Consider the following optimization problem (Equation 6):

$$\begin{aligned} &\text{minimize } f(\theta) \\ &\text{subject to}: c(\theta) \leq 0 \end{aligned} \tag{6}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function, $c : \mathbb{R}^n \rightarrow \mathbb{R}^c$ is the constraints function. Penalty methods replace the problem with one or more problems without constraint of form (Equation 7):

$$\text{minimize } \Phi(\mathbf{x}) = f(\mathbf{x}) + \phi g(c(\theta)) \tag{7}$$

where ϕ is a *penalty factor* and $g(\cdot)$ is the *penalty function*.

2.4.2 Lagrange multipliers

Let θ_0 be an optimal solution to the following optimization problem (Equation 8):

$$\begin{aligned} &\text{minimize } f(\theta) \\ &\textit{subject to}: c(\theta) = 0 \end{aligned} \tag{8}$$

where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is the objective function, and $c : \mathbb{R}^n \rightarrow \mathbb{R}^c$ is the constraints function, both having continuous first derivatives. We introduce a new variable (λ) called a Lagrange multiplier and study the Lagrange function defined by Equation (9):

$$\mathcal{L}(\theta, \lambda) = f(\theta) - \lambda c(\theta) \tag{9}$$

The Lagrange multipliers theorem states that if $f(\theta_0)$ is a minimum of $f(\theta)$ for the original constrained problem and $\nabla c(\theta_0) \neq 0$, then there exists a λ_0 such that $\nabla \mathcal{L}(\theta_0, \lambda_0) = 0$, i.e., (θ_0, λ_0) is a stationary point for the Lagrange function. Lagrange multiplier methods generate a class of algorithms for solving constrained optimization problems [e.g., the Augmented Lagrangian method (Bertsekas, 1997)].

2.4.3 Probabilistic interpretation

Another way to integrate constraints with DL is to use a probabilistic interpretation. The output of a NN is a probability distribution over the classes (in the case of classification). If we have constraints on the output, we can measure how close the output is to satisfying these constraints. Let the NN output be $(x_0, x_1, x_2) \in [0, 1]$ and the constraint be that exactly one of these values should be true (i.e., one-hot encoding):

$$Exactly - one = \begin{cases} a \wedge \neg b \wedge \neg c \\ \vee \\ \neg a \wedge b \wedge \neg c \\ \vee \\ \neg a \wedge \neg b \wedge c \end{cases} \tag{10}$$

where $a, b, c \in \{0, 1\}$. Then, the probability that the constraint in Equation (10) is satisfied is given by Equation (11):

$$\begin{aligned} &x_0(1 - x_1)(1 - x_2) + \\ &(1 - x_0)x_1(1 - x_2) + \\ &(1 - x_0)(1 - x_1)x_2 \end{aligned} \tag{11}$$

In other words, we can measure the probability that the constraint is satisfied given the output of the network.

3 Constrained neural architecture search

The purpose of this section is to illustrate different NAS approaches and their possible integration to generate NNs that meet the physical constraints required by embedded systems (e.g., latency, memory, and energy). Therefore, we mainly see penalization and rejection strategies. We also consider AutoML as a multi-objective problem in which many different and conflicting objectives are optimized.

3.1 Approches to NAS

In Elsken et al. (2019), Elsken et al. provide an overview of existing work in this field, classifying it along three dimensions: search space, search strategy, and performance estimation strategy.

Morphism-based NAS systems start from a basic network and, iteration after iteration, modify its structure, including changes in depth, width, kernel size, and even subnetworks. In this survey, we will focus on three types of morphism-based NAS systems, looking at an example for each:

a) An AutoML system that makes several search strategies available (Jin et al., 2019).
b) A symbolic tuner that exploits symbolic rules and Bayesian optimization to explore the search space (Fraccaroli et al., 2021, 2022);
c) Some tuners for microcontroller systems that uses multi-objective optimization (Dong et al., 2018; Tan et al., 2019; Liberis et al., 2021).

AutoKeras (Jin et al., 2019), an open source system based on Keras, is an example of the AutoML system that offers several search strategies. The goal is to allow domain experts who are not familiar with machine learning technologies to easily use machine learning techniques. AutoKeras provides several tools to define and explore the search space through different algorithms (e.g., Bayesian optimization, random search, and grid search) and strategies (e.g., rejection and penalization). Symbolic DNN-Tuner (Fraccaroli et al., 2021) uses Bayesian optimization (BO) that is the state-of-the-art HPO algorithm for DL. BO keeps track of past results and uses them to build a probabilistic model, constructing a probability density of the HP space. Symbolic DNN-Tuner (Fraccaroli et al., 2021) aims to improve BO applied to DNNs through an analysis of network results on training and validation sets. The system applies symbolic tuning rules, implemented in probabilistic logic programming (PLP) (Riguzzi, 2022). The results obtained from the training and validation phases are logically evaluated, and by applying symbolic tuning rules, the network architecture and its HPs are corrected, leading to improved performance.

Figure 1 shows the architecture of Symbolic DNN-Tuner where the neural block returns values from the trained network that are passed both to the Improvement Checker (2), which checks the improvement of the network, and to the symbolic program (1) consisting of three parts: facts, which store the data obtained from the neural block; diagnosis, which analyzes the behavioral problems of DNNs; and tuning, which is composed of the Symbolic Tuning Rules. Using ProbLog (De Raedt et al., 2007) inference, it is possible to query this program and obtain the tuning actions (TAs) (3). The TAs are then passed to the neural block and applied to the DNN structure or HP search space (4).

Finally, μNAS (Liberis et al., 2021) is an example of a NAS for microcontrollers. This system focuses on using deep learning to add computational intelligence to small personal IoT devices. This would allow computations to be performed locally, ensuring that the user's data remains on the device. As a result, it achieves a greater degree of privacy and autonomy. IoT devices are powered by microcontroller units (MCUs). MCUs are ultra-small computers with very limited resources contained in a single chip. This allows MCUs to be cheaper and more energy-efficient than desktop devices or cell phones. However, these advantages lead to drastically reduced computing power. In Liberis et al. (2021), the authors propose the use of a reduced search space that adheres to the constraints on the limited resources and a multi-objective function to explore the search spaces. The MOO aims to find the optimal parameters α^* defined as follows (Equation 12):

$$\begin{aligned}\alpha^* &= \arg\min_{\alpha\in\mathcal{S}} \mathcal{L}(\alpha) \\ &= \arg\min_{\alpha\in\mathcal{S}} \Big(1.0 - \mathit{ValAccuracy}(\alpha), \\ &\qquad \mathit{ModelSize}(\alpha), \\ &\qquad \mathit{PeakMemUsage}(\alpha), \\ &\qquad \mathit{Latency}(\alpha)\Big)\end{aligned} \tag{12}$$

where *ValAccuracy* is a measure of the quality of the network, *ModelSize* is the size of the network, *PeakMemUsage* is the maximum number of parameters stored at a time, and *Latency* is the time it takes to perform a single inference. The Liberis and Lane (2019) algorithm was used to compute the maximum number of stored parameters. To optimize the multiobjective function, the authors used the method introduced by Paria et al. (2020).

In Liberis et al. (2021), the authors showed that, with proper design of the search space and explicit identification of physical constraints, it is possible to create a NAS system that discovers resource-efficient models for a variety of image classification tasks. Table 1 shows the main results of the μNAS compared with other constrained NAS. They use multiply accumulate operations (MACs) as a measure to quantify the latency time as a function of the model size.

Other examples of physically constrained NAS using MOO can be found in the literature. For example, in Dong et al. (2018), the authors propose Device-aware Progressive Search for Pareto-optimal Neural Architectures (DPP-Net), which optimizes device-related and device-independent targets by applying multi-objective optimization. DPP-Net employs a compact search space inspired by state-of-the-art mobile CNNs and further improves the search efficiency by adopting progressive search (Liu et al., 2017). The authors test their system on CIFAR-10 and they compere it with the state-of-the-art (Huang et al., 2017, 2018; Zoph et al., 2018). We show these results in Table 2.

In Tan et al. (2019), the authors proposed an automated mobile neural architecture search (MNAS) approach that explicitly incorporates model latency into the main objective so that the search can identify a model that achieves an optimal trade-off between accuracy and latency. The authors define the objective function as follows:

Given a model m, let $ACC(m)$ denote its accuracy on the target task, $LAT(m)$ the inference latency on the target mobile platform, and T the target latency. A common method is to treat T as a hard constraint and maximize the accuracy under this constraint (Equation 13):

$$\begin{aligned}&\underset{m}{\text{maximize}} \quad ACC(m) \\ &\text{subject to} \quad LAT(m) \leq T\end{aligned} \tag{13}$$

Tan et al. (2019) use a custom weighted product method to approximate Pareto optimal solutions, with an optimization objective defined as Equation (14):

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T}\right]^w \tag{14}$$

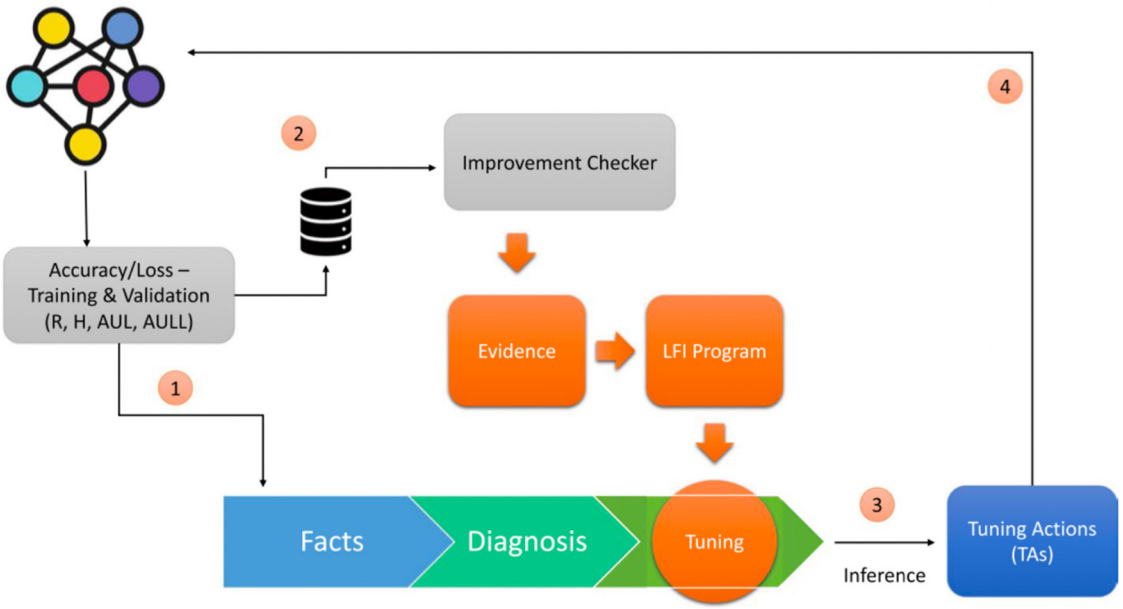

FIGURE 1
Symbolic DNN-tuner architecture [reprinted with permission from Fraccaroli et al. (2021)].

TABLE 1 Pareto-optimal architectures discovered by *μNAS* vs. other Resource—Constrained NAS.

Dataset	Model	Acc. (%)	Model size	MACs
MNIST	SpArSe (Fedorov et al., 2019)	98.64	2770	-
	BonsaiOpt (Kumar et al., 2017)	94.38	490	-
	ProtoNN (Gupta et al., 2017)	95.88	63′900	-
	μNAS(1174 steps, 1 GPU-day)	99.19	480	28.6 K
CIFAR-10	LEMONADE (Elsken et al., 2018)	≈ 91.77	10K	-
	μNAS(4205 steps, 23 GPU-days)	86.49	11.4 K	384 K
Speech Commands	RENA (Zhou et al., 2018)	94.04	47 K	≈ 700*M*
	DS-CNN (Zhang et al., 2017)	94.45	< 38.6 K	≈ 2.7*M*
	MCUNet (Lin et al., 2020)	91.20	< 1 M	-
	μNAS(1960 steps, 39 GPU-days)	95.36	37 K	1.1 M

where w is an application-specific constant. The authors test their system on ImageNet classification and compare their model with both manually designed mobile models and other automated approaches. The results are shown in Table 3.

Zhou et al. (2018) developed the Resource-Efficient Neural Architect (RENA), a resource-limited efficient NAS that uses reinforcement learning with network embeddings. The framework consists of a network of policies to generate actions that define the architecture of the neural network. The environment provides the performance of the trained neural network and its resource utilization. RENA uses a policy gradient with accumulated rewards to train the policy network. To find neural architectures that satisfy multiple resource constraints, a reward based on the model performance should be penalized according to the amount of constraint violation. A hard penalty may be effective for some constraints, but it would be difficult for the controller to learn from very sparse rewards under tight resource constraints. Therefore, RENA uses a soft continuous penalty method to find architectures with high performance while meeting all resource constraints. The results obtained with this framework are shown in Table 4.

A popular case study is the application of machine learning models for the predictive maintenance of IoT edge devices in a factory. Since these devices have limited memory and computing resources, we need to design optimized neural architectures for them, striking a balance between model accuracy and resource efficiency. Using RENA, we are able to develop models that can effectively predict device failures while operating within the constraints of edge devices. The implementation of RENA therefore enables the company to use machine learning for the predictive maintenance of its IoT edge devices efficiently.

TABLE 2 DPP-Net main result on CIFAR-10.

Model	Type	Error rate	Params	FLOPs
DenseNet-BC ($k = 12$) (Huang et al., 2017)	Manual	4.51	0.8M	-
CondenseNet-86 (Huang et al., 2018)	Manual	5.0	0.52M	65.8M
NASNet-B (Zoph et al., 2018)	Auto	3.73	2.6M	-
DPP-Net (Dong et al., 2018)	Auto	4.36 ∼ 5.84	11.39M ∼ 0.45M	1364M ∼ 59.27M

TABLE 3 MNAS main result on ImageNet classification.

Model	Type	Top-1 Acc. (%)	Params	MACs
MobileNetV1 (Howard et al., 2017)	Manual	70.6	4.2M	575M
SqueezeNext (Gholami et al., 2018)	Manual	67.5	3.2M	708M
ShuffleNet (1.5x) (Zhang et al., 2018)	Manual	71.5	3.4M	292M
ShuffleNetV2 (1.5x) (Ma et al., 2018)	Manual	72.6	5.4M	524M
CondenseNet (G=C=4) (Huang et al., 2018)	Manual	71.0	-	299M
MobileNetV2 (Sandler et al., 2018)	Manual	72.0	-	597M
NASNet-A (Zoph et al., 2018)	Auto	74.0	2.9M	274M
AmoebaNet-A (Zhou et al., 2018)	Auto	74.5	4.8M	529M
PNASNet (Liu C. et al., 2018)	Auto	74.2	3.4M	300M
DARTS (Liu H. et al., 2018)	Auto	73.1	6.9M	585M
MNAS	Auto	75.2 ∼ 76.7	3.9M ∼ 5.2	312M ∼ 403M

Another case study example are TinyML models designed specifically for wearable devices with limited memory and processing capacity. They are optimized using techniques such as pruning and quantization to save energy and resources while still providing accurate predictions. For example, on environmental monitoring sensors, a system such as μNAS can be applied to design compact neural architectures for low-power microcontrollers, enabling real-time data analysis and maximizing battery life. These examples highlight the effectiveness of constrained NAS approaches, such as RENA, DNN-Tuner, and μNAS, in designing neural architectures tailored to specific constraints.

3.2 Integrated system

The methods outlined above shed light on the distinctions between various NAS approaches. However, the question arises: can MOO methods be seamlessly integrated with each other? Moreover, is it feasible to introduce constraints without resorting to MOO?

One prospective avenue is to introduce into the Symbolic DNN-Tuner (Fraccaroli et al., 2021) a multi-objective function, akin to the μNAS model, which stands as a subject for future exploration. In such a scenario, the optimization function would be multi-objective in nature, aiming to optimize the network while considering physical constraints. This function could be optimized using one of the methods delineated in Section 2.3.1. Following this optimization, incorporating tuning ations (TAs) would enable the system to intervene when the networks violate the constraints. Table 5 illustrates the tuning actions employed by Symbolic DNN-Tuner (Fraccaroli et al., 2021), alongside potential additional rules to satisfy new constraints, presented in the lower part of the table.

Another way to impose a constraint might be to add a penalty to the metric used to evaluate the networks. For example, AutoKeras adds a penalty term to the network loss. This means that the more a constraint is violated, the greater the penalty will be. In the previous example of physical constraints, assuming an upper limit on FLOPS, the closer the network to the limit, the greater the penalty. Thus, a NAS will tend to prefer smaller networks since large networks would lead to higher loss functions.

4 Constrained networks

The widespread success of DL has led several researchers to look for ways to improve it through constraint-based domain knowledge. There are contexts in which purely data-driven models are not ideal, such as when data are sparse or learning tasks are very challenging. It is possible to achieve a significant increase in the performance of NNs by exploiting domain knowledge, using problem-specific information for simplifying the training process (e.g., the shape of the output, the data generation process, the experience of a domain expert, etc.). Therefore, it makes sense to exploit domain information in order to not to start from scratch when tackling difficult learning tasks for NNs.

In various fields, constraints are applied during the training of neural networks to improve the performance and reliability of the models. In the case of unbalanced

TABLE 4 RENA main result on speech commands.

Model	Resource constraint	Parameters	Accuracy (%)	FLOPs
GRU (Zhang et al., 2017)	-	0.093 M	92.94	0.68 B
DS-CNN (Zhang et al., 2017)	-	0.023 M	93.39	6.07 B
CRNN (Zhang et al., 2017)	-	2.447 M	94.40	46.21 B
RENA	-	0.143 M	95.81	3.39 B
RENA	Model size < 0.05 M	0.047 M	94.04	1.40 B
RENA	Model size < 0.1 M	0.067 M	94.82	6.53 B
RENA	Comp. complexity < 1 GFLOPs	0.425 M	93.16	0.89 B
RENA	Comp. complexity < 5 GFLOPs Model size < 0.1 M	0.171 M	95.02	3.30 B
RENA	Comp. complexity < 1 GFLOPs	0.035 M	93.07	1.0 B

datasets, penalties for misclassification errors on minority classes are applied to address the issue (Sangalli et al., 2021). Anomaly detection systems could use constraints to penalize misclassification of normal cases as anomalies during the training process. Finally, medical diagnostic systems could incorporate medical constraints during training to ensure that the provided diagnoses align with clinical evidence and medical guidelines. These constraints play a crucial role in enhancing the effectiveness and validity of the neural network models in these respective domains.

Below, we will illustrate some applications of constrained NNs; in particular we will see examples of constraints applied through penalty functions, such as Lagrange multipliers (Section 4.1), applied in unbalanced data and OOD detection contexts. Finally, we will see an example of the application of the probabilistic interpretation (Section 4.2).

4.1 Applications

4.1.1 NNs with unbalanced data

Datasets with unbalanced data are one of the most common problems in ML. However, it is possible to introduce constraints given by context-knowledge so as to mitigate the effect of unbalanced classes.

Sangalli et al. (2021) proposed to see the training of a DNN for binary classification under conditions of class imbalance as a constrained optimization problem. They consider the medical imaging context where applications with data imbalance are ubiquitous (Litjens et al., 2017) and some types of errors are more severe than others. For example, in a diagnosis application, discarding a cancer case as healthy (False Negative) is more costly than classifying a healthy subject as having cancer (False Positive). For this reason, networks working in the medical field generally tend to have high True Positive Rates (TPRs). The authors define a constraint, using the Mann-Whitney statistics (Mann and Whitney, 1947), to maximize the AUC, but to asymmetrically favor reducing False Positives in the presence of high TPR (or low False Negative Rates). They then use the Augmented Lagrangian Method (ALM) (Bertsekas, 1997).

The optimization problem they solve is defined as Equation (15):

$$\begin{aligned} &\arg\min_{\theta} F(\theta) \\ &\text{subject to :} \\ &\sum_{k=1}^{|N|} \max\left(0, -\left(f_\theta(x_j^p) - f_\theta(x_k^n)\right) + \delta\right) = 0, j \in \{1, ..., |P|\} \end{aligned} \tag{15}$$

where, $f_\theta(x)$ indicates output probability of DNN on input x, and $P = \{x_1^p, \ldots, x_{|P|}^p\}$ is the set of positive samples and $N = \{x_1^n, \ldots, x_{|N|}^n\}$ is the set negative samples.

The constraint states that the output of the NN for each sample of the positive class should be larger than the outputs of all of the negative samples by a margin δ. Furthermore, satisfying the constraint would directly guarantee maximal AUC.

Figure 2 shows a toy example. The upper part shows 10 data samples sorted with respect to the output of the classifier, i.e., samples on the right are assumed to produce a higher output than those on the left. The blue area in the lower part of the figure shows the AUC for the toy data samples. Let us consider two different approaches to increase the AUC. Using standard optimization with binary cross entropy as the loss function, we would add the red box while solving the problem in Equation (15) via the method of Lagrange multipliers results in addition to the green box to the blue area. Both optimizations lead to the exacts same improvement in AUC, but only the addition of the green box reduces the FPR while maintaining the same TPR. Two possible extensions for multiclass classification are also proposed in Sangalli et al. (2021). The authors also perform an extensive evaluation of constraints for binary and multiclass image classification problems on both computer vision and medical imaging datasets by simulating different class imbalance ratios. They obtained results showing that constraints improve baseline performance in most cases of both binary and multiclass classification experiments.

4.1.2 OOD detection with constrained networks

Katz-Samuels et al. (2022) proposed a framework for out-of-distribution (OOD) detection in the wild, dubbed WOODS (Wild

TABLE 5 Problem, symptoms, and tuning actions.

Problem	Symptoms	Tuning actions (TAs)
Overfitting	Gap between accuracy or loss in training and validation	Regularization and Batch
		Normalization
		Increase dropout
		Data augmentation
Underfitting	High loss	Decrease the learning rate
	Low accuracy	Increase the number of neurons
		Addition of fully connected layers
		Addition of convolutional blocks
Increasing loss	Loss trend analysis	Decrease the learning rate
Fluctuating loss	Fluctuation of the loss	Increase the batch size
		Decrease the learning rate
Low learning rate	Evaluation of the shape of the loss	Increase learning rate
High learning rate	Evaluation of the shape of the loss	Decrease learning rate
Peak Memory Usage	High number of Parameters for layer	Decrease the number of neurons
Model Size	High number of total Parameters	Decrease the number of layers or the number of neurons
Latency	High FLOPS	Decrease the number of layers or the number of neurons

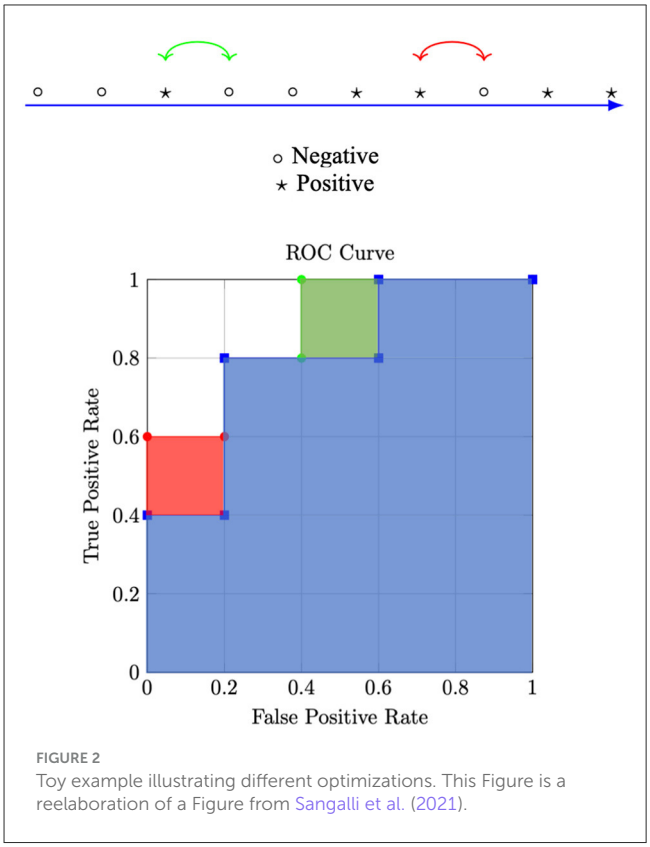

FIGURE 2
Toy example illustrating different optimizations. This Figure is a reelaboration of a Figure from Sangalli et al. (2021).

OOD detection sans-Supervision). In this approach, a pre-trained model to perform in-distribution (ID) classification is deployed in the "wild" open world, where it will encounter large amounts of unlabeled ID and OOD data. In WOODS, the model can be tuned using the "wild" data to perform accurate OOD detection and ID classification.

The authors use wild data because it can be found in huge quantities, can be collected at low cost at the time of implementing an ML system, and often corresponds better to the actual distribution than data collected offline. However, it is difficult to exploit because it is naturally composed of examples of both ID and OOD data.

WOODS is based on constrained optimization and solves it through the Augmented Lagrangian method applied to deep neural networks.

Let $X = \mathbb{R}^d$ denote the input space and $Y = \{1, ..., K\}$ denote the label space. We assume access to the labeled training set $D_{in}^{train} = \{(x_i, y_i)\}_{i=1}^n$, drawn i.i.d. from the joint data distribution P_{XY}. Let $\mathbb{P}_{\text{in}}$ denote the marginal distribution on X , which is also referred to as the ID and $\mathbb{P}_{\text{out}}$ denote different distribution on samples that the model has not been exposed to during training, which is also referred to as the OOD. Let $f_\theta : X \mapsto \mathbb{R}^{|Y|}$ denote a function for the classification task, which predicts the label of an input sample and $g_\omega : X \mapsto \{\text{in}, \text{out}\}$ as the function for OOD detection. WOODS uses the Huber contamination model (Huber, 1992) to model the marginal distribution of the wild data (Equation 16):

$$\mathbb{P}_{\text{wild}} := (1 - \pi)\mathbb{P}_{\text{in}} + \pi\mathbb{P}_{\text{out}}, \text{ with } \pi \in (0, 1] \tag{16}$$

The objective can be described as follows (Equation 17):

$$\begin{aligned} \underset{\theta}{\text{minimize}}\ & \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\text{wild}}}\left(\mathbb{1}\left\{g_\theta(\mathbf{x}) = \text{ in }\right\}\right) \\ \text{s.t. } & \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{\text{in}}}\left(\mathbb{1}\left\{g_\theta(\mathbf{x}) = \text{ out }\right\}\right) \leq \alpha \\ & \mathbb{E}_{(\mathbf{x},y)\sim\mathbb{P}_{\mathcal{XY}}}\left(\mathbb{1}\left\{f_\theta(\mathbf{x}) \neq y\right\}\right) \leq \tau. \end{aligned} \tag{17}$$

where $\mathbb{1}\{\cdot\}$ is the indicator function, and $\alpha, \tau \in [0, 1]$.

In other words, the authors aim to minimize the number of samples that are classified as ID by applying two constraints:

1. The error of declaring an ID data from $\mathbb{P}_{\text{in}}$ as OOD must be low;
2. The multiclass classification model must maintain the best attainable (or nearly attainable) accuracy of a base classifier designed without an OOD detection requirement.

4.2 Application of the probabilistic interpretation

When the output of a model is structured, knowledge of the structure type (one-hot encoder, ranking, path graph, etc.) can be useful information to integrate in the training of a neural network. In Xu et al. (2018), define a semantic loss to calculate how close the network output is to satisfying a given constraint, thus exploiting

the concepts seen in Section 2.4. The idea is to penalize networks that do not satisfy constraints on the output.

The semantic loss $L^s(\alpha, p)$ is a function involving the propositional logic sentence α and a set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$. It also incorporates a probability vector p associated with these variables, where each element p_i represents the predicted probability for the variable X_i and corresponds to a single output from the neural network. The semantic loss is defined by the following equation:

$$L^s(\alpha, p) \propto -log \sum_{x \models \alpha} \prod_{i\,:\,x \models X_i} p_i \prod_{i\,:\,x \models \neg X_i} (1 - p_j) \tag{18}$$

Semantic loss is simply another regularization term that can be directly added to an existing loss function. More specifically, given some weight w, the new loss becomes (Equation 19):

$$existing\ loss + w \cdot semantic\ loss \tag{19}$$

For example, for given the exactly-one constraint seen in Section 2.4, the semantic loss is given by Equation (20):

$$L^s(exactly - one, p) \propto -log \sum_{i=1}^{n} p_i \prod_{j=1, j \neq i}^{n} (1 - p_i) \tag{20}$$

In general, for arbitrary constraints, computing semantic loss using Equation (18) is computationally expensive. Therefore, advanced automated reasoning, particularly knowledge compilation, is required (Anderson, 1983).

The authors show that semantic loss is not effective enough for simple supervised classification problems. However, it is useful, provided the output domain is a sufficiently complex space. A prime example of this is the path graph problem, where the goal is to predict the shortest path and the constraint is for the output to be a valid path. By testing the same network with and without semantic loss, Xu et al. (2018) show that the accuracy is significantly improved for consistent and constrained paths.

5 Conclusion and future work

The integration of domain knowledge, expressible in the form of constraints, into deep neural networks has gained research interest in recent years. In this study, we showed several AutoML and NAS frameworks that use multi-objective optimization and discussed how they compare with the state-of-the-art. As future work, we proposed utilizing knowledge to integrate Symbolic DNN-Tuner with multi-objective functions. Then, we presented some approaches that use domain knowledge in NNs. In particular, we showed how this integration can be done using Lagrange multipliers, both for domain constraints and detection cases such as OOD detection. We also discussed semantic loss, which calculates how close the network output is to satisfying a given constraint.

In summary, by combining developments of both approaches, AutoML and constrained neural networks, we obtain several promising applications of constraints in neural systems, resulting in more robust learning capabilities for neural-symbolic systems.

Author contributions

AB: Investigation, Writing – original draft, Writing – review & editing. MF: Conceptualization, Supervision, Writing – review & editing. EL: Methodology, Supervision, Writing – review & editing. FR: Methodology, Supervision, Writing – review & editing.

Funding

The author(s) declare that no financial support was received for the research, authorship, and/or publication of this article.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The author(s) declared that they were an editorial board member of Frontiers, at the time of submission. This had no impact on the peer review process and the final decision.

Publisher's note

References

Agre, G., and Koprinska, I. (1996). "Case-based refinement of knowledge-based neural networks," in *The International Conference" Intelligent Systems: A Semiotic Perspective* (Piscataway, NJ), 20–23.

Anderson, J. R. (1983). "Knowledge compilation," in *Machine Learning: An Artificial Intelligence Approach* (Berlin), 289.

Avila Garcez, A. S., and Zaverucha, G. (1999). The connectionist inductive learning and logic programming system. *Appl. Intellig.* 11, 59–77. doi: 10.1023/A:1008328630915

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). Segnet: a deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 39, 2481–2495. doi: 10.1109/TPAMI.2016.2644615

Benmeziane, H., Maghraoui, K. E., Ouarnoughi, H., Niar, S., Wistuba, M., and Wang, N. (2021). A comprehensive survey on hardware-aware neural architecture search. *arXiv*[preprint] arXiv:2101.09336. doi: 10.24963/ijcai.2021/592

Bergstra, J., Yamins, D., and Cox, D. (2013). "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference on Machine Learning* (New York: PMLR), 115–123.

Bertsekas, D. P. (1997). Nonlinear programming. *J. Operat. Res. Soc*. 48, 334–334. doi: 10.1057/palgrave.jors.2600425

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: encoder-decoder approaches. *arXiv*[preprint] arXiv:1409.1259. doi: 10.3115/v1/W14-4012

Dai, Z., Liu, H., Le, Q. V., and Tan, M. (2021). Coatnet: Marrying convolution and attention for all data sizes. *Adv. Neural Inf. Process. Syst*. 34, 3965–3977. doi: 10.48550/arXiv.2106.04803

De Raedt, L., Kimmig, A., and Toivonen, H. (2007). "Problog: A probabilistic prolog and its application in link discovery," in *IJCAI 2007, the 20th International Joint Conference on Artificial Intelligence* (Messe Wien, Vienna: IJCAI), 2462–2467.

Deng, L., and Yu, D. (2014). Deep learning: methods and applications. *Foundat. Trends Signal Proc.* 7, 197–387. doi: 10.1561/2000000039

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. (2018). "DPP-Net: Device-aware progressive search for pareto-optimal neural architectures," in *The European Conference on Computer Vision (ECCV)*, 517–531.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: a survey. *J. Mach. Learn. Res*. 20, 1997–2017. doi: 10.1007/978-3-030-05318-5_3

Fedorov, I., Adams, R. P., Mattina, M., and Whatmough, P. (2019). Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. *Adv. Neural Inf. Process. Syst*. 32, 4977–4989. doi: 10.48550/arXiv.1905.12107

Feurer, M., and Hutter, F. (2019). "Hyperparameter optimization," in *Automated Machine Learning* (Cham, Berlin: Springer), 3–33.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28 (NIPS 2015)* (Montreal, QC: NIPS 2015), 28.

Fraccaroli, M., Lamma, E., and Riguzzi, F. (2021). Symbolic DNN-tuner. *Mach. Learn*. 111, 625–650. doi: 10.1007/s10994-021-06097-1

Fraccaroli, M., Lamma, E., and Riguzzi, F. (2022). Symbolic dnn-tuner: a python and problog-based system for optimizing deep neural networks hyperparameters. *SoftwareX* 17:100957. doi: 10.1016/j.softx.2021.100957

Garcez, A., d., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., et al. (2019). Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. doi: 10.48550/arXiv.1905.06088

Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., et al. (2018). "Squeezenext: Hardware-aware neural network design," in *The IEEE Conference on Computer Vision and Pattern Recognition Workshops* (Seattle: IEEE), 1638–1647.

Gijsbers, P., LeDell, E., Thomas, J., Poirier, S., Bischl, B., and Vanschoren, J. (2019). An open source AutoML benchmark. *arXiv*[preprint] arXiv:1907.00909. doi: 10.48550/arXiv.1907.00909

Gupta, C., Suggala, A. S., Goyal, A., Simhadri, H. V., Paranjape, B., Kumar, A., et al. (2017). "Protonn: Compressed and accurate knn for resource-scarce devices," in *International Conference on Machine Learning* (New York: PMLR), 1331–1340.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). "Mask R-CNN," in *The IEEE International Conference on Computer Vision* (Paris: IEEE), 2961–2969.

He, K., and Sun, J. (2015). "Convolutional neural networks at constrained time cost," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 5353–5360.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 770–778.

He, X., Zhao, K., and Chu, X. (2021). Automl: A survey of the state-of-the-art. *Knowl.-Based Syst*. 212:106622. doi: 10.1016/j.knosys.2020.106622

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv*[preprint] arXiv:1704.04861. doi: 10.48550/arXiv.1704.04861

Hsu, C.-H., Chang, S.-H., Liang, J.-H., Chou, H.-P., Liu, C.-H., Chang, S.-C., et al. (2018). Monas: multi-objective neural architecture search using reinforcement learning. *arXiv*[preprint] arXiv:1806.10332. doi: 10.48550/arXiv.1806.10332

Huang, G., Liu, S., Van der Maaten, L., and Weinberger, K. Q. (2018). "Condensenet: an efficient densenet using learned group convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 2752–2761.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 4700–4708.

Huber, P. J. (1992). "Robust estimation of a location parameter," in *Breakthroughs in Statistics* (Berlin: Springer), 492–518.

Hwang, C.-L., and Masud, A. S. M. (2012). *Multiple Objective Decision Making–Methods and Applications: a State-of-the-Art Survey, Volume 164*. Berlin: Springer Science & Business Media.

Jin, H., Song, Q., and Hu, X. (2019). "Auto-Keras: An efficient neural architecture search system," in *25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (New York, NY), 1946–1956.

Katz-Samuels, J., Nakhleh, J. B., Nowak, R., and Li, Y. (2022). "Training OOD detectors in their natural habitats," in *International Conference on Machine Learning* (New York: PMLR), 10848–10865.

Klein, A., Falkner, S., Mansur, N., and Hutter, F. (2017). "Robo: A flexible and robust bayesian optimization framework in python," in *NIPS 2017 Bayesian Optimization Workshop* (Cambridge, MA: MIT Press), 4–9.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Commun. ACM* 60, 84–90. doi: 10.1145/3065386

Kumar, A., Goyal, S., and Varma, M. (2017). "Resource-efficient machine learning in 2 kb ram for the internet of things," in *International Conference on Machine Learning* (New York: PMLR), 1935–1944.

Lamb, L. C., Garcez, A., Gori, M., Prates, M., Avelar, P., and Vardi, M. (2020). Graph neural networks meet neural-symbolic computing: a survey and perspective. *arXiv*[preprint] arXiv:2003.00330. doi: 10.24963/ijcai.2020/679

Liashchynskyi, P., and Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv*[preprint] arXiv:1912.06059. doi: 10.48550/arXiv.1912.06059

Liberis, E., Dudziak, Ł., and Lane, N. D. (2021). "μNAS: constrained neural architecture search for microcontrollers," in *The 1st Workshop on Machine Learning and Systems* (Edinburgh: EuroMLSys'21), 70–79.

Liberis, E., and Lane, N. D. (2019). Neural networks on microcontrollers: saving memory at inference via operator reordering. *arXiv*[preprint] arXiv:1910.05110. doi: 10.48550/arXiv.1910.05110

Lin, J., Chen, W.-M., Lin, Y., Gan, C., Han, S., et al. (2020). Mcunet: Tiny deep learning on iot devices. *Adv. Neural Inf. Process. Syst*. 33, 11711–11722. doi: 10.48550/arXiv.2007.10319

Lin, T., Wang, Y., Liu, X., and Qiu, X. (2021). A survey of transformers. *arXiv*[preprint] arXiv:2106.04554. doi: 10.48550/arXiv.2106.04554

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). "Focal loss for dense object detection," in *The IEEE International Conference on Computer Vision* (Paris: IEEE), 2980–2988. doi: 10.48550/arXiv.1708.02002

Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., et al. (2017). A survey on deep learning in medical image analysis. *Med. Image Anal*. 42, 60–88. doi: 10.1016/j.media.2017.07.005

Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., et al. (2018). "Progressive neural architecture search," in *The European Conference on Computer Vision (ECCV)* (Tel Aviv: IEEE), 19–34.

Liu, H., Simonyan, K., Vinyals, O., Fernando, C., and Kavukcuoglu, K. (2017). Hierarchical representations for efficient architecture search.

Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: differentiable architecture search. *arXiv*[preprint] arXiv:1806.09055.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., et al. (2018). *Nsga-net: A Multi-Objective Genetic Algorithm for Neural Architecture Search*. New York, NY.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. (2018). "Shufflenet v2: practical guidelines for efficient cnn architecture design," in *The European Conference on Computer Vision (ECCV)* (Tel Aviv: ECCV), 116–131.

Mann, H. B., and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *Ann. Mathem. Statist*. 18, 50–60. doi: 10.1214/aoms/1177730491

McCulloch, W. S., and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys*. 5, 115–133. doi: 10.1007/BF02478259

Melis, G., Dyer, C., and Blunsom, P. (2017). On the state of the art of evaluation in neural language models. *arXiv*[preprint] arXiv:1707.05589. doi: 10.48550/arXiv.1707.05589

Miettinen, K. (1999). *Nonlinear Multiobjective Optimization, Volume 12*. Berlin: Springer Science & Business Media.

Paria, B., Kandasamy, K., and Póczos, B. (2020). "A flexible framework for multi-objective bayesian optimization using random scalarizations," in *Uncertainty in Artificial Intelligence* (New York: PMLR), 766–776.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). "You only look once: Unified, real-time object detection," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Las Vegas: IEEE), 779–788.

Riguzzi, F. (2022). *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning*. Denmark: River Publishers.

Sahin, S., Tolun, M. R., and Hassanpour, R. (2012). Hybrid expert systems: a survey of current approaches and applications. *Expert Syst. Appl*. 39, 4609–4617. doi: 10.1016/j.eswa.2011.08.130

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). "Mobilenetv2: Inverted residuals and linear bottlenecks," in *The*

IEEE Conference on Computer Vision and Pattern Recognition (Seattle: IEEE), 4510–4520.

Sangalli, S., Erdil, E., Hötker, A., Donati, O., and Konukoglu, E. (2021). "Constrained optimization to train neural networks on critical and under-represented classes," in *Advances in Neural Information Processing Systems* (Cambridge, MA: MIT Press), 34.

Simonyan, K., and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv*[preprint] arXiv:1409.1556. doi: 10.48550/arXiv.1409.1556

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artif. Intell.* 46, 159–216. doi: 10.1016/0004-3702(90)90007-M

Snoek, J., Larochelle, H., and Adams, R. P. (2012). "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems* (Cambridge, MA: MIT Press), 25.

Staudemeyer, R. C., and Morris, E. R. (2019). Understanding LSTM-a tutorial into long short-term memory recurrent neural networks. *arXiv*[preprint] arXiv:1909.09586. doi: 10.48550/arXiv.1909.09586

Swersky, K., Snoek, J., and Adams, R. P. (2013). "Multi-task bayesian optimization," in *Advances in Neural Information Processing Systems*, 26.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 1–9.

Talbi, E.-G. (2021). Automated design of deep neural networks: a survey and unified taxonomy. *ACM Comp. Surv. (CSUR)* 54, 1–37. doi: 10.1145/3439730

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., et al. (2019). "MNASNET: Platform-aware neural architecture search for mobile," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 2820–2828.

Tan, M., and Le, Q. (2019). "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning* (New York: PMLR), 6105–6114.

Tenney, I., Das, D., and Pavlick, E. (2019). BERT rediscovers the classical NLP pipeline. *arXiv*. arXiv:1905.05950. doi: 10.18653/v1/P19-1452

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY), 847–855.

Turing, A. M. (2009). *Computing Machinery and Intelligence*. Berlin: Springer.

Vanschoren, J. (2019). "Meta-learning," in *Automated Machine Learning* (Berlin: Springer), 35–61.

Veniat, T., and Denoyer, L. (2018). "Learning time/memory-efficient deep architectures with budgeted super networks," in *IEEE Conference on Computer Vision and Pattern Recognition* (Seattle: IEEE), 3492–3500.

Wang, B., Sun, Y., Xue, B., and Zhang, M. (2018). "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *2018 IEEE Congress on Evolutionary Computation (CEC)* (Rio de Janeiro: IEEE), 1–8.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Van den Broeck, G. (2018). "A semantic loss function for deep learning with symbolic knowledge," in International Conference on Machine Learning (New York: PMLR), 5502–5511.

Zhang, X., Zhou, X., Lin, M., and Sun, J. (2018). "Shufflenet: an extremely efficient convolutional neural network for mobile devices," in *IEEE Conference on Computer Vision and Pattern Recognition* (Piscataway, NJ: IEEE), 6848–6856.

Zhang, Y., Suda, N., Lai, L., and Chandra, V. (2017). Hello edge: keyword spotting on microcontrollers. *arXiv*[preprint] arXiv:1711.07128. doi: 10.48550/arXiv.1711.07128

Zhou, Y., Ebrahimi, S., Arık, S., Ö., Yu, H., Liu, H., et al. (2018). Resource-efficient neural architect. in *arXiv*[preprint] arXiv:1806.07912. doi: 10.48550/arXiv.1806.07912

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). "Learning transferable architectures for scalable image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition* (Salt Lake City: IEEE), 8697–8710.