Automated Deep Learning: Neural Architecture Search Is Not the End

XUANYI DONG, Complex Adaptive Systems Lab, University of Technology Sydney, Australia and Brain Team, Google Research, USA

DAVID JACOB KEDZIORA, Complex Adaptive Systems Lab, University of Technology Sydney, Australia

KATARZYNA MUSIAL, Complex Adaptive Systems Lab, University of Technology Sydney, Australia

BOGDAN GABRYS, Complex Adaptive Systems Lab, University of Technology Sydney, Australia

Deep learning (DL) has proven to be a highly effective approach for developing models in diverse contexts, including visual perception, speech recognition, and machine translation. However, the end-to-end process for applying DL is not trivial. It requires grappling with problem formulation and context understanding, data engineering, model development, deployment, continuous monitoring and maintenance, and so on. Moreover, each of these steps typically relies heavily on humans, in terms of both knowledge and interactions, which impedes the further advancement and democratization of DL. Consequently, in response to these issues, a new field has emerged over the last few years: automated deep learning (AutoDL). This endeavor seeks to minimize the need for human involvement and is best known for its achievements in neural architecture search (NAS), a topic that has been the focus of several surveys. That stated, NAS is not the be-all and end-all of AutoDL. Accordingly, this review adopts an overarching perspective, examining research efforts into automation across the entirety of an archetypal DL workflow. In so doing, this work also proposes a comprehensive set of ten criteria by which to assess existing work in both individual publications and broader research areas. These criteria are: novelty, solution quality, efficiency, stability, interpretability, reproducibility, engineering quality, scalability, generalizability, and eco-friendliness. Thus, ultimately, this review provides an evaluative overview of AutoDL in the early 2020s, identifying where future opportunities for progress may exist.

Additional Key Words and Phrases: Automated Deep Learning (AutoDL), Neural Architecture Search (NAS), Hyperparameter Optimization (HPO), Automated Data Engineering, Hardware Search, Automated Deployment, Life-long Learning, Persistent Learning, Adaptation, Automated Machine Learning (AutoML), Autonomous Machine Learning (AutonoML), Deep Neural Networks, Deep Learning

1 INTRODUCTION

In the quest for artificial intelligence (AI), history may judge the early 2010s as a mental reset, stimulating a new era of research and development with unrivaled intensity. Within those reformative years, the field of machine learning (ML) witnessed a shifting of priorities and approaches. Two threads of aspiration stand out:

- Deep Learning (DL) – The idea that multi-layered artificial-neuron networks are central to pushing the capabilities of ML.
- Automated Machine Learning (AutoML) – The idea that no part of an ML workflow should necessarily depend on human involvement.

It was inevitable that these two ideologies would eventually converge, fusing into the novel subject of automated deep learning (AutoDL).

Admittedly, while AutoDL is a "hot topic" in 2021, the foundations underlying this surge of activity stretch back for decades. The notion of ML itself [187] was established in the 1950s, aiming to tune mathematical models of desirable functions via automated data-driven algorithms. In time, by the turn of the 21st century, numerous ML models and algorithms would be in practical use, with support vector machines and other kernel methods proving particularly popular [114]. However, the concept of a neuron, inextricably linked to human intelligence, always seemed an
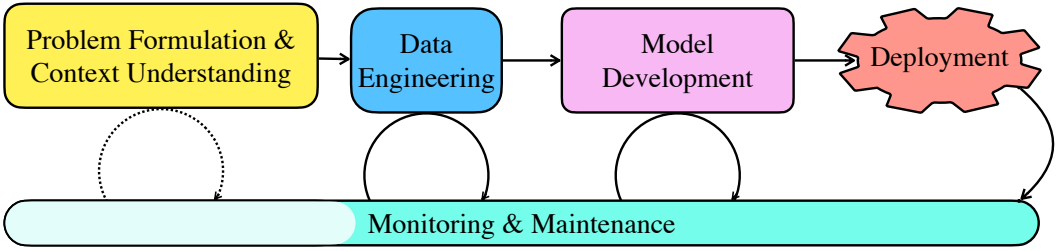
Fig. 1. Schematic of an end-to-end DL workflow, i.e., the processes involved in applying DL to a problem. Traditionally, human decisions are required for every part of this workflow, such as analyzing a problem context, defining an ML task, designing a model, manually tuning hyperparameters, selecting training strategy, etc.

obvious basis for ML. Depicted computationally as early as in the 1940s [185], their representational power in multi-layered arrangements was evident by the late 1960s, exemplified by the proto-DL "group method of data handling" (GMDH) [125]. Since then, with stutters around AI winters, numerous types of neural layers and architectural variants have been proposed and adopted. These include recurrent structures [115, 166], convolutional and downsample layers [86], auto-encoder hierarchies [15], memory mechanisms [208], and gating structures [112]. As a result, the historical successes of artificial neural networks (ANNs) are undeniably many, encompassing handwriting recognition [154], time series prediction [277], video retrieval [131, 297], mitosis detection [48], and so on. Yet the advantages of deep neural networks (DNNs), including their status as universal approximators [116], are countered by the unwieldy nature of their complexity. For instance, while backpropagation was established as reverse automatic differentiation in the 1970s [165], this DNN training technique did not become generally feasible until relatively recently. Thus, the rising dominance of DL in the 2010s [153, 237] is as much an outcome of big data infrastructure and hardware acceleration, specifically graphical processing units (GPUs), as it is the result of any one theoretical advance.

In contrast, the evolution of AutoML is harder to pin down, primarily because the scope of automating higher-level ML mechanisms can be made extremely broad. The extended history of this topic is grappled with elsewhere [140]. Nonetheless, the mainstream interpretation of AutoML – and even the abbreviation itself – has been forged on the back of advances in ML model/algorithm selection and the optimization of their user-defined hyperparameters. Accordingly, if the success of a DNN in the 2012 ImageNet competition [148] heralds the modern DL era, then the release of Auto-WEKA in 2013 [265] marks the start of the modern AutoML era. Within several years, by late 2016, these threads would start to entwine within the sub-field of neural architecture search (NAS) [13, 324]. This was not the first time that AutoML techniques had been applied to neural networks, but it was the moment that the broader data science community took notice. It was also opportune; the website Papers-With-Code [75] highlights that, while the number of DL publications has skyrocketed since 2012, year-by-year performance improvements on many benchmark datasets have diminished, i.e. those related to vision, text, audio, and speech. There is a sense that, as state-of-the-art (SoTA) DL models have become highly sophisticated, a reliance on human design is locking out broader engagement behind steep learning curves, while also hindering further metric progress. Automation through NAS is a vital step in enabling a broader community to push these technical limits.

Importantly, while NAS launches the modern AutoDL story, it does not encompass it. Model selection, i.e. the design of a neural network, is but one stage of a DL workflow. As illustrated in

Figure 1, there are many other subtasks involved in ML/DL, such as defining a problem of interest, collecting and organizing data, generating features, deploying and adapting trained models, and so on. This workflow may often be sequential in research and development, but real-world applications are much more agile and will typically reiterate through earlier operations and, in the case of large-scale systems, these processes may even be asynchronous. In effect, AI based on DL cannot reach its full potential without considering the entire life cycle of a solution, from its design to the maintenance phase.

We now bring attention to a previous review [140], which surveyed efforts to automate all aspects of this workflow in the general context of ML, with additional focus on how the resulting mechanisms may be integrated into a single architecture. The review touched on NAS and other elements of DL, but it could not cover the full extent of work in the AutoDL sphere. It did not need to; on a high level, working with DNNs fits smoothly into the conceptual framework of both AutoML and its extension, autonomous machine learning (AutonoML). However, on a practical level, the complexity of DNNs throws up many challenges that have arguably constrained the breadth of developments in AutoDL as compared to standard AutoML. Instead, what is remarkable is the ***depth*** of research in AutoDL, with numerous innovations brought about by attempts to surmount these obstacles, all with the aim of making the automation of DL feasible. Certainly, it would be remiss to trivialize AutoDL as just a subset of AutoML. Likewise, critically evaluating the limitations of present-day AutoDL is just as worthwhile as highlighting its accomplishments. For instance, the field of DL is sometimes criticized for a tunnel-vision focus on model-performance metrics within a limited set of benchmarks, an attitude which, while valid, risks missing the broader perspective on all that AutoDL may become [59, 91]. In essence, there is a need to consider several questions more thoroughly:

- As we enter the 2020s, what is the current research landscape of DL?
- What makes a "good" DL model?
- How can automated systems best pursue and support this model "goodness"?
- Is the field of AutoDL even advanced enough for such a meta-analysis?

This work is an extension of the broadly scoped AutonoML review [140] with an in-depth focus on the newly popularized topic of AutoDL. While there are many surveys in this sphere[1] [72, 80, 219, 282, 307], most focus on deep analysis within one or two sub-domains of AutoDL. In contrast, we examine research along the entirety of DL workflow – if it exists – and try to assess, as of 2021, what the present role of AutoDL is and where its evolution is leading. We first provide an overview of AutoDL in Section 2, introducing several fundamental concepts. Then, partitioning major AutoDL research into sections inspired by a DL workflow, as per Figure 2, we explore automation for: task management (Section 3), data preparation (Section 4), neural architecture design (Section 5), hyperparameter selection (Section 6), model deployment (Section 7), and online maintenance (Section 8).

Crucially, a major component of this review is a reaction to the sheer quantity of publications in the space of AutoDL; we aim to provide summary assessments of surveyed AutoDL algorithms/research in terms of ten carefully designed criteria, not just accuracy alone. These are introduced in Section 2.4 and form an evaluative framework for overviews within every aforementioned section, as well as, in Section 9, a final critical discussion around the entire field of AutoDL.

---

[1]One of the authors maintains a publicly accessible curated list of AutoDL resources at: https://github.com/D-X-Y/Awesome-AutoDL
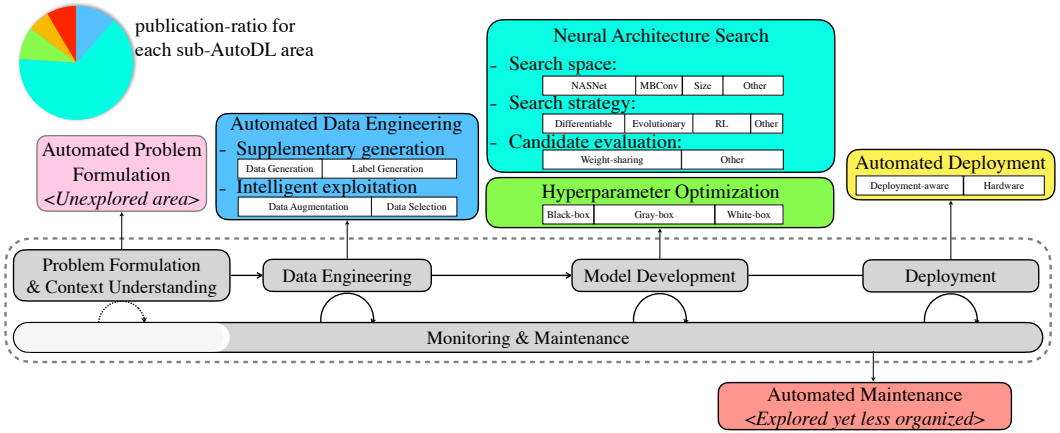
Fig. 2. Breakdown schematic for research activity in AutoDL, with surveyed publications attributed to different phases of a DL workflow and then further subcategorized. The pie chart denotes the ratio of publications across all workflow phases, while the white stacked bars denote ratios within each subcategory. All statistics are derived from the Awesome-AutoDL project, at: https://github.com/D-X-Y/Awesome-AutoDL

2 AUTODL: AN OVERVIEW

The aim of AutoDL is to support, if not outright replace, the manual operations that data scientists undertake when applying DL to a problem. Section 2.1 elaborates what such a DL workflow may entail. Of course, with AutoML having faced these same challenges for simpler ML models/algorithms, there is plenty of overlap between AutoDL and its more generic predecessor. Thus, the basic concepts of AutoML are introduced in Section 2.2, with a particular focus on an "ML pipeline". Many of these notions are almost directly transferable. Even so, the complexity of deep neural structures forces new challenges and different priorities upon AutoDL, which distinguish it as a research topic of its own. These are summarized in Section 2.3 and motivate the sections beyond. Finally, Section 2.4 systematizes the ten criteria by which we propose AutoDL research should be judged; these will underpin brief evaluative overviews presented throughout this monograph.

2.1 The DL Workflow

A DL task typically starts with defining a problem of interest. This primarily involves translating a desire conceived by humans into a computer-operable representation, e.g., the search for a predictive function from pixel maps to categorical classes, where sparse labeling may require semi-supervised learning techniques. Once the problem is defined, the next step is usually to manage the input space for a prospective DNN model. With a general assumption that input data should be independent and identically distributed (i.i.d.), strategies for data collection and organization need to be carefully considered. Neural networks also train better when raw data is intelligently preprocessed, and there are many ways this can be done. For example, principal component analysis (PCA) can be used to change the basis of high-dimensional data, i.e. instances with many features, such that data variances are maximized along a minimal number of dimensions; a subsequent projection eliminates the axes beyond these so-called principal components, achieving a dimensionality reduction with minimal information loss [117, 202]. Preprocessing can also include the encoding of categorical data as integers or one-hot vectors [4], as well as feature scaling via normalization, standardization, or power transformation [27].

Eventually, the time comes to construct a DL model. In standard formulation, this model is a DNN with multiple layers of neurons and various ways of connecting these layers together, e.g., using full connectivity, local convolutional connectivity, or even the outright layer skips of a residual neural network (ResNet) [106]. Accordingly, a selection of a neural architecture and training algorithm is required; these are associated with what are commonly called "model/architecture hyperparameters" and "algorithm/training hyperparameters", respectively. The DL model is then trained on input data, whereby the weights of this DL model are tuned[2] to best represent a desirable function. Historically, there have been many proposals for how to do this, ranging from GMDH [125] to unsupervised winner-takes-all methods [35, 85, 151]. However, backpropagation[3] has been the dominant training strategy of modern times for fully connected multi-layer neural networks. With initial derivations and implementations [165] tracing back to the 1960s and 1970s, respectively, it was popularized in the 1980s [152, 225, 281] and effectively used for training multi-layer perceptrons (MLPs), even though it would be decades before advances in hardware would enable ubiquitous usage on large-scale problems. Notably, the vanilla form of gradient descent through backpropagation has some drawbacks regarding speed, convergence, generalization, and so on. Numerous upgrades have been proposed over the decades, such as stochastic gradient descent (SGD), momentum SGD [225], resilient propagation [221], and adaptive estimation [141]. Neural Architecture Search (NAS) has also become one of the key research topics in DL model construction; see Section 5.

Once the DL model is selected and trained, there is yet more to do within a typical DL workflow. In practical applications, a DL model needs to be deployed, sometimes on custom devices and hardware. For example, MobileNet-V2 [232] is deployed on an Edge Tensor Processing Unit (Edge TPU) [96] to enable a 400 frames-per-second (FPS) inference speed, CycleGAN [318] targets Nvidia Graphics Processing Units (GPUs) for efficient execution, and face-recognition algorithms based on DL have been successfully deployed on smartphones [199]. In most cases, the deployed DL model is identical to the trained one in both structure and network weights. However, in some resource-constrained scenarios, the models must be compressed via pruning, quantization, or sparsity regularization [61, 101, 102, 167], before feeding them into real production. Furthermore, there is a problem of learning from and adapting to changing environments with continuously streaming, non-stationary data that must be addressed by AutoDL approaches. While this problem has been researched in the broader field of ML for a number of years [87, 136, 320], even starting to be considered and addressed in the context of fully automated and autonomous ML systems [140], it continues to be a major challenge for DL. As the problem arises in many real-world scenarios such as stock markets [8, 300] and consumer recommendation systems [107, 220], where DNNs are routinely deployed, robust adaptive capabilities are required to keep these models up-to-date.

Evidently, the success of a DL solution hinges on much more than model selection and the design of good neural architecture. That stated, many phases of the DL workflow can still be expressed in similar ways, i.e., they can often be re-framed as an optimization problem. Table 1 dissects numerous seminal works in AutoDL according to such an interpretation, and these commonalities will be expanded upon over the course of this review. Certainly, this representation is convenient, as a unified perspective across the entire DL workflow makes it arguably easier to build prescribed automated frameworks that manage a DL problem from end to end. Moreover, this means that there is always a baseline way to assess mechanized approaches at almost any phase of the workflow. Specifically, one can ask: what is the efficacy/speed of the search process? Does it maximize DL model performance? Of course, whether this is a sufficient form of evaluation is another matter,

---

[2]We avoid using the word "optimize" for network weights in this manuscript, so as to avoid confusion with the optimization of architectures, hardware, etc.
[3]Technically, the term 'backpropagation' refers to the computation of an error gradient, but it is often used loosely to also include a gradient-based optimization method that acts on this value.

Table 1. AutoDL algorithms dissected as optimization strategies.

	Algorithm	Search Space	Search Strategy	Boosts for Candidate Evaluation	Application
Auto. Data Engineering	[180] 2015.02	training data	Differential	optimal storage of discarded entropy	CLS: MNIST
	[78] 2018.05	training data selection	REINFORCE	weight sharing; mini-batch sample	CLS: MNIST/CIFAR/IMDB
	[218] 2018.03	training data re-weight	Differential	weight sharing; online approximation	CLS: MNIST/CIFAR
	[50] 2018.05	transformation in PIL [1]	PPO	smaller model; reduced dataset	CLS: five datasets
	[163] 2019.05	transformation in PIL	BayesOpt	weight sharing	CLS: four datasets
	[323] 2019.06	transformation in PIL	PPO	reduced dataset	Object DET: VOC/COCO
	[196] 2019.09	classical NLP augmentations	REINFORCE	fewer epochs	Dialogue tasks
	[51] 2019.09	transformation in PIL	Random	smaller model; reduced dataset	CLS + Object DET
	[256] 2019.12	training data	Differential	weight normalization	CLS + Game: CartPole
	[161] 2020.03	transformation in PIL	Differential	reduced dataset	CLS + Object DET
Neural Architecture Search	[17] 2009.09	cell topology in LSTM	Evolution	N/A	Grammar benchmarks
	[135] 2015.07	topology and operation in LSTM	Evolution	easy-to-hard tasks to filter	Music + Language
	[324] 2016.11	filter size + connectivity	PPO	fewer epochs	CLS + Language tasks
	[13] 2016.11	MetaQNN space	Q-learning	fewer epochs; early stop	CLS: four datasets
	[217] 2017.03	unrestricted CNN space	Evolution	weight inheritance	CLS: CIFAR-10/100
	[325] 2017.07	NASNet space	PPO	fewer epochs	CLS: five datasets
	[29] 2017.08	SMASH space	Random	weight generation via HyperNet [100]	CLS: five datasets
	[314] 2017.08	BlockQNN space	Q-learning	early stop	CLS: CIFAR/ImageNet
	[212] 2017.10	activation functions	PPO	smaller model	CLS + Translation
	[215] 2018.02	NASNet space	Evolution	smaller model; fewer epochs	CLS: CIFAR/ImageNet
	[206] 2018.02	RNS	REINFORCE	weight sharing [206]	CLS + Language tasks
	[171] 2018.06	RNS	Differential	weight sharing; smaller model; etc	CLS + Language tasks
	[33] 2018.06	tree-structure	REINFORCE	Net2Net [41]	CLS: CIFAR/ImageNet
	[263] 2018.07	MBS	PPO	fewer epochs	CLS + object DET
	[19] 2018.07	modified NASNet space	Random	weight sharing	CLS: CIFAR/ImageNet
	[177] 2018.08	RNS	Differential	weight sharing; neural predictor	CLS + language
	[34] 2018.12	MBConv-based space	Differential	weight sharing	CLS: CIFAR/ImageNet
	[168] 2019.01	RNS + connectivity	Differential	weight sharing; smaller model	SEG: three datasets
	[45] 2019.03	ShuffleNetv2-based backbone	Evolution	weight sharing	CLS + object DET
	[287] 2019.04	architecture generator space	manual	fewer epochs	CLS + object DET
	[92] 2019.04	FPN space	PPO	smaller model; fewer epochs	object DET: COCO
	[61] 2019.05	depth + width	Differential	weight sharing	CLS: CIFAR/ImageNet
	[79] 2019.06	densely connected search space	Differential	weight sharing	CLS + object DET
	[224] 2020.04	architecture generator space	BayesOpt	fewer epochs	CLS: six datasets
	[170] 2020.04	normalization+activation	Evolution	smaller dataset; fewer epochs	CLS + SEG + GAN
	[270] 2020.04	width + resolution	Differential	weight sharing	CLS: ImageNet
Hyperparameter Opt.	[21] 1999.09	a few differentiable HPs	Differential	N/A	Synthetic data
	[250] 2012.06	a few HPs	BayesOpt	modeling costs; parallel	Diverse tasks
	[180] 2015.02	hundreds of differentiable HPs	Differential	optimal storage of discarded entropy	CLS: MNIST + Omniglot
	[159] 2016.03	hundreds of HPs	Random (Bandit)	adaptive resource allocation	Diverse tasks
	[175] 2016.04	tens of HPs	CMA-ES [104]	limited time budget; parallel	CLS: MNIST
	[77] 2018.07	tens of HPs	BayesOpt	adaptive resource control; parallel	Diverse tasks
	[118] 2018.02	RL loss	Evolution	truncated trajectory; parallel	Physics: MuJoCo
	[174] 2019.11	millions of differentiable HPs	Differential	Neumann series	Diverse problems
	[60] 2020.06	MBS + tens of HPs	REINFORCE	weight sharing	CLS: six datasets
	[12] 2020.10	LR + WD	Differential	truncated trajectory	Few-shot CLS
Auto. Deployment	[192] 2017.06	device placement	REINFORCE	distributed training	CLS + translation
	[214] 2017.07	CMOS-based space + Arch. + HP	BayesOpt	N/A	CLS: MNIST
	[194] 2018.10	FPGA space + Arch. + HP	BayesOpt	N/A	N/A
	[200] 2019.06	PUMA space [10] + Arch. + HP	BayesOpt	N/A	CLS: Flower17/CIFAR
	[132] 2019.07	FPGA space + MBS	PPO	fewer epochs; multi-level exploration	CLS: CIFAR/ImageNet
	[47] 2019.06	Eyeriss space [46] + MBS	Differential	weight sharing	CLS: CIFAR/ImageNet
	[296] 2020.02	ASIC space + Arch.	REINFORCE	N/A	CLS: three datasets
	[316] 2021.02	edge accelerator space + MBS	PPO	weight sharing; neural predictor	CLS: ImageNet + SEG
Auto. Maintenance	[257] 1992.07	learning rate	Differential	N/A	Synthetic data
	[113] 2001.09	learning algorithm	Differential	N/A	Synthetic data
	[280] 2016.07	trace hyperparameter	Differential	greedy strategy; approximation	Synthetic: Ringworld
	[272] 2016.11	RL algorithm	RL	limit maximum trials	Markov decision tasks
	[82] 2017.03	initialization	Differential	truncated trajectory	Few-shot CLS
	[127] 2017.11	HPs	Evolution	parallelization	Game: Atari + StarCraft-II
	[293] 2018.05	HPs in return function	Differential	gradient approximation	Game: Atari
	[268] 2019.09	auxiliary task as questions	Differential	truncated trajectory	Synthetic + Game: Atari
	[66] 2019.06	initialization + LR/WD + stop steps	REINFORCE	limit maximum steps	Recommendation
	[143] 2019.10	RL loss scalar	Differential	truncated trajectory; parallelization	Game + Physics
	[292] 2020.07	RL target scalar	Differential	truncated trajectory; single agent	Game: Atari

Algorithms are sorted chronologically per phase of a DL workflow. Abbreviations are: "Auto." for "Automated", "Opt." for "Optimization", "RNS" for "reduced NASNet space", "MBS" for "MBConv-based space", "Arch." for "architectural search space", "HP" for "hyperparameters", "WD" for "weight decay", "CLS" for "classification", "DET" for "detection", and "SEG" for "segmentation".

and we begin such discussion in Section 2.4, but the optimization representation nonetheless serves to contextualize the historical evolution of AutoML/AutoDL.

2.2 Connections to AutoML

The automation of ML aims to minimize the need for human input at all stages of an ML workflow. There are numerous motivations for this endeavor, both social and technical. On the human side, expert technicians can turn their attention and energy to other productive areas, while democratization allows ML techniques to be employed by lay users. However, on the technological side, automation can also improve the efficiency and speed of finding ML solutions, the quality and consistency of those ML solutions, the reusability of ML methodologies, and so on. In pragmatic terms, this involves engineering a high-level AutoML system capable of managing the lower-level processes within an ML workflow, as schematized within Figure 2 for the DL case.

Now, terminology is key here. We define an ML/DL workflow as the steps that a user must traditionally be involved in to produce and maintain a performant ML/DL solution; see Section 2.1. This is distinct from what we introduce as an ML/DL pipeline, which describes the series of operations that inflow data undergoes in order to be transformed into useful outputs for descriptive, predictive, and prescriptive analytics. In the ML sphere, such a pipeline can consist of data imputers, feature engineers, predictors, and ensemblers, among other options [230, 231]. It follows that the overarching purpose of an ML workflow is to find the best ML pipeline, as judged by some performance metrics. Consequently, a portion of a modern AutoML system is always dedicated to what is effectively a direct optimization problem, i.e., selecting and tuning the components of such a pipeline. However, AutoML can also encompass functionality that supports this goal indirectly, e.g., a natural-language user interface (UI) or a meta-model designed to analyze dataset similarity. Thus, it is extremely challenging to distill both AutoML and its dynamical extension, AutonoML, into a simple engineering principle or a generic ML model development architecture [136, 140].

Here, due to the constrained focus of present-day AutoDL, only the basics of AutoML need to be introduced. Crucially, the core of every AutoML system is a module for hyperparameter optimization (HPO). Its job is to explore a search space of configurations that define, for example, a predictor. Each iteration is usually trained and then evaluated for model quality, e.g. classification accuracy or detection precision, so as to find an optimal variant of the predictor. If a user wants to then try other types of predictors and associated training algorithms, this becomes the combined algorithm selection and hyperparameter optimization (CASH) problem [265]; it is usually handled by treating the ML model/algorithm as just another high-level hyperparameter in configuration space. Beyond this, the next level up in model complexity is a full multi-component pipeline search, which the CASH solvers of many SoTA AutoML packages strive to manage.

Analogizing this search space within AutoDL is not trivial. The early AutoDL community – prior to the abbreviation being established – have not always been aware of AutoML research. When the term "NAS" was introduced [13, 324], the focus on neural architecture resembled a multi-component pipeline search with half the HPO, i.e., an HPO without algorithm hyperparameters. In fact, inconsistently with AutoML, AutoDL researchers can often refer to HPO as the strict complement of NAS, i.e., solely an optimization of a training algorithm [60, 80]. We will adopt this terminology and emphasize these distinctions when needed for clarity. Nonetheless, the analogy holds. Much like a multi-component CASH-solver, a NAS strategy will often have a pool of possible components, i.e., layers, to build a DNN from. The final layer of such an architecture is equivalent to a simple ML predictor, while all other layers can be seen as feature-space transformations. Of course, that is not to say that DL does all its feature engineering within a DNN; the term "auto-augmentation" usually refers to optimizing data preprocessors outside of a neural network [50, 51].
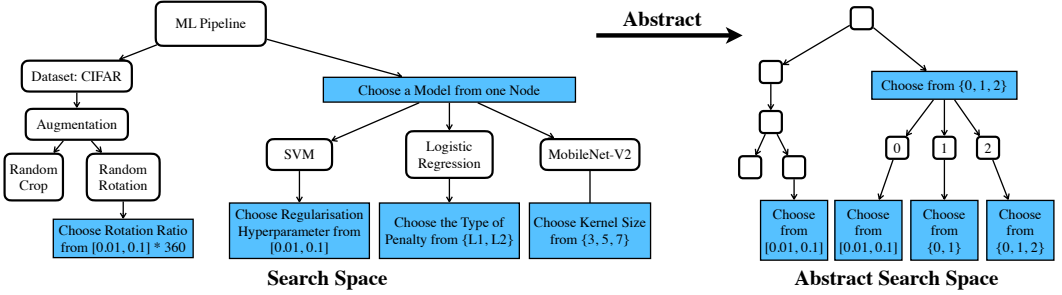
Fig. 3. An illustrative representation of an ML/DL pipeline search space, both concrete and abstract. White rounded-rectangle nodes denote fixed components within the search space, while blue rectangle nodes denote optimizable options. After an AutoML/AutoDL search, the resulting pipeline will have three components: a fixed crop, an optimized rotation, and one of three predictors with optimized hyperparameters.

Given the importance of a search space to ML/DL pipeline optimization, we provide a simple example of how configurations can be expressed. First, a few diverse examples of hyperparameters are provided:

- **Case$_1$**: Learning rate, as sampled from the range of $[0.01, 0.1]$ for each 90-epoch training run of ResNet.
- **Case$_2$**: The regularization variable for a Support-Vector Machine (SVM), possibly sampled from the range of $[0.01, 1.0]$.
- **Case$_3$**: The penalty for a logistic regression model, selected from {L1, L2}, as considered when maximizing classification accuracy for the Canadian Institute for Advanced Research (CIFAR) dataset [147].
- **Case$_4$**: Maximum rotation degree for a random rotation augmentation policy, as sampled from $[0.01, 0.1] \times 360^o$ for each 90-epoch training run of MobileNet-V2 on CIFAR.
- **Case$_5$**: Kernel size, selected from {3, 5, 7}, when training MobileNet-V2 variants.

From this, it is clear that hyperparameters can be continuous (Case$_1$, Case$_2$ and Case$_4$), categorical (Case$_3$), discrete (Case$_5$), and so on. Some will even be conditional, with, for instance, the choice of a predictor determining which variables are available to optimize. Combining these all together can thus produce a very complex and high-dimensional configuration space for ML/DL pipelines. This is exemplified in Figure 3, which depicts the search space covered by four of the above cases in both explicit and abstract form [204]. Notably, these configurations still only represent a pipeline of no more than three components, i.e. an image crop, an image rotation, and a monolithic model.

Once a configuration space is defined, there are many ways to search through it. Most solvers of the HPO/CASH/pipeline problem are black-box optimizers. The simplest ones are based on grid search or random search, the former being quite standard for manual HPO among ML practitioners. These strategies make no assumptions about the mapping from configuration space to solution quality, e.g., its derivatives, and are easy to scale up. However, the evaluation of each candidate solution can be computationally expensive on its own; traversing a large configuration space can be extremely costly. Thus, in practice – although random search has proved remarkably efficient [25] – a principled search strategy is desirable.

Broadly stated there are three types of optimization routines typically employed in AutoML/AutoDL:

- Population-based [93, 175, 217, 229] – These approaches operate on sets of configurations named populations. Many seek gradual improvements through genetic-based processes between individuals within the population, e.g. crossover, mutation and selection.
- Bayesian Optimization (BayesOpt) [77, 159, 193] – These approaches use a probabilistic approximation, also known as a surrogate, for the mapping between configuration space and ML pipeline quality. They alternate between two steps, the first being the use of an acquisition function on the surrogate to select the next most promising configuration to evaluate. The second step is evaluating an ML pipeline with that configuration and then using the new knowledge to update the fit of the surrogate.
- Distribution-based [171, 174, 180, 239, 258, 281] – These approaches learn a parametric model of the probability distribution for whether an ML pipeline candidate will have a high metric score. The parametric models are usually tuned by reinforcement learning (RL) [239, 258, 281] or gradient methods [171, 174, 180].

These three kinds of approaches can be further mixed into hybrid search strategies. Nonetheless, even with SoTA optimization routines, the cost of ML/DL pipeline optimization can remain extreme. There is plenty of ongoing research focused on efficiency gains within both ML and its narrower DL subset. One way to boost search speed is to rely on low-fidelity approximations for pipeline evaluations, such as via dataset subsampling for training/testing or early-stopping for training algorithms. However, there are many other proposed options as well.

Crucially, these are just the basics of AutoML; model selection can be tweaked in various ways. For instance, there are many investigations into the idea of meta-learning [22, 233, 234, 236], where the historical application of ML workflows to ML problems may boost the efficiency and quality of a current solution search. Then there is multi-objective optimization [123, 182], which appreciates the fact that model validity metrics, e.g., classification accuracy or detection precision, are not solely responsible for a good model. Some alternative requirements, like short runtimes, can be aggregated with model accuracy easily [34, 263], but others may be more challenging to evaluate, let alone Pareto-optimize, such as model interpretability or the convenience of user interactivity.

The field of AutoML has also just started exploring the idea of dynamic environments in earnest, i.e., inflow data that changes over time to represent different information. Unsurprisingly, the desire for an AutoML system to respond autonomously has produced developments in managing multi-pipeline solutions and adapting models. We refer the reader to the AutonoML review for further information [140]; it is an expansive subject. Here, we will only elaborate topics if existing AutoDL research warrants it.

2.3 AutoDL Beyond AutoML

As this review will make clear, while boosting search efficiency is important in the field of AutoML, it is critical to AutoDL. A DL model in the form of a DNN, combined with auto-augmentation, is far more flexible and complex than a typical ML pipeline. This means that DL pushes the limits of computational resource usage, hardware provisioning, model search space, and so on.

Given that model construction is so challenging, research in AutoDL has generally focused on a much narrower scope than AutoML. The bulk of existing surveys assess NAS [72, 219, 282] and HPO [80, 307]. Similarly, we have examined a spread of AutoDL approaches for model search, as dissected in Table 1; these will be discussed from various angles in the following sections.

However, AutoDL research has its own notable fringes, occasionally expanding beyond even the current domain of AutoML. For example, standard AutoML cares moderately about computational resources, whereas AutoDL *needs* optimal infrastructures to run SoTA DL models effectively. Thus, out of necessity, hardware search has become an attractive facet of AutoDL, with certain

experiments varying these infrastructures [192, 194] while keeping other elements constant, e.g., data preparation, the DL model, and the training strategy. Then there is DL pipeline ensembling; the analog is not unheard of in AutoML/AutonoML [140], but its appeal has arguably driven its development in AutoDL much more strongly. After all, if a single DL model is so computationally expensive to construct, would it not be beneficial to keep the completed product around in a pool or ensemble, so as to leverage whatever lessons it has learned? With benefits in robustness, reusability, and generalizability, this approach has been adopted several times [32, 36, 210, 287, 308].

The take-away here is that AutoDL is worthy of independent consideration alongside AutoML; the field has had to embrace various innovations to face the challenges of DL model complexity.

2.4   Assessment Criteria for AutoDL Research

Here lies a central problem: to varying degrees along the workflow, the field of AutoDL has been flooded with research. It is extremely challenging for a would-be developer of an integrated AutoDL system to decide which techniques and mechanisms to favor. In fact, we argue that the predominant focus on end-point accuracy/efficiency is insufficient to assess a piece of AutoDL research, even after accounting for the shortcomings of current benchmarking practices. Thus, this review uniquely proposes that AutoDL researchers/developers should pay attention to a more encompassing set of ten criteria.

Altogether, the ten are listed below.

I. Novelty: How does the AutoDL algorithm distinguish itself from all existing works in AutoDL?

II. Solution Quality: How well does the AutoDL algorithm minimize the error of a target DL model?

III. Efficiency: Does the AutoDL algorithm achieve its aims with minimal resource expenditure, especially in terms of time costs? How does it impact the resource costs of a target DL model?

IV. Stability: How consistent is the performance of the AutoDL algorithm with respect to statistical variability? How dependent is its performance on the choice of settings?

V. Interpretability: Is the AutoDL algorithm theoretically sound and human-understandable? How does it impact the explainability of a target DL model?

VI. Reproducibility: Are reported results associated with the AutoDL algorithm easily reproduced? Have they been reproduced?

VII. Engineering Quality: Does the AutoDL algorithm have an implementation? Is this codebase well managed, documented, accessible, and of a high standard?

VIII. Scalability: Is it feasible for the AutoDL algorithm to scale to a larger model or more data?

IX. Generalizability: Can the AutoDL algorithm be applied to different tasks, datasets, search spaces, etc.?

X. Eco-friendliness: What is the environmental impact of both the AutoDL algorithm and its target DL model?

These criteria will form the basis for evaluative assessments in the sections to come. However, although Section 9.1 discusses relevant questions to ask of individual publications, motivating a detailed breakdown, most of the overviews in this monograph concern entire threads of AutoDL research related to individual phases of a DL workflow. Thus, there is more to explain about how we formulate such assessments, which are necessarily aggregated and mostly qualitative.

First of all, there is an important caveat to acknowledge. An AutoDL algorithm, i.e., a process, is distinct from an impacted target DL model, i.e., an outcome. Some AutoDL algorithms will directly construct this model, e.g., NAS mechanisms, and others will simply influence it, e.g., automated mechanisms for data engineering or maintenance. Now, an issue arises in that, while the notion of accuracy for an AutoDL algorithm is derived and will always refer to the performance of a target DL model, both an AutoDL algorithm and a target DL model will have their own distinct qualities

in terms of efficiency, interpretability, etc. The two need not be aligned either; an efficient NAS approach may produce an inefficient DNN, and, likewise, an inefficient mechanism can produce an efficient model. However, in practice, this survey has not found such discrepancies to be particularly common. If an ethos of efficiency or interpretability drives research in this field, as an example, then both AutoDL algorithm and target DL model tend to benefit from those improvements. Thus, from an aggregate perspective, it is safe to make research-trend assessments based on evaluating the AutoDL process itself.

With that issue addressed, we now elaborate on the upcoming overviews. To begin with, there is one major divergence for a criterion-based assessment between publications and entire research trends. Specifically, the novelty (I) of a methodological category is represented by the years in which seminal works for the approach were published and, accordingly, how long the computer science community has been aware of its existence. Distinct from novelty for an individual publication, which should always be significant, novelty for an entire topic is meant to be more informative than judgemental; older approaches are likely to be more robust and well-explored, while newer approaches are more likely to leverage SoTA breakthroughs. Also, as a side note, it is very case-dependent on whether it is more instructional to present these histories in the context of DL or, more broadly, ML. Associated table captions will clarify which historical context is used.

In all other criteria, assessments for a methodological paradigm are but averages of all surveyed publications that theoretically/experimentally employ that paradigm. Most (II–V & VIII–X) are given a rating of low, medium, and high. Designations of mixed and unknown, i.e., "?", are also possible.

Solution quality (II) represents the contribution of an AutoDL approach towards the validity of a resulting DL model, according to self-reported but peer-reviewed claims. For model-development techniques, this is baseline accuracy (suppose it is a classification task). For maintenance techniques, this is accuracy integrated over time. For everything else, this is improvement beyond the baseline accuracy. Next up is efficiency (III), which, with adjustments for memory usage, primarily considers how quickly an AutoDL algorithm runs, again as self-reported. For model-development techniques, this includes the training time for the DL model as well. Stability (IV) then follows, acknowledging how tight self-reported statistical bounds are on the accuracy of an AutoDL approach, as appropriately defined with respect to a phase of the DL workflow.

Interpretability (V) notes the degree to which either, one, the operation/impact of an AutoDL algorithm is immediately clear or, two, publications associated with the research trend make the effort to theoretically elucidate how and why the algorithm works; ablation studies are an example of a gold standard for this criterion. Scalability (VIII) then assesses how resilient the efficiency of an AutoDL approach is when handling an increasingly difficult DL problem, as evidenced by self-reported complexity analyses or similar extrapolations. Naturally, dataset size is a baseline metric for problem difficulty at all phases of a DL workflow, but model-development and deployment methodologies are also considered with respect to the size of relevant search spaces, and deployment and maintenance approaches additionally deal with model size. As for generalizability (IX), this criterion captures the diversity of DL problems that an AutoDL approach can be applied to as is. This is often calculated as the inverse of how many context-specific assumptions are present in the self-reported theoretical foundations of the methodology.

Finally, we turn to the remaining criteria. Notably, the ones already listed can be compiled directly from researchers reporting their own findings, albeit in an aggregate sense. In contrast, reproducibility (VI) and engineering quality (VII) rely on secondary benchmarks and implementations, respectively. It is beyond the scope of this review to grade these with a desirable rigor, so the associated criteria are simply marked "✓" if there is enough literature to assess relevant research trends appropriately, and "?" if not. As for eco-friendliness (X), this criterion is stimulated

by growing environmental concerns and can be graded within the low-to-high scheme but, in practice, will often be marked as unknown in this review. One can assert that a ranking for an AutoDL algorithm is likely to correlate with efficiency and scalability, but, except in the most obvious cases, this review requires a direct analysis of power consumption to support a qualitative assessment.

With these ten criteria defined and their evaluation explained, we emphasise that such a set provides a broader assessment framework beyond the useful but limited representation of AutoDL algorithms in Table 1. Indeed, this kind of consistency and thoroughness is needed to compensate for the idiosyncrasies at every phase of the DL workflow, and we posit that such an evaluative framework is a prerequisite to truly identifying promising directions in AutoDL.

3   AUTOMATED PROBLEM FORMULATION

Employing the DL approach for real-world applications spans a wide range of processes, shown in Figure 1. If an ideal AI system is to one day automate this entire procedure, then, for completeness, it is worth discussing the formulation of a learning task from a problem context. Put simply, a problem context conceptualizes broad human-defined goals, such as creating "an undefeatable computer opponent for the game of Go" or "a car that automatically drives people to their desired destination". It also covers the environment in which these goals apply, such as "the rules of Go" or "the geography and physics of road transportation". Traditionally, data scientists have had to manually translate these conceptual contexts into computer-actionable tasks. For instance, one may decide to frame the design of a Go agent as an RL-based optimization task for a DNN, where the probability of winning is the objective function and an appropriately constrained input space represents board positions [245].

This ability to interpret a general problem context and forge a pragmatic pathway to a DL solution is a challenge; it may ultimately be the final obstacle for pure AutoDL, given how difficult it is to artificially mimic human creativity. Unsurprisingly, there is no major literature on this topic currently, with the majority of existing work in both AutoML and AutoDL focusing more on model-construction aspects. For that same reason, it is difficult to speculate whether AutoDL would treat task auto-formulation differently from AutoML. Certainly, AutoDL opens up new types of learning tasks to map problems into, e.g., the development of convolutional long short-term memory (LSTM) networks for dynamic image recognition problems, but this is an issue of categorization rather than a fundamental contrast.

Nonetheless, speculation aside, this space is not untouched. One example is the Libra system[4], which aims to assist – if not automate – the act of declaring ML/DL tasks via natural language processing (NLP). It enables this by constructing a semantic context around datasets and other objects, making it possible to interpret requests such as "please model the median number of households" or "predict the proximity to the ocean". Likewise, the notion of problem-to-task translation links closely to scattered but growing research in the area of automated human-AI interfacing; interested readers are pointed to the section on user interactivity in the AutonoML review [140]. For now, however, progress in this area remains too sparse to evaluate in terms of the criteria introduced in Section 2.4.

4   AUTOMATED DATA ENGINEERING

In real-world applications, the path from a raw data source to a model input can be a long one. At the earliest extreme, raw data can be encoded in numerous ways, e.g., vector versus pixel graphics, and can possess context-based idiosyncrasies, e.g., asynchronous timestamps for datastreams. Given

---

[4]See: https://github.com/Palashio/libra

all the expert knowledge ingrained in these arbitrarily unique formats, data wrangling joins task formulation as a process that is intensely challenging to automate. Of course, eventually, truly autonomous learning agents will need to be capable of seeking out problem-relevant data, at least as well as a human trawling the internet. Perhaps these efforts will be aided by modern innovations around the "extract, transform, load" (ETL) paradigm, renowned in data engineering. However, to date, it is rare to find even AutoML-based research/technology that does not assume some level of convenient formatting for collated data.

Nonetheless, there have been reasonable investigative attempts at automating most other phases of data engineering. Traditional AutoML often works with relatively simple and quickly trained ML models, so it is arguably more advanced in this space; see the Automated Feature Engineering section of the AutonoML review [140]. Even so, AutoDL research has explored data preparation too, and its focus is driven by the unique demands of DL models. Simply put, a DNN needs a lot of labeled data[5]. Its complex nature as a universal approximator affords many representational benefits, but the numerous (effective) degrees-of-freedom requires a large number of training data instances to achieve good generalisation and avoid overfitting. This can make it difficult to properly train, for example, GPT-3 with its 175 billion weights [30]. Thus, most AutoDL research in this space prioritises one of two approaches: (i) generate more data or (ii) use data in a better way.

4.1 Supplementary Generation

When ground-truth observations are limited, one solution is to artificially generate new instances. Many sophisticated generators can be employed in this capacity, e.g., deep Generative Adversarial Networks (GANs) [95], deep Variational Autoencoders (VAEs) [269], or modern physics engines [73]. Abstractly put, the idea is to automatically "paint out" the broader space represented by limited real data via interpolative or – more riskily – extrapolative procedures. In practice, the assumptions underlying such estimations can be complex enough to seem arcane, e.g., the function learned by a GAN discriminator or the chaotic dynamics of complex physical equations. Nonetheless, the procedure has proven merit, provided that a data generator, e.g., in the form of a neural network [256], can, via optimization or similar, properly capture the salient characteristics of the real data.

This is key; automated data generation cannot supply a model with any useful information beyond its limited observations. It can actually introduce false assumptions that degrades model performance, made clear once the model is tested on the real data environment. Accordingly, to pursue a deeper understanding of this issue, several research efforts have examined just how reductively observed knowledge can be distilled [180, 275]. Indeed, this kind of compression is the premise behind autoencoding. But it remains an inescapable fact that, if a model requires new discriminative information, not just data estimates, an AutoDL system will need to seek it out. For instance, while the system can itself be designed to pick reliable pseudo-labels for unlabeled data, known as self-training, a common approach is to query an oracle – often a human – for such annotations. If these instances of data are selected intelligently, this is called active learning, and even the high-level strategies for active learning have been explored as targets for automated selection [145].

4.2 Intelligent Exploitation

Even if data is limited, the practicalities of ingesting data and training a DNN can still very much affect its performance. Batch size is a classic hyperparameter to consider, but even the order of

---

[5]Labels encompass manual annotations for supervised learning, pseudo-labels for unsupervised learning, and generated proxy-task labels for self-supervised learning.

Table 2. Evaluative assessment for trends in automated data engineering.

		Novelty	Solution	Effic.	Stability	Interp.	Reprod.	Engi.	Scalability	General.	Eco.
Supplementary Generation	data generation	≈ 2 (30)	Low	Low	Low	High	?	?	?	Low	?
	label generation	≈ 2 (40)	High	Low	High	Low	?	?	High	Low	?
Intelligent Exploitation	data augmentation	≈ 5 (30)	High	Mixed	High	Low	?	?	High	Low	?
	data selection	≈ 4 (30)	Medium	Mixed	High	High	?	?	High	High	?

Each row marks an emergent trend in AutoDL, specifically automated data engineering. Each column marks a criterion – see Section 2.4 – by which the trend is assessed. The evaluations are mostly qualitative, averaged across the most significant works researching the trend. Where a graded value is not provided, "?" indicates that a rigorous assessment is not achievable without more research works to analyze. Novelty denotes years since seminal works in DL (ML) were published. Abbreviations are: "Solution" for Solution Quality, "Effic." for Efficiency, "Interp." for Interpretability, "Reprod." for Reproducibility, "Engi." for Engineering Quality, "General." for Generalizability, and "Eco." for Eco-friendliness.

data ingestion matters [23]. In effect, research here revolves around information content within data and how to maximally squeeze out its beneficial impact on an ML model. So, as an example of automating data provision, a neural energy network has been explored as a teacher to select suitable data for another student network [64]. Elsewhere, a weighting function – represented later by an MLP [243] – has been used to flexibly control the influence of incoming data on model training; gradient-based optimization methods have attempted to learn the best values of these weights, so as to minimize model validation loss [218].

Of course, not all processes of automatically "getting useful data" can be divided so cleanly between generating new instances and using existing ones better. The complex data types used by DNNs, such as images and freeform texts, often contain extra information content that can be pulled out by relatively simple transformations, e.g., image rotations or by replacing text with synonyms. This concept is known as "augmentation" in the DL community, straddling the line between generative and transformative, and its use can significantly boost DL performance [50]. Many forms of augmentation have been explored, including normalization, standardization [124], factor design [300], image augmentation [242], text augmentation [178], etc. However, while most of these simple transformations are well defined, it can still be difficult to manually choose their hyperparameters. For trivial instance, without domain knowledge, when do image rotations go from a better discernment of a handwritten '6' to a misidentification of the digit '9'? Thus, a new thread of research in AutoDL named auto-augmentation aims to take such choices out of human hands. The original work explores image augmentation policies, seeking to optimize a mix of transformation types and magnitudes [50]. Subsequent works have tried to improve accuracy, efficiency, and generalization ability [51, 196].

4.3 Overview

In summary, the current state of automated data engineering in AutoDL – what little of it exists – is primarily about getting as much data as possible, i.e., data generation and label generation, and leveraging what exists with maximal efficiency, i.e., data augmentation and data selection. First to note though, from Table 2, is that many of the related approaches have been developed in the context of ML over many decades, even if they have only being associated with DL for a handful of years.

Traditionally, data engineering efforts have been guided with or without model performance in mind, and these are sometimes called "wrapper" or "filter" methods, respectively; filter methods, e.g., PCA, often lean on statistical properties of the datasets alone. Although filter methods are much faster than wrapper methods, especially given that training a DNN for validation is so

expensive, virtually all AutoDL endeavors in this area have thus far focused on model performance, i.e., they deal with wrapper methods. Unsurprisingly, this preoccupation challenges the efficiency of automated data engineering in general, although other factors can of course impact the rating either way. For instance, data generation often requires the expensive training of generator models, while, conversely, the computational demands for both data augmentation and data selection can be ameliorated somewhat by weight sharing – see Section 5.3 – and online approaches, respectively.

Nonetheless, computational cost aside, there is a reason these approaches are employed to begin with. The information content provided by both label generation and data augmentation has been showed to markedly improve model accuracy. Data generation and selection both have promise too, although the latter refines rather than enriches a dataset and thus strains against its own performance ceiling, i.e., it is limited by what data is available. To be fair, both have also only had limited exploration in AutoDL thus far, applied only to small datasets, which means there has not been the same degree of model-improvement claims in the literature.

In fact, data generation in AutoDL can be considered particularly experimental at the current time, suggesting that approaches do not have the same degree of performance stability as the other three trends of interest, even if, admittedly, auto-augmentation has only been primarily tested within the context of vision problems. Additionally, data generation sticks out in terms of scalability. Specifically, algorithms for the other three AutoDL trends typically scale reasonably well with respect to the size of a problem dataset, i.e., the yardstick of interest for these approaches, as they simply involve operations applied to instances of data. In contrast, it is unclear as to how an interpolating/extrapolating data generation algorithm depends on dataset size, given that some data instances paint out a far more informative picture of a global features-to-label space than others. In short, more study is required.

Turning to the remaining criteria in Table 2, the interpretability of data-engineering algorithms is roughly in opposition to their associated accuracy improvements. It is hard to speculate whether the anti-correlation is coincidental or not, but it is evident that the practices of data generation and data selection have been contemplated heavily in the literature, supported by visualization in the former case, while algorithms for label generation and data augmentation have had very little theoretical analysis to date. When it comes to whether these approaches generalize though, data selection is the sole winner, with procedures usually agnostic to data structure and format, problem formulation, etc. In contrast, the other three AutoDL trends often manifest in algorithmic implementations that are tied to certain tasks or applications. Finally, it is not possible to comment on reproducibility or engineering quality, as benchmarks and codebases are in short supply, while eco-friendliness has not yet been studied. This is to be expected, given that existing research into automated data engineering within DL still seems mostly proof-of-principle or focused on theoretical design.

For completeness, it is worth returning to how data engineering differs between ML and DL. While auto-augmentation in the DL context is often related to generating transformed copies of existing data, there is a fuzzy overlap with feature engineering, which typically refers to in-place data transformations. We do not stress the semantics here; auto-augmentation is nascent enough to be somewhat fluid. Nonetheless, we emphasize that AutoML has often rigidly separated the task of optimizing preprocessing pipelines, i.e., data engineering, from the CASH problem, i.e., model development. In the contrasting case of DL, while augmentation covers network-external preprocessing, much of the feature engineering is still relegated to the early layers of a DNN as part of a fuzzy overlap. It is thus understandable why the topic of NAS dominates AutoDL, even if the field will eventually need to grapple with data preparation more broadly, especially as AutoDL moves beyond academic research and into real-world application.

5   NEURAL ARCHITECTURE SEARCH

Flexible neural architecture design is arguably the core advantage/challenge offered up by DNNs. Thus, unsurprisingly, automating that design has attracted significant attention from the DL community. Fundamentally, this endeavor revolves around connecting layers of neurons into an extended architecture, where the full network can be considered as a directed acyclic graph (DAG). Examples in Figure 4 depict this formulation, with input tensors being transformed into output tensors via a number of intermediate operations. More precisely, intermediate feature tensors are produced via the transformation and fusion (with some learnable weights) of existing feature tensors. The final output, resulting from the last operation and in tensor format, represents useful information, e.g., classification predictions or discriminative features in supervised and unsupervised learning, respectively. Given this context, the modern surge in popularity of DL has been driven by proposals of new and effective forms for such architectures [105, 106, 148, 246, 262], advances that have significantly boosted the performance of numerous applications. However, recent years have shown clear diminishing returns; SoTA architectures are just too complex to invent by hand.

Now, the automation of neural architecture design is not new [17]. However, early works focused on shallow networks and did not show promising empirical results when compared with manual designs, especially on large-scale benchmarks. Then, in November of 2016, two concurrent works [13, 324] were publicly released, showing for the first time that automatically designed DNNs could be competitive with – if not better than – manually designed SoTA architectures. It was at that time that the term "Neural Architecture Search" was proposed [324], referring to AutoDL algorithms specifically engineered to search for neural architectures. Since then, NAS has been cemented as a core element of AutoDL.

In NAS, researchers mainly focus on three parts: (1) the search space, containing all the possible candidate neural architectures that can be chosen, (2) the search strategy, defining how to find a good candidate architecture from the search space, and (3) the evaluation of the candidate architecture, which generates a performance metric to guide the search strategy. Accordingly, these three aspects form the basis of comparison for 25+ NAS algorithms in Table 1, and the rest of this section reviews these concepts in greater depth.

5.1   Search Space

As shown in Figure 4, NAS in the context of DNNs is essentially a search through possible DAG topologies, with variations in both connectivity and transformative operators. The set of these possibilities is called a search space, and different NAS algorithms constrain this set in different ways according to both expert knowledge and domain characteristics.

Size-related Search Spaces: As stressed before, while NAS has been popularized within the last handful of years, this is not to imply that no previous work has ever tried to optimize the structure of a neural network [250, 265]. Most attempts, however, have attacked the problem in a relatively rudimentary manner, focused primarily on network depth and layer width for feedforward networks without substantially perturbing their topology. In more recent phases of the DL era, efforts to automatically control the size of modern architectures were initially associated with the concept of structured network pruning, complete with learnable pruning ratios and dynamic networks [58, 81]. These approaches would decide on, for instance, the number of channels to use per layer, as well as other depth values. Naturally, such work would eventually be recontextualized as part of NAS [32, 61, 84], yet, despite this unification, the research thread still has its own priorities, e.g., how to preserve model performance after pruning or how to adjust DNN weights as part of a dynamic inference procedure [103].
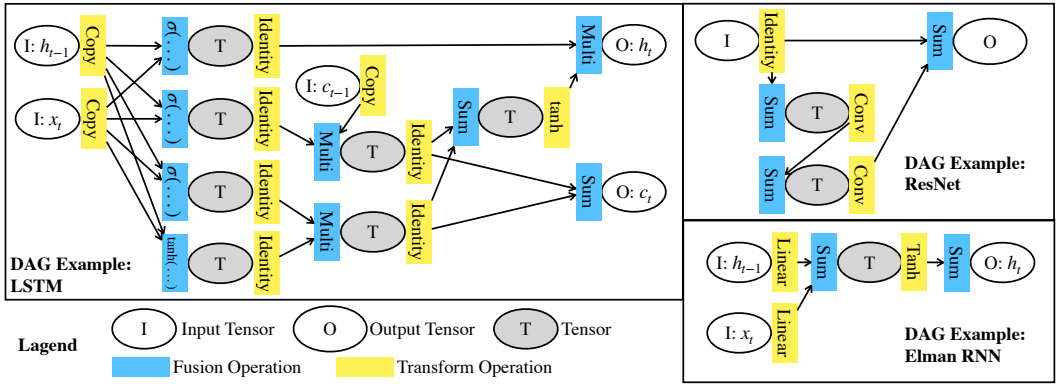
Fig. 4. Illustrative representations of three popular neural architectures in directed acyclic graph (DAG) format. In each DAG, a node (circle) represents a feature tensor and an edge (arrow) indicates tensor flow. The blue block ahead of each tensor denotes a fusion operation to combine incoming flows, while the yellow block behind each tensor denotes a transform operation applied to the outgoing flow.

Convolutional Search Spaces: The eponymous "NAS" work designed a search space for a convolution neural network (CNN) by allowing the kernel height, the kernel width, the number of kernels, and the connectivity between different layers to be searchable, while fixing the depth [324]. Concurrently, MetaQNN explored a different convolution search space [13]. The MetaQNN search space makes the depth searchable, allows the layer type to be selected from a {convolution, pooling, fully connected} set, and includes the hyperparameters associated with each layer type, such as the number of kernels for convolution. The automatically discovered architectures within these two search spaces achieved competitive results on CIFAR compared to the popular ResNet and DenseNet [120]. In fact the high performance-to-manual-effort ratios of these works can be credited with the initial pull of researchers to the field of NAS.

However, in the wake of these seminal NAS works [13, 324], which proved more accurate than several popular deep CNNs on tested benchmarks, it was quickly realized that architectural search space could rapidly explode beyond feasible use. A pragmatic philosophy arose, founded in the notion of transferability. In essence, it asked: can good architectures be built by stacking together reusable cells/blocks, each larger than a single layer? The NASNet algorithm was one of the first to use this approach, working with a cell-decomposable high-performance network tuned to CIFAR [325]. Specifically, it hypothesized that stacking more cells on this network would make it adept at dealing with the larger and higher-resolution ImageNet, while still leveraging the previously optimized sub-structure of the CIFAR-based architecture. Each type of reusable cell would be optimized by searching through a set of possible DAG topologies, similar to Figure 4, with 13 possible options of basic pre-defined transformations for the operators. Subsequent works have since improved the NASNet search space by removing useless operators [169, 171, 206] or relaxing topological constraints [65, 303]; we refer to these variants as NASNet-like search spaces.

The concept of reusability forces a dramatic reduction of a search space, although there remains some debate whether this constraint on DNN solutions is overly limiting [79]. For now, though, the consensus view is to continue minimizing computational costs. In fact, even NASNet-like search spaces have been considered too bloated, with resulting networks inducing too many flows from input to output tensors, negatively impacting inference speed. For instance, when comparing an architecture from a NASNet-like search space with ResNet, where the two have a similar count of

floating-point operations (FLOPs) and are trained on ImageNet, ResNet has a far superior GPU-based inference speed [34]. In an effort to counter this, the MnasNet algorithm has since been proposed with the aim of discovering DNNs for edge devices [263], such as mobiles, with a search space inspired by the mobile inverted bottleneck convolution (MBConv) used in MobileNet-V2 [232]. The MnasNet search space still carries over the cell-stacking notion of NASNet, allowing both the number of cells and convolution kernels to be optimized, while also searching through different transformation operators. However, the topology of each cell is fixed as an MBConv structure enhanced by squeeze-and-excitation (SE) principles. Other algorithms like ProxylessNAS [34] and FBNet [284] have likewise gone down the MBConv route, showing significantly improved inference latency over models produced via NASNet-like search spaces.

Naturally, not all works in this topic align with NASNet-like and MBConv-based approaches. Some have experimented with representation, e.g., exploring a tree-structured architecture space [33], while others have designed search spaces with certain outputs in mind, e.g., densely connected networks [79].

Other Search Spaces: The seminal work that introduced NAS in 2016 simultaneously presented results for both image-based CNNs and text-based recurrent neural networks (RNNs) [324]. Indeed, while RNNs have not been as heavily investigated as CNNs in the realm of NAS, possibly because training topologically complex RNNs to convergence is extremely challenging [63, 171], they do have a history. Efforts to optimize the topology of a memory cell in an LSTM network [112] have been made as early as in 2009 [17]. Arguably, the 2016 work keeps things relatively simple, effectively fixing cell topology beforehand and only searching through the type of operators involved in this cell. However, variations in topology would later be included in the search space [206]. Overall, NAS-discovered RNNs are shown to outperform the vanilla LSTM on some small benchmarks, but evidence is still lacking to compare against SoTA manual designs at the larger scale [171]. In addition, the complex topological structure of current NAS-generated RNNs makes it difficult to utilize the parallel-computation advantage of modern accelerators. As a result, the realistic training/inference speed for these architectures is unexpectedly slow [63, 171]; more research is required to make NAS a feasible approach for generating high-performance RNNs.

Of course, not all search spaces are designed purely with CNNs or RNNs in mind. For instance, researchers have developed a normalization-activation search space [170] where basic mathematical operations are employed in the architectural DAG, such as addition, multiplication, tanh, sigmoid, sqrt, etc. As another example, attention-based sequence-to-sequence models have recently been explored [251], drawing inspiration from both a NASNet-like search space and Transformer architectures [267]. In fact, given the popularity of the Transformer, we are currently witnessing a gradual shift of focus from CNN/RNN-based search spaces to Transformer-based search spaces [40, 252].

Improving Search Space Design: It is becoming apparent that different types of problem/network may benefit from NAS searching on different instances of a search space, at least in terms of options available for model topology and transformation operators. Accordingly, research attention has recently focused on seeking design principles for compact but encompassing search spaces, especially for particular classes of network [210], e.g., ResNet-like models. Other attempts to constrain search spaces continue, e.g., via the use of a network generator [287]. These generators produce a large but controllable set of DNN candidates to sift through, which is a different approach from grappling with a complex and expansive multi-dimensional search space. Taking this concept even further [204, 224, 287], researchers could potentially factorize a huge search space by a small number of generators, each covering their own subspace. Naturally, whether this is a good idea in practice remains to be seen; the NAS problem is then effectively elevated to one of "generator search". If generators only need to be built once – and once only – to optimally capture the space for certain classes of problem/network, then this may be an appealing design principle. However,

if this is not the case, then assessing a subdivision of NAS into two levels depends again on how compact but encompassing the effective search space becomes.

Ultimately, it is evident that the automation provided by NAS simply shunts manual design into other processes. For all its benefits, there is still a substantial reliance on human decisions to craft an effective search space, even if those choices are made at the developer level rather than by the user. Certainly, several of the works reviewed in this section have worked on automating search space design [210, 224, 287], but, in practice, hyperparameters need to be selected at whatever level they are shifted to. The only way to avoid making assumptions is if it is proven that certain search spaces are ideal for certain classes of network/problem, and this verges on the topic of meta-knowledge. That stated, one can – with respect to the architectural DAGs exemplified by Figure 4 – consider a human choice to be (1) a topological constraint [63, 171, 325] or (2) the inclusion of an advanced operator [171, 263]. Thus, full generalization appears to be a prerequisite for maximally automating the NAS process, i.e., loosening all constraints and selecting a very basic set of mathematical operators [170, 216]. However, even with modern computational resources, NAS without some degree of human search-space design is currently infeasible.

5.2 Search Strategy

Once a set of possible architectures is defined by a searchable and potentially complex space, it is up to a search strategy to explore this space efficiently and locate an optimal architecture. Given that each candidate network can be evaluated for performance, typically accuracy, any black-box optimization method can be used as a search strategy in NAS, such as RL, an evolutionary algorithm, BayesOpt, etc. Notably, because AutoDL as a field evolved organically to focus on neural architecture before optimizing hyperparameters more broadly, contrasting the developmental flow of general AutoML, Section 6 is a more appropriate place to discuss the details of optimization methods. Here, we mainly focus on how they are tailored for NAS.

RL-based NAS methods typically encode a neural architecture as a series of variables, which can be interpreted as building instructions for the model. For instance, these variables may index existing node inputs within a DAG, exemplified in Figure 4, as well as the types/configurations of operators to attach to the current network. It is then up to a component called a "controller" to select candidate encodings within this space and have the performance of their represented models tested, with good outcomes guiding the controller in its continued exploration.

Originally, the eponymous NAS algorithm utilized an LSTM network as this controller [324], thus ascribing a sequential nature to the encodings; each sequence would represent the way to progressively grow out a network. More recent efforts have instead simplified the controller to work with a collection of *independent* multinomial variables that represent distributions over the transformative operators available to a candidate DNN [20, 60]. So, whereas an LSTM is one predictor that progressively predicts an optimal encoding, the simplified controller can be considered as a set of parallel predictors, each one responsible for one variable in the encoding. Thus, the upgraded controller can immediately construct an architecture without building in sequence, but its effectiveness on broader search space still needs investigation.

Importantly, training the controller is orthogonal to both its design and that of the encodings it searches through. For instance, Proximal Policy Optimization (PPO) [239] is utilized for NASNet [324, 325], REINFORCE [281] is applied by both ENAS [206] and its subsequent works [20, 60], and Q-learning [276] is employed by MetaQNN [13]. To date, simple RL methods like REINFORCE appear sufficient for the purposes of NAS [20] in popular NASNet-like and MBConv-based settings, although the more challenging search spaces discussed in Section 5.1 may eventually require more advanced RL algorithms.

Evolutionary NAS methods deal with the same encoding issues – the definition of architectural genotype – that RL-based strategies do, but they otherwise employ standard procedures for evolving a population of networks into fit-for-purpose models. Most of the time, researchers only vary two specific aspects: the mutation strategy for an individual architecture and the evolution process for the whole population [135, 215, 217, 251, 252]. Given the DAG representation in Figure 4, mutation typically involves adding/removing edges and nodes, changing transformation operators, or merging two graphs. Evolution processes can be more varied, although pairwise competition, a form of tournament selection [93], has currently received a lot of attention [217]. A couple of works have employed this mechanism, although with an additional age-based mechanism to prioritize younger individuals, i.e., candidate networks more recently added to a population [215, 251].

A fairly common issue with applying evolution to NAS and other AutoDL sub-areas is the isomorphism of architectures. Two candidate networks with different DAG representations may be mathematically identical to each other. For example, the NATS-Bench topology search space [59, 65] has 15K unique DAG encodings, yet there are only 6.5K unique architectures among them. Unsurprisingly, the isomorphism of architectures severely impacts search effectiveness and consequently causes a significant waste of computational resources. To combat this issue and identify/avoid isomorphisms, researchers have explored graph hashing [302, 303], customized DAG-to-string algorithms [65], and conjugate matrix ensembles [259]. Among them, hashing seems to be a particularly promising approach for helping AutoDL algorithms avoid duplicated computation [216].

BayesOpt-based NAS methods are based on, as the name suggests, a Bayesian formalism [138, 250, 279, 315], inheriting the strengths of the strategy applied to HPO in the context of AutoML. In fact, because BayesOpt was uniquely promoted into the NAS context from a DL-external community, it is more appropriate to elaborate on the approach when discussing HPO in Section 6.1. However, with the understanding that BayesOpt relies on an iteratively updated prior probability distribution to estimate DL model performance, there have been a few adjustments to the approach when extended from HPO to NAS. These have come in the form of succinctly encoding a candidate architecture as an input to the prior, as well as occasionally using a high-level neural network for the prior itself. Additionally, given the cost of evaluating a candidate DNN at each iteration, custom proxy training recipes are often employed; methods for boosting candidate evaluation are discussed in Section 5.3.

Differentiable NAS methods distinguish themselves from the aforementioned approaches by rejecting the limitation of a discrete and non-differentiable search space. The DARTS algorithm is seminal in this research area [171], seeking a relaxation into a *continuous* search space, so as to efficiently search architectures using optimization algorithms based on gradient descent. In a way, the associated continuous encodings resemble fuzzy sets or quantum superpositions, and they are eventually defuzzified/collapsed back into a discrete representation.

Notably, while DARTS popularized differentiable optimization in NAS, many issues in the original work were left unaddressed, such as the accuracy of gradients with respect to architectural encoding, the inconsistency between continuous and discrete search spaces, the implicit assumption behind linearly weighted sums for evaluating superposed networks, the bias of operator weights, etc. These issues have gradually been addressed by subsequent investigations [61, 63, 288, 291, 309]. For example, sophisticated differentiable HPO methods have been used to more accurately calculate gradients with respect to architecture [174]. A relationship has also been established between the performance of a DARTS-produced architecture and the eigenvalues for the Hessian matrix of validation loss, again calculated with respect to architectural encoding [309]; this was used as a regularization factor in early-stopping a search. Elsewhere, the Gumbel-softmax distribution [129] has been applied when discretizing architectural encodings [288], so as to alleviate the inconsistency between continuous and discrete search space.

Differentiable NAS remains popular, primarily due to three reasons: (1) it requires significantly decreased computational resources over alternative approaches; (2) the codebase for DARTS is open-source and easy to use; (3) DARTS is easily extendable. However, it does have two main drawbacks. First, the accuracy of architectures discovered by differentiable NAS is worse than those found via the RL-based procedures and evolutionary methods that currently claim the SoTA label for NAS [215, 264]. Second, the most appropriate representation/evaluation of superposed candidate networks in a continuous search space is still an open question.

5.3 Efficient Candidate Evaluation

Regardless of the black-box search strategy employed by a NAS algorithm, evaluating the accuracy of an architecture requires fully training it from scratch to convergence. This may cost several GPU days for a modern DNN on a large-scale dataset [106, 215]. Hence, it is computationally expensive to train/test even a single architecture, let alone the thousands of search-space samplings that may be needed to find an optimum, local or otherwise.

A straightforward and intuitive way to counter this cost is via a **low-fidelity approximation**, which is usually designed heuristically. As shown in Table 1, many works scale down the model [171, 324], sub-sample the dataset [63], reduce the number of training epochs [324, 325], set a constrained time budget [77, 175], early-stop the training [13], or explore different generalizable metrics [260]. These proxy strategies will decrease model accuracy as compared with a full training. However, by assuming models maintain proportionality in their relative performance, a comparative ranking of architectures can be estimated. Of course, the validity of low-fidelity approximations depends on whether this assumption holds true.

An alternative to low-fidelity approximation is the use of a **neural performance predictor** [52, 278]. This is a regression model that operates on architectural encodings [52, 278], a learning curve [39, 56], or both [14], to predict performance, e.g. final validation accuracy or latency [34]. The regression model can be non-DL-based [14], a simple MLP [52], or even something as advanced as a graph convolutional network [278]. After this predictor is optimized, it can be employed to boost the evaluation of candidate architectures in many ways, such as by replacing a low-fidelity approximation strategy [278] or augmenting it via early-stopping [14]. Naturally, training a regression model can still take numerous network evaluations, which is a nontrivial computational cost. Thus, these approaches are usually applied for NAS benchmarks [59, 65, 244, 303] and are still considered expensive for large-scale datasets.

Weight generation and weight inheritance are more efficient solutions for NAS. The SMASH algorithm [29] exemplifies the former approach by training a hypernetwork [100, 235] simultaneously with the search process, with the description of an architecture as input and its tuned weights as the target. The idea here is that a model generator in a NAS process should eventually, with support from the hypernetwork, be able to immediately flesh out any candidate architecture with its "correct" weights, no training required. Its effectiveness is only evaluated on a small-scale search space. When the search space becomes more complex and larger, training such hypernetwork would become evidently hard. Weight inheritance provides an alternative shortcut, where, for NAS procedures that employ mutations, a new candidate network can adopt some of the weights from a former candidate [33, 217, 299]. This strategy does not do away with retraining entirely, but there is an obvious speedup due to fewer model parameters that need to be tuned. In fact, weight inheritance can be taken to a further extreme in the notion of weight sharing [206], where all assessed candidates are sub-networks of – and thus share their weights with – a single giant "super-network". In this way, the cost of training millions of candidate architectures is amortized.

Weight sharing in particular has attracted much attention due to its simplicity and efficiency [206]. As shown in Table 1, it has become standard for efficient NAS. However, unsurprisingly, the efficiency of weight sharing has a drawback: search effectiveness [59, 65, 311]. As the weights of the super-network are tuned for itself, there is no guarantee that any candidate model among its immense number of sub-networks shares the same optimality of performance. It is certainly not clear how consistent/predictable performance degradation from weight sharing is, which, in turn, decreases the accuracy of relative rankings for all candidates. Of course, many works have tried to improve upon this technique by, for example, reducing the correlation between super-network and sub-network via path dropout [19], stabilizing model training via computing batch statistics on the fly [19], accelerating super-network training in differentiable NAS via the use of a straight-through Gumbel-softmax estimator [63], alleviating co-adaptation of shared weights via uniform sampling [62, 99], reducing inconsistent statistics between different candidates via switchable normalization [305], etc. While these strategies have improved the empirical performance of weight sharing, the issues mentioned above are still unsolved and lack theoretical analysis. Nonetheless, on the balance of efficiency versus effectiveness, weight sharing remains popular.

Indeed, for as long as DL wrestles with a paucity of computational resources, NAS research into shortcuts for both training and evaluating a candidate is expected to continue, and the most aggressive forms of evaluation are ones that **avoid training entirely**. For instance, appearing on arXiv in June 2020 [186], a publication proposed that, for any architecture, a model quality score could be computed by analyzing the activation map between different batch samples of data. This method showed competitive performance on both NAS-Bench-101 [303] and NAS-Bench-201 [65], but the results on the large-scale dataset are missing. Similarly, five metrics borrowed from the network pruning community have also been explored as potential performance estimators [2], with the analysis on more NAS benchmarks. Elsewhere, two metrics, neural tangent kernel [126] and the number of linear regions for a CNN [289], have both been considered for ranking architectures, likewise without training [43].

Ultimately, it is currently difficult to assess the right balance between search efficiency and search effectiveness. Some recent investigations report that applying NAS without wasting time on training weights has produced comparable – sometimes even better – DL models than previous SoTA NAS techniques [43]. But a general lack of comprehensive and diverse benchmarking in the field means that such a contentious debate is unlikely to be settled soon.

5.4 Overview

To assess the state of NAS research at the current time, consider first the search-space approaches evaluated in Table 3. As is evident, there have been numerous exotic propositions in this area and, overarchingly, it is hard to conclude anything about them beyond the fact that they rarely generalize well, often being customized for specific applications and downstream tasks. However, it is a little more straightforward to evaluate the mainstream approaches. Size-related search spaces make for a good baseline in this comparison, as they are simple and stably optimized, thus being easily engineered, reproduced, scaled up, and applied wherever. However, they do not lend themselves to particularly efficient searches, and their limitations preclude the discovery of high-accuracy neural networks.

For problems well handled by CNNs, NASNet-like search spaces were the first in the modern era of AutoDL to be employed. Their main selling point is accuracy, enabling novel SoTA topologies for DNNs to be found. On the other hand, as is to be expected for new experimental forays, this has come with a set of trade-offs that have not been entirely managed yet. Traversing these complex spaces remains relatively slow and inefficient, and the strong coupling between sequential choices in the search process makes it difficult for the algorithm to stably converge and scale up. There are

Table 3. Evaluative assessment for trends in NAS.

		Novelty	Solution	Effic.	Stability	Interp.	Reprod.	Engi.	Scalability	General.	Eco.
Search Space	Size-related	≈20	Medium	Medium	High	?	✓	✓	High	High	?
	Conv: NASNet-like	≈4	High	Low	Medium	?	✓	?	Medium	High	?
	Conv: MBConv-based	≈3	High	Mixed	High	?	?	✓	High	High	?
	Others	≈10	Mixed	Mixed	Mixed	?	?	?	Mixed	Low	?
Search Strategy	RL-based	≈5	High	Mixed	?	Medium	✓	✓	High	High	?
	Evolutionary	≈10	High	Mixed	?	Medium	✓	✓	High	High	Mixed
	BayesOpt-based	≈9	High	Mixed	?	Medium	✓	✓	Medium	High	?
	Differentiable	≈3	Medium	High	?	Medium	✓	?	High	High	?
Candidate Evaluation Boosts	Heuristic low-fidelity approx.	≈10	High	Low	Mixed	?	✓	?	Low	High	?
	Neural predictor	≈6	Medium	Low	Mixed	?	✓	?	Low	High	?
	Weight generation/inheritance	≈6	Medium	Medium	Medium	?	?	?	Medium	Medium	?
	Weight sharing	≈3	Medium	Medium	Low	?	?	?	High	Medium	?
	NAS without training	≈1	Low	High	?	?	?	?	Mixed	?	?

Each row marks an emergent trend in AutoDL, specifically NAS. Each column marks a criterion – see Section 2.4 – by which the trend is assessed. The evaluations are mostly qualitative, averaged across the most significant works researching the trend. Where a graded value is not provided, "✓" indicates a rigorous assessment is possible with analysis beyond the scope of this review, while "?" indicates that not even this is achievable without more research works to analyze. Novelty denotes years since seminal works in DL were published. Abbreviations are: "Solution" for Solution Quality, "Effic." for Efficiency, "Interp." for Interpretability, "Reprod." for Reproducibility, "Engi." for Engineering Quality, "General." for Generalizability, and "Eco." for Eco-friendliness.

also no popular libraries available that are suited to handling the graph representation required by NASNet-like search spaces, making it difficult to comment on engineering quality, although there has been interest in the trend for sufficiently long enough that research results have been benchmarked to a degree [65, 244, 294, 303].

Search spaces that are MBConv-based are a slightly more recent proposal, partially reacting to the inefficiencies present with NASNet-like spaces. Arguably, their effectiveness still relies on secondary factors, such as whether MBConv-based searches leverage weight-sharing or rely on multiple trials. Nonetheless, the representation of a candidate architecture in an MBConv-based space is not as complicated or tightly coupled, meaning that, with configurations expressed in list format, it is relatively easy to slot in well-engineered NAS/HPO libraries. This ease of network representation also pays dividends in terms of scalability and the stability of optimization convergence. However, research incorporating MBConv-based search spaces has still not been benchmarked heavily, and – this is an issue across the board – there have been no strong theoretical analyses of search spaces, meaning that it is not perfectly clear why certain arrangements produce better DL models than others.

The onus of attaining good DL performance ultimately falls on search strategies though, and all four trends in Table 3 are ranked as strong contributors to model accuracy. Certainly, they are far more robust than manual searches, although the debate about random search and its own effectiveness remains [160]. However, reflecting the challenges of training complex DNNs, the efficiencies of most are rated as mixed, strongly dependent on candidate evaluation boosts. In fact, applied in standard fashion or even with low-fidelity approximations, these search strategies are considered slow and resource-intensive in the literature, with only weight-sharing and related techniques managing to increase the efficiency score. The one relatively speedy exception is differentiable NAS, which tries to relax search spaces so that gradient descent is a valid procedure; this experimental technique results in model-accuracy claims that are slightly less impressive, although these may improve as the youngest trend acquires more research focus.

Overall, search strategies in NAS are often treated as generalizable black-box optimization methods, i.e., very high-level, that may as well have coincidentally been employed in the search for accurate DNNs. They are thus subject to standard limitations that do not depend on the NAS context, such as BayesOpt struggling to scale well in search spaces with extremely high dimensions. Moreover, there are also many good software libraries available for the four approaches, with the possible exception of differentiable NAS. This general applicability also means that the strategies are regularly benchmarked, opening themselves up to reproducibility assessments, and, likewise, a moderate level of theoretical analysis exists. The assertions made by these investigations, however, are limited by a reliance on conditions that may not hold in practice, especially once approximations and proxies are included in NAS. To date, they also have little to say about the convergence stability of the studied search methods, at least with respect to NAS.

Moving down to a lower level, much more variation exists when assessing the diversity of options for boosting efficiencies in candidate evaluation. As Table 3 suggests, the passage of time has witnessed progressively more dramatic attempts to speed DL up, from low-fidelity approximations to the recent NAS-without-training proposal. The assumptions underlying each new trend become seemingly looser and more radical, which lessens accuracy guarantees on the final DNN that NAS produces. Conversely, as intended, the speed of NAS is substantially accelerated, especially after the introduction of weight generation, where candidate evaluations are no longer necessitated.

Given the number of years over which low-fidelity approximations and neural-predictor estimations have been researched in the context of DL, there already exists some commentary on the results and reproducibility of both the former [70] and the latter [294]. The stability of both methods is very dependent on multiple factors, e.g., the proxy task used, and they scale linearly with respect to the size of a search space. They are also usually agnostic to architectural design and are thus very generally applicable. In contrast, weight generation and sharing need customization for specific architectures and search spaces, with low guarantees on stable convergence. However, the assumptions involved mean that, while the approaches do still need extra evaluations as a search space increases to provide optimal benefit, the scaling relation is attractively sublinear.

In terms of eco-friendliness, first impressions suggest that NAS has quite a problem [255], and, certainly, minimizing power usage remains a significant challenge. However, it remains contentious within the DL community as to how dire the impact truly is. As expected, intelligent use of efficiency boosts can substantially reduce carbon emissions when searching through a large space for an optimal architecture [201]. The debate has been further confounded by an assertion in early research work [255] that NAS algorithms must be run from scratch every time a model is freshly trained. In contrast, other publications [156, 201] argue that, in practice, NAS is only ever performed once per combination of problem domain and architectural search space. They also claim that other discounts in carbon emissions are often overlooked, especially when considering the billions of inferences that could have otherwise been made with an inefficient manually constructed DNN. In truth, this debate is unlikely to be settled without many more extensive investigations.

For now, throughout Table 3, a lot of unknowns remain. The more novel a trend is, the less it has been rigorously studied. Indeed, it is challenging to comment on reproducibility for weight generation and sharing due to a dearth of benchmarking. In the case of NAS without training, which is effectively a brand new proposition, the stability and generalizability of the approach are entirely uncertain at the current time. Nonetheless, even accounting for the passage of time, broader gaps in theory and practice remain. For instance, how exactly these techniques relate to performance is not presently interpretable, beyond the common-sense understanding that making shortcut assumptions results in increased speed. Likewise, the methods have not been implemented yet in any notable libraries. So, for all the attention that NAS has received from the DL community, there is plenty more progress to make, even within this core aspect of AutoDL.

6 HYPERPARAMETER OPTIMIZATION

The definition of a hyperparameter is blurred in the literature. In the broadest sense, they are traditionally human-chosen parameters – model-based and algorithm-based – that control a process of learning; they are not determined by that learning process. However, for various historical reasons, the full implication of this definition has often gone unrecognized. This is why the decision to treat ML model type as a hyperparameter was itself considered a surprising innovation of AutoML, allowing model selection to be repackaged into a broader CASH problem [265].

Now, granted, some DL researchers have considered architectural structure to be a set of hyperparameters [55], but early-NAS effectively developed without strong awareness of the AutoML community. This siloed approach would eventually be rebuked, with a publication stating that, "while the NAS literature casts the architecture search problem as very different from hyperparameter optimization, … most NAS search spaces can be written as hyperparameter optimization search spaces" [310]. The paper would go on to challenge the then-predominant approach of running NAS first and then optimizing (the remaining) hyperparameters as an independent post-hoc step. However, this distinction between two processes has stuck, and it is currently a convention in AutoDL that the word 'hyperparameter' often relates solely to the training algorithm [60], e.g., representing learning rate, weight decay, dropout rate, etc. We maintain this convention in this review.

Within such a context, HPO in AutoDL has certain unique differences from HPO in AutoML, and they are not purely semantic. For one thing, AutoML works with a diversity of model types and training algorithms that discourages an optimizer from making assumptions ahead of time. In contrast, because all DNNs are based on the same universal approximator in the form of an artificial neuron, training algorithms are fewer in number; it is possible to have a favorite choice without neglecting outright better performers. It follows that, while the type of training algorithm can still be made searchable [52, 60], a parameter like learning rate may be more efficiently optimized [16] if the selected training algorithm is known to be SGD [225] or Adam [141]. We thus classify AutoDL HPO algorithms by how much they need to know about the training algorithm applied to a base model: black-box, gray-box, and white-box.

6.1 Black-box HPO Approaches

Black-box HPO approaches have a long-standing history [49, 250]. In the context of HPO in AutoDL, they assume that a training procedure defined by a candidate set of hyperparameters can only be evaluated by the end result of the process, e.g., the accuracy of a DNN that is trained. Thus, they are exceedingly general; the underlying search techniques have presence in AutoML, and they can be directly applied to other optimization problems in AutoDL, including Automated Data Engineering, NAS, and Automated Deployment. Broadly speaking, there are three popular categories in AutoDL-based HPO: RL [239], evolutionary approaches [93], and BayesOpt [250].

Reinforcement Learning: As with NAS in Section 5.2, RL-based HPO can be enacted by designing/extending a controller to sample candidate hyperparameters. Every time these hyperparameters are selected, a corresponding training procedure is instantiated to train a DL model, which may itself have been selected by a controller. Evaluation metrics, such as accuracy on a validation set, are used to judge both the candidate DNN and training procedure in tandem. These metrics represent a reward, and RL algorithms work to maximize this reward [6, 239, 276, 281], thus teaching the controller to sample better candidates in the future, whether or not architectural choices are rolled into that search space.

An open challenge for RL-based HPO is how best to reformulate hyperparameter search into an RL problem. This optimization has previously been treated as a sequential selection of hyperparameters [134] but, recently, it has also been simplified into a single-step decision-making problem [60]. There are many design choices to be made with RL strategies, and broader discussions can be found elsewhere [258].

Evolution: To apply evolutionary algorithms [54, 93, 229, 234], a population is created by randomly sampling candidates from a hyperparameter search space. In this population, each candidate is represented as an encoding, treated analogously to a string of Deoxyribonucleic Acid (DNA). Typical evolutionary algorithms will iteratively (1) alter candidate encodings within the population, (2) train and evaluate a DL model subject to each set of candidate hyperparameters, obtaining validation accuracy as a fitness metric, and (3) remove low-fitness candidates from the population, replacing them with higher-fitness encodings [93]. In this way, the population is progressively improved and, at some stopping point, the highest-fitness candidate is selected as an optimal set of hyperparameters. Of course, there are many ways to alter/replace encodings, e.g., via mutation or crossover, so there are many variants of evolutionary algorithms in existence [217].

BayesOpt: Despite the sophistication of RL and evolution-based techniques, random search is also a common option, and it can be surprisingly effective in practice [25, 160, 306]. In general, though, it is assumed that principled search methods can navigate to optima more efficiently. With all the potential "messiness" of hyperparameter space, from discontinuities to conditional variables, BayesOpt methods have proven particularly popular and effective [24, 130, 250, 310]. These consist of two components: a Bayesian-based surrogate model for estimating how a candidate set of hyperparameters maps to a performance metric, based on evaluations already made, as well as an acquisition function that decides where to sample next, so as to iteratively rein in the performance estimates of the surrogate.

While RL and evolutionary algorithms have been around for a while, the development of BayesOpt is what propelled AutoML into a broader spotlight within the early 2010s [121, 122]. However, all three types of techniques have representation in AutoDL. For now, it remains an open question as to whether one approach is better than another, and in which problem settings. Some preliminary works have benchmarked different HPO algorithms on small datasets and search spaces [69, 144], but more investigation is required to generalize these conclusions to large-scale scenarios, especially to bolster confidence in any comparative rankings.

6.2   Gray-box HPO Approaches

Black-box optimization is flexible, but if one can be confident in assumptions/knowledge about what lies "inside the box", it is often possible to search through a space of solutions far more efficiently. This is often unofficially referred to as *gray-box optimization*. By definition, its applicability is very dependent on the search problem of interest, and associated methods are often just upgraded forms of the generic black-box techniques described in Section 6.1.

Within HPO specifically, multi-fidelity optimization is among the most popular gray-box approaches [77, 83, 128, 137, 139, 159], where variably cheap and accurate proxies/estimates of model performance are leveraged to aid the search. For instance, if one is able to train models on small amounts of data, i.e., low-fidelity approximations, it is possible to quickly extrapolate these performances into a full learning curve [39, 56, 261]. This predictive curve can provide advice on many matters, e.g., whether to continue training, whether to add more computational resources, or whether to 'early-stop' an unpromising set of hyperparameters. The strategy of successive halving is another technique that similarly starts with low-fidelity approximations [139]. It evaluates candidates trained on minimal data/time, throws away the worst half, evaluates the remnants on an increasing amount of data and computational budget, throws away the worst half, and so on;

eventually one high-fidelity evaluation is left. This process has since been refined into an algorithm named Hyperband by hedging its aggressiveness [159], and Hyperband has subsequently been fused with BayesOpt techniques into BOHB [77], which has proven itself a highly efficient and effective HPO strategy for certain "well-behaved" datasets [65, 144, 295, 311]. Why then are these approaches considered gray-box? Their performance depends on the extrapolation of low-fidelity approximations to be predictable and well-behaved, which benefits from some understanding of – or confidence in – hyperparameter space.

There are other gray-box search approaches that likewise make assumptions on the behavior of DNN training algorithms when hyperparameters are varied. For instance, certain shortcuts can be made if a DNN is assumed to be trained via gradient-based means [127]. Elsewhere, there has been a study of what happens when intelligently decomposing a black-box objective into composite functions, one of which is cheap to evaluate [11]. Of course, if a hyperparameter space is reasonably familiar or well-understood, HPO methods can also be warm-started with good hyperparameter candidates. This has been done for both an RL controller [60] and evolutionary algorithms [155, 251].

Further works are listed in the HPO row of Table 1, where the column of "Boosts for Candidate Evaluation" indicates the shortcuts being employed; these imply which inside-the-box assumptions about search space/strategy are being made. Importantly, while gray-box approaches have been very successful in trading off generality for efficiency, they still require some level of sampling, i.e., fully training/evaluating a candidate model, and this computational expenditure is not negligible. Thus, in practice, black-box and gray-box methods can both be infeasible for large DL models.

6.3 White-box HPO Approaches

What if we fully open up the black box? Unlike AutoML, which often juggles many disparate ML models/algorithms, a significant portion of DL involves feed-forward neural networks that all share the same fundamental principles. Many of these principles relate to the layered nature of a DNN. Chief among them is that, via the chain rule, one can calculate how a change in any weight parameter corresponds to a change in network performance, i.e., an error gradient. Indeed, while this notion of backpropagation has been explored in AutoML [189], with parallels between ML pipelines and DNNs discussed in Section 2.2, it remains particularly appropriate for DL due to the mathematics involved.

So then, can error gradients with respect to hyperparameters – hypergradients – also be computed/leveraged? After all, if training a model gradually tunes model parameters, then why not hyperparameters too? Sure enough, researchers have pursued this thread from before the 2000s. For instance, the gradient of cross-validation error with respect to weight decay has previously been calculated within a simple single-layer network, and this hypergradient was used to adjust weight decay during network training [150]. Other contemporary work made computing hypergradients somewhat simpler by developing a relation between hyperparameters and network weights, then leveraging this via the implicit function theorem [21]. Since then, hypergradient descent methods have continued to see strong attention, with, for instance, an algorithm being proposed to update hyperparameters by computing reverse-mode derivatives across truncated gradient descent steps [57]. A subsequent effort would upgrade this approach via the computation of exact hypergradients, additionally wrestling with the substantial memory-based storage costs of the procedure [180].

Notably, many early attempts focus on calculating "exact" hypergradients, which is computationally expensive for a DL model. Thus, to improve scalability and generalizability, researchers have recently developed different approaches involving approximate hypergradients. Some have considered gradually tightening the accuracy of such an approximation during the course of training, i.e., via an exponentially decreasing tolerance sequence [203]. Others have analyzed truncated

Table 4. Evaluative assessment for trends in HPO.

	Novelty	Solution	Effic.	Stability	Interp.	Reprod.	Engi.	Scalability	General.	Eco.
Black-box HPO	≈ 30	High	Low	High	High	✓	✓	Low	High	Low
Gray-box HPO	≈ 30	Medium	Medium	Medium	Medium	✓	✓	Low	High	Medium
White-box HPO	≈ 21	Medium	High	Low	Medium	?	?	High	Low	High

Each row marks an emergent trend in AutoDL, specifically HPO. Each column marks a criterion – see Section 2.4 – by which the trend is assessed. The evaluations are mostly qualitative, averaged across the most significant works researching the trend. Where a graded value is not provided, "✓" indicates a rigorous assessment is possible with analysis beyond the scope of this review, while "?" indicates that not even this is achievable without more research works to analyze. Novelty denotes years since seminal works in ML were published. Abbreviations are: "Solution" for Solution Quality, "Effic." for Efficiency, "Interp." for Interpretability, "Reprod." for Reproducibility, "Engi." for Engineering Quality, "General." for Generalizability, and "Eco." for Eco-friendliness.

backpropagation for use in approximating the gradients of weight parameters with respect to hyperparameters [241]. Another lingering issue is that hypergradient calculations often rely on the expensive computation of an (inverse) Hessian, i.e., the second-derivatives of model error, which is infeasible for large-scale networks and/or a large number of hyperparameters. Efforts to surmount this challenge include approximating the Hessian matrix by an identity matrix [176] and approximating the inverse Hessian matrix by a Neumann series [174].

In summary, it is clear that white-box HPO can be far more efficient than black-box/gray-box HPO; hyperparameter updates can be applied per forward/backward pass during model training rather than after the model is evaluated. In effect, white-box approaches roll HPO *into* the process of model training. However, hypergradient methods rely on mathematical equations that embody several assumptions, e.g., the continuity and differentiability of model loss with respect to hyperparameters, and, based on HPO setup, these do not always hold true.

6.4 Limitations in Applicability

Many aforementioned HPO methods and upgrades are grounded in strong theoretical bases. None, to date, stand out exclusively among the rest. This is no surprise, as the no-free-lunch theorems apply to optimizers at any level [283]. That does not mean that *certain* sets of hyperparameter spaces do not have an optimal HPO strategy; this has been explored by optimizing a hyperparameter optimizer in the form of an RNN [9, 44, 158][6]. But the point stands: the applicability of AutoDL-based HPO mechanisms must be carefully considered when choosing one for a real-world problem.

Crucially, this section has shown that HPO methods contend with a trade-off between generality and efficiency. Any principled strategies beyond purely random search need to leverage some degree of knowledge/assumptions about a search space, and, in return for quicker/better searches, these requirements become more restrictive along the spectrum from black-box to white-box. Granted, hyperparameter space can already be significantly complex and messy, even with the AutoDL limitation to model training procedures. For instance, black-box BayesOpt has long grappled with surrogates for dimensions that can be continuous, categorical, or conditional [121, 122]; research continues in this area almost a decade later [53, 223]. However, white-box hypergradient-based HPO methods rely on differentiability, and this efficiency extreme can thus only be applied to certain selections of continuous hyperparameters [174, 176, 203], such as learning rate [16] or continuous regularization [176].

Nonetheless, while the generality-efficiency trade-off will likely always remain, HPO research continues to push the boundary. For example, population-based training (PBT) proposes to train

---

[6]We have no official stance on whether "hyperhyperparameter" should be introduced into the AutoDL lexicon.

a group of models together under different sets of hyperparameters, where those individual sets are tuned depending on how the rest of the population is faring [127]. This is a joint optimization of parameters and hyperparameters that does not involve hypergradients, discarding their differentiability restrictions. It is thus fast, but the computational cost now depends on the scale of parallel training involved. Elsewhere, a bilevel optimization procedure has introduced so-called "best-response functions" as trainable mappings between the values of hyperparameters and corresponding optimal network parameters [179]. This work likewise avoids hypergradients and their limitations, allowing the training-simultaneous tuning of discrete hyperparameters, data augmentation hyperparameters, and dropout probabilities. Also of note is another recent effort that aims to maintain general applicability to hyperparameters, encapsulating the procedure of applying hyperparameters to model weights as a black box [60].

The take-away from this discussion is that, as with NAS, HPO in AutoDL continues to see a flurry of research, with numerous novel techniques being frequently proposed. However, *also* as with NAS, HPO in AutoDL is still arguably in a nascent stage. Systematic benchmarking is limited, making consensus comparisons difficult. The technical reason behind this is clear, namely the computational expense of running NAS and HPO. This is why, as summarized in Table 1, existing HPO methods have mainly experimented on small-scale models, e.g., linear models or shallow networks, as well as datasets that are either small or synthetic. Nonetheless, there have been recent HPO investigations on larger-scale datasets, such as the CIFAR-style AlexNet [174] and the vision dataset ImageNet [52, 60]. It is simply a matter of time. As computational resources increase and the demand for NAS/HPO in real-world applications grows, circumstances will eventually drive more rigorous assessments of applicability.

6.5 Overview

Black-box optimization methods, gray-box shortcuts and even the fundamentals behind white-box approaches [21] were all introduced to the ML community several decades ago, as noted within Table 4. In general, black-box methods rely on thorough optimizations with complete evaluations of candidate networks and are thus most accurate and stable. They are slow, however, and the assumptions underlying gray-box and white-box approaches – every shortcut used arguably weakens the interpretability of the method – sacrifice accuracy for progressive improvements in search efficiency. In this case, it is also reasonable to associate the efficiencies with a reduced reliance on computational resources and, accordingly, a better ranking for eco-friendliness.

Crucially, gray-box HPO can be considered as black-box HPO with efficiency boosts, while white-box HPO relies on intrinsically different optimization methodologies, leveraging implicit/explicit assumptions that are particular to neural networks. This means that the accuracy trade-off for white-box HPO, fine-tuned for DNNs, is not as severe as might be expected. The efficiencies of white-box HPO, a result of dodging multiple candidate evaluations, also makes related approaches highly scalable. However, this close tie-in to DNN formalism does mean that white-box HPO is heavily dependent on problem context and search space, while black-box and gray-box HPO are relatively generalizable. Accordingly, black-box HPO, whether modified to be gray or not, has been benchmarked heavily and implemented within many software packages; future in-depth surveys may comment further on reproducibility or engineering quality. White-box HPO needs more experimentation and analysis.

7 AUTOMATED DEPLOYMENT

The topic of deploying an ML model into a production environment is an immense one, straddling theoretical principles and real-world practicalities. Generalized commentary is further complicated by just how many ways an ML model may be used. Will it serve as the predictive back-end of a

queriable web app? Will it be hooked into a robotic framework as a prescriptive system? Will it interface with a high-fidelity "digital twin" of physical reality [89]? The field of AutoML has barely begun grappling with the notion of automated deployment, and much of this discussion occurs beyond academia, with best practices for machine learning operations (MLOps) being hashed out by commercial entities [183].

Nonetheless, when it comes to DL specifically, particular trends of research stand out, driven primarily by resource concerns; DNNs are heavyweight models in terms of both storage and inference. Given that deployment settings can range from edge devices to the cloud, AutoDL strives to answer two mirrored questions:

(1) Can a DL model be optimized for a specific production environment?
(2) Can a production environment be optimized for a specific DL model?

7.1 Deployment-aware AutoDL

Many DL projects have rigid deployment constraints; the onus is on the model to accommodate these requirements. Thus, while maximal predictive accuracy is still a primary objective, secondary objectives may involve inference latency, memory footprint, and energy cost. In AutoDL-related literature, model-construction efforts that focus on these considerations are given differing names. For example, there is "platform-aware NAS" for accommodating mobile devices [263, 299], "energy-aware pruning" for constraining network connectivity [298], and other research published as latency-aware [34] or resource-aware [290]. Accordingly, we generalize such approaches under the banner of deployment-aware AutoDL.

There are several common approaches to dealing with multiple objectives [123, 182]. One of the simplest is constrained optimization, where a target metric such as inference latency is given an upper bound, and any architectures that do not operate within the tolerable range are discarded or, if possible, adjusted back into that range [298, 299]. However, if the constraints are poorly behaved, i.e., highly nonlinear, too many unsatisfactory candidates may be constructed during exploration, which is inefficient. In addition, sampling fully constructed models to evaluate other objective functions negates the innovative shortcuts behind white-box HPO.

Other options for multi-objective optimization via NAS and HPO are also available. A common alternative is to bundle all target metrics into a single one, using this combined objective function to guide AutoDL search algorithms. Such efforts often focus on network latency, although these values must usually be estimated; efficient AutoDL algorithms cannot spend time evaluating every candidate architecture within an actual production environment. A typical estimation process then is to (1) pre-compute the latency for hundreds of candidates, (2) train a small DNN on these values to predict latency in general, and (3) use this predictor to approximate the latency of candidate architectures during the AutoDL process [20, 34, 316]. Consequently, previous investigations have explored algebraic combinations of model accuracy and latency, including a re-scaled multiplication [263] and an addition [34]. The latter of these used the metric for differentiable NAS [34], but latency has also been factored into a reward function for RL-based NAS [20]. Other examples of combined metrics also exist, e.g., applying a piece-wise function for the secondary objective [61] or an absolute function for the re-scaled secondary objective [20].

Sometimes the demands of a production environment can be a little more niche, such as when a device does not support full-precision computing; this can be desirable when aiming for cheap and fast DL applications. The Infineon XC800 family of microcontrollers exemplifies this, operating in 8-bit. So, to convert a model trained on a higher-precision processor, the typical solution is to use the so-called quantization technique [102], which approximates the original network by another one with low-bit weights. However, it is highly possible that, even for the same DL task, the optimal

architectures on two platforms with different computing precision may be structured/connected completely differently [274, 301]. Ideally, NAS should be undertaken with quantization already in mind, and several works have explored this angle [31, 273, 274, 285].

Ultimately, it is an inescapable fact that numerous different deployment environments exist in the real world, each possibly having its own requirements in terms of latency, energy, etc. Thus, while there are many approaches for building bespoke DL models, it becomes infeasible to constantly reconstruct models for any applications that are designed with broad release in mind. Unsurprisingly, a cross-platform ethos has been embraced by certain investigative works. For instance, one attempt proposes designing a once-for-all (OFA) network that supports diverse architectural settings, such that any supported environment works well with a unique sub-network of the OFA model, no further training required [32]. There are other efforts that likewise seek to obtain multiple models for target environments with but one search [304]. Naturally, while these shortcuts boost search efficiency, the accuracy of each sub-network cannot be guaranteed. Thus, these methods often utilize techniques of knowledge distillation [110] to transfer knowledge between super-networks and sub-networks. Nonetheless, further research is required to compensate for the accuracy drawbacks of OFA approaches. For now, it appears that planning a DL project cannot remain agnostic with respect to its eventual deployment environment.

7.2 Hardware Search

What if the opportunity arose to mold a deployment environment around a DL model? Such a circumstance would seem relatively rare at the current time, due to the typical rigidity of hardware constraints, but some commercial/industrial entities have both the capacity and will to be flexible in how they provision resources.

This becomes, as with the majority of research into automation along the DL workflow, an optimization problem. Indeed, all discussions about black-box/gray-box optimizers in Section 6 remain relevant here, but the search space now includes hardware. For instance, multi-objective BayesOpt has been used to co-design both a neural network and an associated energy-efficient AI accelerator, with the latter using Complementary Metal–Oxide–Semiconductor (CMOS) technology [214]. This effort explored a 14-dimensional search space with one NAS variable, eight (training-based) HPO variables, and five hardware variables, e.g., bit-length and memory bandwidth.

In fact, this co-design approach is fairly typical in the field, and it is not the only investigation that has leveraged multi-objective BayesOpt. The search technique has been applied to accelerators built from a Field-Programmable Gate Array (FPGA) [194], while a variant of BayesOpt has also optimized Programmable Ultra-efficient Memristor-based Accelerators (PUMAs) alongside DNNs trained on AlexNet/VGG [200]; the architecture configurations here were relatively simple, covering kernel size, width, and depth. Of course, other search techniques have been applied too, with RL being used to co-optimize an FPGA-based accelerator and an MBConv-based architecture [132]. Likewise, RL has been employed in producing chip floorplans for the next generation of Google-designed AI accelerators [191]. Additionally, given that black-box methods can be computationally expensive, white-box methods have also seen their share of usage [47, 316], e.g., one effort [47] applies differentiable NAS/HPO to optimize accelerators based on Eyeriss [46] while searching for MBConv-based architectures [263].

Importantly, while hardware search presently appears to be a promising research direction for AutoDL, especially with its comparative novelty versus NAS, long-term impact is arguably weakened by the current lack of consensus about AI accelerators. Basically, AI acceleration is an emerging technology with no dominant design, and research associated with a particular format relies heavily on the success and uptake of that format. Hence, there is ongoing debate around

Table 5. Evaluative assessment for trends in automated deployment.

	Novelty	Solution	Effic.	Stability	Interp.	Reprod.	Engi.	Scalability	General.	Eco.
Deployment-aware AutoDL	≈ 4	High	Mixed	Medium	High	✔	?	Mixed	Mixed	Mixed
Hardware Search	≈ 5	High	Low	?	Low	?	?	Low	Low	?

Each row marks an emergent trend in AutoDL, specifically automated deployment. Each column marks a criterion – see Section 2.4 – by which the trend is assessed. The evaluations are mostly qualitative, averaged across the most significant works researching the trend. Where a graded value is not provided, "✔" indicates a rigorous assessment is possible with analysis beyond the scope of this review, while "?" indicates that not even this is achievable without more research works to analyze. Novelty denotes years since seminal works in DL were published. Abbreviations are: "Solution" for Solution Quality, "Effic." for Efficiency, "Interp." for Interpretability, "Reprod." for Reproducibility, "Engi." for Engineering Quality, "General." for Generalizability, and "Eco." for Eco-friendliness.

what type of systems to focus on. For instance, one work disregards FPGAs in favor of an "industry-standard" edge accelerator [316], while another explores co-optimization for Application-Specific Integrated Circuits (ASICs) [296], which are flexibly designed and thus powerful but often tricky to standardize. Simply put, the evolution of this research sub-field will depend on how the standards of computational hardware themselves evolve.

Beyond training/inference acceleration, the management of hardware resources is also a potential target of automation, explored as early as in the 1990s [313]. This broad topic has many particular incarnations in AutoDL, e.g., how to delegate pipelined DL operations among various devices [3, 190, 192], how to effectively administer different DL jobs within a computing cluster [205], how to schedule execution for a DL compiler [42], etc. In essence, optimization algorithms here are challenged by the extremely distributed nature of a production environment, potentially managed at different levels and operating in distinct ways, e.g., online versus offline. These sorts of issues may become more and more important as time goes on and new principles of computing become mainstream.

As a final assessment, the endeavor of automating hardware search is a promising extension of AutoDL beyond the theoretical focus of NAS to the practicalities of production environments. It is not a dramatic change in perspective, with many of the works discussed above and summarized in Table 1 reusing existing NAS/HPO algorithms; this bodes well for the future design of autonomous AutoDL systems that aim to integrate all DL workflow processes under one umbrella. However, the real world is messy and definitely not standardized. While it is reasonable to expect a data scientist to have familiarity with the fundamentals of DL, many of the reviewed works require specialized expert knowledge of hardware accelerators, distribution systems, and so on [190–192, 214, 316]. For instance, domain knowledge is required when deciding how to avert any undue impact on search efficiency caused by the simulation cost of a hardware accelerator [191, 316]. In effect, this means that there is a high barrier to entry for research and development around this form of automated deployment, simply in terms of hardware-related expert knowledge, computational resources, and software infrastructure. Hopefully, this will be ameliorated in time, as existing automation work has already proved itself very valuable to ML-focused engineering and the next generation of AI accelerators [191].

7.3 Overview

The two trends in automated deployment focus on optimizations in opposing directions. Put simplistically, one adjusts software for the sake of hardware and one adjusts hardware for the sake of software. Thus, despite these endeavors entering the domain of DL at roughly the same time, it is not surprising that hardware manipulation currently scores much worse over a broad range

of assessment criteria, as listed in Table 5. Research in both trends is of course equally valuable, each demonstrating that model performance and thus accuracy is maximized when production and deployment environments are considered together. Nonetheless, progress in hardware search is stymied by the great variety of physical architectures that exist; the proponents of any novel procedure will also need to promote a particular system to the rest of the DL community.

Accordingly, research in the sub-field of hardware search remains more scattered and exploratory. A lack of benchmarking and standardized implementation makes it difficult to comment about the reproducibility of results and their stability, let alone how complicated proposed techniques are to engineer. The presence of multiple layers between physical systems and abstracted DL algorithms also affects interpretability, obscuring how exactly hardware manipulations contribute to positive DL outcomes. Moreover, because physical environments can typically only be optimized via gray-box procedures at best, efficiency boosts are limited and associated search procedures do not scale well with the complexity of hardware systems.

In contrast, research in deployment-aware AutoDL is a little more unified, enough for some initial analysis of both stability and reproducibility [157]. Admittedly, there are only a limited number of software libraries that currently implement related techniques, but the fundamentals underpinning the adjustment of DL models for deployment environments are fairly well understood, at least in contrast to hardware search. Beyond that, the efficiency, scalability and generalizability of deployment-aware AutoDL cannot be simply summarized, as these criteria depend far too much on the specifics of the algorithm in question. The assessments in Table 4 are informative in this case; white-box optimization methods will be efficient and scalable, but not generalizable, while black-box and gray-box methods will reverse those qualities.

8 AUTOMATED MAINTENANCE

The future of AutoML, at least in the short to medium term, is continuous learning; this is what a previous review identified and argued [140]. In fact, there is currently a disconnect between how the majority of ML models are designed/used – "one-and-done" – and what the growing demands of real-world applications are. Recent global events have shown that models built on static assumptions can be very fragile[7], highlighting a dire need, depending on industry and problem context, for continuous monitoring and maintenance. We label systems that are able to automatically provide this support as AutonoML frameworks, with the understanding that true autonomy is impossible without the capacity for persistent automated management of models. For now, there is no archetypal AutonoML package that we can highlight, but many vendors have listed adaptation capability on their development roadmaps, researchers have published prototypes of adaptive ML systems even before the modern wave of AutoML [136, 228], and there is a growing number of academic research efforts in this direction [37, 181]. Evidence shows that initial steps are being taken in the evolution from AutoML to AutonoML.

So, what about "AutonoDL"? Does such a research parallel exist? The answer to this is very nuanced. At zeroth order, it is reasonable to assert that continuous learning is not currently a major priority in the DL community. This is understandable, as the major selling point of DL is in fact its representational power. Manageably complex models are great for approximating complicated input-output mappings. The corresponding downside though is that training a DNN well often relies on the repeated presentation of concepts in data. These models are not agile, at least traditionally, and they fare poorly with conceptual instability [184]. Nonetheless, although continuous learning may not currently be practical, a lot of recent effort has gone into making

---

[7]https://www.dominodatalab.com/blog/how-covid-19-has-infected-ai-models

repeat model development as flexible as possible, designed to accommodate changes in both data and task.

Therefore, after briefly considering the ideal of online learning in Section 8.1, and why it is difficult, we discuss relying on continuous monitoring instead in Section 8.2, covering possible data-environment dynamics to watch out for, before reviewing popular paradigms for automatic model maintenance in Section 8.3.

8.1 The Challenges of Online Learning

Learning within dynamic data environments is not a new topic. Over several decades, many bespoke ML models/algorithms have been proposed with the aim of operating on streams of data, e.g., incremental decision trees [209]. In the ideal case, these ML models should continue to develop their structures and tune their parameters and all associated pre-processing or data transformation steps [322] while they are deployed, either on delayed training data or on some feedback to query-based model responses, e.g., from a user, a digital-twin simulation, a robotic system, etc. So, why not simply mimic this approach and leave a DNN in training mode during deployment? Unfortunately, all the challenges of standard DL training immediately apply, but now in an environment that is difficult to control. For instance, in the case of backpropagation, deciding on batch size is already a challenge [247], as it determines how much a single instance of data affects a model. Unsurprisingly, it is even more difficult to curate an informative batch of queries and feedback-derived labels on the fly, especially one that provides sufficient repeat presentation of any important but infrequently observed concept.

Even then, best practices of training aside, the fundamental obstacle that typical DL models face is catastrophic interference/forgetting [184, 266]. Neither network depth nor connectivity, despite other benefits, helps isolate learned concepts; the latter actively competes with information localization. Thus, when an input-output mapping is approximated by a monolithic model, new concepts can easily overwrite old ones. This can be highly undesirable for a sequential learner, e.g., if an NLP text generator encounters low-quality writing after training on a high-quality corpus. Simply put, monolithic models are particularly susceptible to experiencing "garbage in, garbage out" (GIGO).

Admittedly, there has been plenty of contemplation around how neural networks may operate in a persistent manner, engaging in lifelong learning while carefully managing the stability-plasticity dilemma [198]. Some attempts to address this draw inspiration from neuroscience, even if the associated adaptive mechanisms are often better suited to models built from more biologically realistic neurons, e.g., spiking neural networks [68]. All the same, there are several strategies for standard artificial neurons that recur throughout the literature. For instance, one approach relies on regularization [142, 162], seeking to avoid catastrophic interference by controlling network weight updates with constraints. This might be a quadratic penalty scaled to the difference between old and new parameters, which slows down how quickly previously learned information is overwritten [142]. Alternatively, one can try to force information locality via some form of ensembling, often employed for data stream analysis [146], or even fully dynamic architectures [227], which grow out new sub-networks as required. There are yet other efforts that attempt to codify dual memories [111], first proposed early on [111], so as to have one set of weights maintain certain concepts in long-term storage and have another set of weights adapt to changing data dynamics. Nonetheless, with few exceptions, such strategies for lifelong learning remain relatively unknown within the DL community, given that managing a standard DNN is already computationally expensive enough as it is.

In short, for many theoretical and practical reasons, continuous model updates are not presently feasible in the field of AutoDL, yet the need to respond to dynamic data environments remains. Thus, rather than online development, the focus turns to a more relaxed form of continuous learning: updates as required. The challenge of catastrophic interference never disappears, but it can now be faced with deliberate intention. However, as a consequence of this approach, the process of adaptation now splits into two, namely continuous monitoring and maintenance, or "when do I do it?" and "what do I do?", respectively.

8.2 Scenarios for Continuous Monitoring

Many scenarios are possible in a dynamic data environment. To ground this discussion, we define "domain" to be some distribution of input-output data that is desirable for learning, possibly a ground truth. Under this definition, the fundamental notion underlying ML is that any ML model can only observe and learn from an n_i-sized sample of a domain indexed by i, i.e., $(x_k, y_k)_{k=1}^{n_i}$, although whether the outputs are accessible to the model depends on training or deployment. Additionally, given a specific task, an input x and an output y drawn from this domain exist within prescribed spaces $\mathbb{X}$ and $\mathbb{Y}$, respectively. Now, DL typically relies on the i.i.d. assumption, where the joint probability distribution of sampling (x_k, y_k) is static, i.e., the samples arise from the same memoryless generative process. In essence, this is what makes an inductive model trained on sample $\mathcal{S}_1$ applicable to sample $\mathcal{S}_2$; it is hoped that their corresponding domains, indexed by 1 and 2, are identical. Of course, the i.i.d. assumption may be flawed to begin with [108, 317], but, regardless, any significant deviation from these statistics can challenge the validity of a trained model. We emphasize the adjective "significant" here, as anomaly detection is its own topic of research [38].

Naturally, the stationary assumption can be broken in many ways, which have been surveyed extensively elsewhere [87]. The Bayes theorem suggests a few possibilities, especially when written as:

$$P(x \cap y) = P(x|y)P(y) = P(y|x)P(x), \tag{1}$$

and these are listed in Table 6. Prior and covariate shift – variations in $P(y)$ and $P(x)$, respectively, between domain samples $\mathcal{S}_1$ and $\mathcal{S}_2$ – are types of data drift relating to sampling representation. This can be a problem, given that predictive DL models strive to determine the conditional probability $P(y|x)$ for all $x \in \mathbb{X}$ and $y \in \mathbb{Y}$; the accuracy of such an estimate suffers for subspaces in $\mathbb{X}$ and $\mathbb{Y}$ that have not previously been encountered with sufficient frequency. Nonetheless, on its own, these kind of changes in a data environment do not negate the utility of an $\mathcal{S}_1$-trained model deployed on $\mathcal{S}_2$. Sometimes, they simply suggest concept-space interpolations/extrapolations need to be tightened. In essence, the two domains may still be identical. On the other hand, concept drift is a scenario where $P(y|x)$ itself changes between domain samples $\mathcal{S}_1$ and $\mathcal{S}_2$, e.g., in the form of changing classification boundaries. Sometimes concept drift even overlaps with the notion of task redefinition, where the label space $\mathbb{Y}$ may change. In such a case, extensive model retraining appears almost inevitable.

For all such scenarios, it is the responsibility of a prospective AutoDL monitoring mechanism to identify when model performance may suffer, so as to trigger a maintenance procedure. The monitoring may be directly reactive, assessing the degradation of a loss metric, or be indirectly

Table 6. Scenarios in a dynamic data environment.

Scenario	Changes $(\mathcal{S}_1 \rightarrow \mathcal{S}_2)$
The i.i.d. Assumption	None
Covariate Shift	$P(x)$
Prior Shift	$P(y)$
Concept Drift	$P(y|x)$
Task Redefinition	$\mathbb{X}$ and/or $\mathbb{Y}$

Set $\mathcal{S}_i$ is defined here as a collection of data that samples an input-output distribution of interest to a DL model, i.e., a domain indexed by i. Input x and output label y belong to input and output spaces $\mathbb{X}$ and $\mathbb{Y}$, respectively, while P denotes a probability with respect to sampling representation.

Table 7. The most common approaches underlying adaptive strategies for AutoDL.

Maintenance Approach	Domain Constraints ($\mathcal{S}_1 \to \mathcal{S}_2$)	Data Availability for Maintenance ($\mathcal{S}_2$)	Notes
Continuous Learning	None	Any	Constantly processes a data stream. Can combine with other approaches.
Domain Adaptation	$\mathbb{Y}_1 = \mathbb{Y}_2$	x: ✓, y: ✗	
Domain Generalization	$\mathbb{Y}_1 = \mathbb{Y}_2$	x: ✗, y: ✗	Often generalizes more than one domain.
Few-shot Learning	$\mathbb{Y}_1 \neq \mathbb{Y}_2$	x: ✓, y: ✓	Prioritizes minimal sampling of $\mathcal{S}_2$.
Zero-shot Learning	$\mathbb{Y}_1 \neq \mathbb{Y}_2$	x: ✗, y: ✓	

Sample $\mathcal{S}_i$ represents a domain indexed by i, where $\mathbb{Y}_i$ is its corresponding output/label space. The inputs and output labels within sample data are represented by x and y, respectively. Domain adaptation/generalization is well-suited for statistical shifts and concept drifts, given a fixed task, while low-shot learning quickly adapts to new classes, a consequence of task redefinition. Note that there are proposed variants to these approaches, e.g., differing degrees of label-based supervision from $\mathcal{S}_2$.

preemptive, examining non-stationary statistics of data. In fact, there are many drift detection mechanisms and strategies in existence, some of which have already been used in the context of adaptive AutoML [37, 140]. We do not discuss these in depth, as, for now, the field of AutoDL has not yet embraced the idea of agile monitoring. However, as a compromise, there have been several past efforts to acknowledge diverse domains and fold their identification into a DL model. For example, the simplest form of the long-established adaptive resonance theory (ART) [97] dynamically sizes a layer of "recognition neurons" to cluster encountered data into distinct categories. Far more recently, research efforts have merged a domain classifier into a DNN [88] and have explored domain encodings via a so-called Memory-based Parameter Adaptation (MbPA) method [254]. While the reactive potential is limited in both these modern examples to the memory space reserved for domains, at least data drifts between those domains can easily be adapted to.

8.3 The Current Paradigms of Maintenance

The vast majority of DL models are monolithic and do not contain specialized domain-memorization structures. Thus, in most cases, the standard form of maintenance is simply retraining on a newly encountered domain. Doing so from scratch, however, is not ideal, especially in the computationally expensive context of DL. Consequently, there has been an increasing focus on developing efficient maintenance strategies for various scenarios. The most common principles underlying such approaches are listed out in Table 7.

First of all, if domains represented by $\mathcal{S}_1$ and $\mathcal{S}_2$ differ in statistics but not in $\mathbb{Y}$, the topics of domain adaptation and domain generalization apply [18, 271]. An example of this is extending an image classifier of birds trained on sketches and cartoons to a more photorealistic domain. In the case of domain adaptation, maintenance occurs reactively, with the model able to learn from newly encountered data that is possibly even labeled. Domain generalization covers more preemptive strategies, improving model adaptability without encountering what it will have to adapt to. Elsewhere in the DL field, low-shot learning attempts to deal with more dramatic task redefinitions, i.e., where label space $\mathbb{Y}$ changes. In the extreme, these endeavors may form a prerequisite towards eventual general intelligence, but, in current practice, the tasks applied to differing domains sampled by $\mathcal{S}_1$ and $\mathcal{S}_2$ are closely related, e.g., both are image recognition problems using CNNs. For few-shot learning [82, 173, 213, 249], maintenance mechanisms do have access to limited data from the new domain, including for unseen classes, and adapt accordingly. For

zero-shot learning [172, 222, 286], an AutoDL system does not expect to encounter any examples of unseen classes, although auxiliary information about these classes is leveraged instead, e.g., information based on attributes or embedding similarity.

In any case, for many reasons described earlier, these updating strategies are often triggered manually and applied offline. However, should the priority focus of the DL community move to AutonoDL, these approaches are well-suited to being appropriated for online adaptation. Thus, it is still worth highlighting the most common paradigm that underpins DL efforts in this space: the meta-model.

Meta-learning [22, 233, 234, 236] in the context of AutoML has been reviewed elsewhere [140], but, in AutoDL, it refers to identifying some similarity, often in terms of "meta-features", between a new domain/task and an old one; this allows previous knowledge to be leveraged while optimizing a DL model for a new environment, not too unlike transfer learning. A meta-model can then be seen as but a high-level context-aware recommendation system. This idea dates back to the year 1987 [234], although, at that time, the application was relatively simple and small-scale. In modern times, it is often a DNN of its own, thus being optimizable via gradient descent [9, 293], RL [158], evolutionary approaches [118, 127], etc. These meta-models are usually trained in offline mode, experiencing many domains/tasks – debate endures around just how much is needed [7, 90] – and they ideally learn which recommendations are optimal for training a base model in each setting. They can also be developed in online mode, alternating in updates with a base DL model [292, 293], but this compounds the risks of learning-based instability.

Importantly, meta-models are highly varied in their usage. For instance, there is a strand of DL research popularly known as "learning to learn" [113, 238], which is motivated by certain questions: when is Adam better than standard SGD? What is the quickest way to train a model for sparse data? And so on. In such a scenario, the meta-model – typically an RNN such as an LSTM – learns correlations between diverse operating contexts and best values for select variables relating strictly to optimization procedures [66], i.e., algorithmic hyperparameters, that can then boost model development in new domains. Of course, meta-model input-output details vary widely across research efforts, whether the approach is applied to standard optimizers [9] or deep RL [67].

Notably, if there is one area of DL that does focus heavily on online operations and adapting to new domains/tasks, it is in fact deep RL. This approach is heavily favored in the AI sub-field of "general game playing", and many related research works are tested in diverse contexts, such as on collections of Atari 2600 video games [268, 293]. As an example of meta-learning for deep RL, one publication [118] focuses on the fact that, in RL, there is often no intrinsic relation between a task objective and a loss/reward function. It thus applies a context-aware temporal CNN to optimize this loss function so that an RL agent, the base DL model, is able to learn a task with maximal efficiency. In this particular approach, the meta-model is closely integrated with the base model, influencing it via backpropagation. Elsewhere, adaptation of the return function has similarly been explored with a gradient-based meta-learning algorithm, tuning hyperparameters, such as discount factor and a bootstrapping parameter, in online fashion [293]. This particular approach, using "meta-gradients", has partially inspired subsequent work, such as an effort to have RL agents seek out useful questions, in general value function (GVF) format, that, when answered, optimally support their learning process [268].

It is now clear that, should an AutoDL monitoring system throw an alert that a DL model needs to be updated, prior experience can definitely accelerate model adaptation. However, it is often difficult to assess the quality of proposed meta-models and meta-learning algorithms. Many works are based on handcrafted update rules targeting select hyperparameters, such SGD learning rate [257], SGD weight decay [66], the decay factor in RL [293], etc. Others gradually move further and further to full automation, leveraging neural networks to simulate the loss function [118, 143], the target

Table 8. Evaluative assessment for trends in automated maintenance.

	Novelty	Solution	Effic.	Stability	Interp.	Reprod.	Engi.	Scalability	General.	Eco.
Domain Adaptation	3 (≈30)	?	High	?	?	?	?	High	Mixed	?
Low-shot Learning	3 (≈29)	?	High	?	?	?	?	High	Mixed	?
Continuous Learning	5 (≈30)	?	High	?	?	?	?	High	Mixed	?

Each row marks an emergent trend in AutoDL, specifically automated maintenance. Each column marks a criterion – see Section 2.4 – by which the trend is assessed. The evaluations are mostly qualitative, averaged across the most significant works researching the trend. Where a graded value is not provided, "?" indicates that a rigorous assessment is not achievable without more research works to analyze. Novelty denotes years since seminal works in DL (ML) were published. Abbreviations are: "Solution" for Solution Quality, "Effic." for Efficiency, "Interp." for Interpretability, "Reprod." for Reproducibility, "Engi." for Engineering Quality, "General." for Generalizability, and "Eco." for Eco-friendliness.

value to maximize in RL [292], the delta of weights [9, 67], etc. So, how much of meta-model design should rely on humans, producing limited but decent strategies? How much should be fully automated, producing generic but potentially unwieldy mechanisms? These are open questions, as is whether and in which cases meta-learning is actually effective [7, 90, 195].

Ultimately, we emphasize that meta-learning is not the same thing as continuous learning, the latter of which is a fundamental requirement of AutonoDL. Meta-learning is a principle that can be applied across the entire DL workflow in Figure 1, so as to leverage previous experience in speeding up the development/deployment process. Similarly, there are ways to react intelligently to new domains that have nothing to do with typical notions of prior knowledge. Nonetheless, this is where AutoDL presently sits on the automated-maintenance front; it is a nascent exploratory topic with little benchmarking beyond an assessment of dynamic hyperparameter control [71].

8.4 Overview

Given how adaptive ML, namely AutonoML, has only just become an emerging thread of research [140], it is of no surprise that there is little consolidated focus in the literature on automated mechanisms for the maintenance of DL models, let alone an AutonoDL field. Indeed, Section 8 has instead discussed the most promising approaches, as listed in Table 8, that are likely to inform adaptation practices in the future. This connection is immediate in the case of continuous learning, and the paucity of research in this area merely reflects the challenges of constantly updating a DNN, but domain adaptation and low-shot learning are often not considered by the DL community in terms of fully automated model maintenance, instead being associated with tackling new DL problems; related publications often miss or ignore the automation aspect in their commentaries. Accordingly, the evaluative assessment in Table 8 is necessarily replete with unknowns. This is the case even though the theoretical foundations for the listed trends stretch back far beyond the DL era, e.g., low-shot learning being considered in the 1990s [235]. Ultimately, it is just not possible at this time to gauge how effectively an approach counters diminishing model accuracy, let alone whether such claims are reproducible over a representative benchmark of dynamic DL problems.

So, if theoretical analyses are limited, making it difficult to comment on stability, and there are no widely adopted implementations of such mechanisms, restricting any eco-friendliness assessment, can anything be surmised about automated maintenance in AutoDL? Well, the most common methodologies employed for domain adaptation, together with low-shot learning and continuous learning, are gradient-based, which means that such approaches can be considered relatively efficient ways to adapt a model. Likewise, they are highly scalable. As for generalizability, this depends on the specific methods used for each form of maintenance. Some are specifically designed for updating a specific network component such as a model head, i.e., the classification layer, and thus may struggle when applied to novel search spaces. Beyond this, one must reserve judgment as

to how the automated maintenance endeavor will evolve. There are certainly promising research directions, but serious consideration of AutonoDL as a field will likely be contingent on further advances in computational hardware.

9 CRITICAL DISCUSSION AND FUTURE DIRECTIONS

It is undeniable that, at present, the topic of DL continues to attract an unparalleled degree of attention within computer science communities, and its successes have spilled over into the sub-field of AutoDL. This monograph has attempted to take a sample of that research, representing the most significant trends in the area, and categorically systematize it with respect to the simple but encompassing DL workflow depicted in Figure 1. However, this has not been a trivial task; the sheer quantity of publications in AutoDL and related sub-fields can be considered daunting. Certainly, popularity has positives, as a critical mass of attention is required to drive progress in a topic, but the negatives are just as evident, risking both a mob-mentality 'clumping' around certain trends – these endeavors may or may not have been exhausted of promise – and a level of low-quality publication 'noise' that obscures possible leads for future advancement.

To combat the counterproductive dangers of research oversaturation, this work has emphasized the need to examine AutoDL research holistically, suggesting a broad set of criteria in Section 2.4 that may be used for evaluation. Granted, this review can only provide a limited assessment under such a framework, or similar, without broader adoption and contemplation by the DL community. Thus, Section 9.1 promotes a practical questionnaire for AutoDL researchers to use, based on the same criteria, when evaluating how their proposed methodologies/experiments fit within the broader context. It is not intended as a rigid document, serving more as a kick-starter for conversation around the topic.

Finally, still guided by the assessment criteria, we consolidate all the assessments within the previous sections, each related to a single stage of a DL workflow, into an overarching overview of the AutoDL field, presented in Section 9.2. This high-level perspective enables subsequent commentary in Section 9.3 on what this review finds the most significant challenges and greatest opportunities to be in terms of future directions.

9.1 A Proposed Questionnaire for Self-assessment

While the full scope of AutoDL research is expansive, especially beyond the bulk of work that exists within the topic of NAS, there are clear commonalities across the field. Each investigation, whether theoretical or experimental, typically seeks to develop a high-level mechanized process that efficiently contributes to the existence of a "good", ideally persistent, DL model. Accordingly, as Table 1 indicated, these attempts can often be characterized as optimizations, regardless of the phase of a DL workflow in which they operate. This also means that it is possible to construct a shared baseline framework by which all AutoDL research can be assessed, which is the purpose of the ten criteria introduced in Section 2.4.

Naturally, a unified assessment framework is appealing for several reasons:

- It helps researchers/developers to be comprehensive in understanding the strengths and limitations of their approach, which can then lead to subsequent improvements.
- It simplifies peer review, supporting the accelerated provision of accurate feedback.
- It enables the contextualization of an individual algorithm within the entirety of AutoDL, promoting informative high-level perspectives of the field.

Granted, critics may claim that a one-size-fits-all framework does not appreciate nuances, and this is a reasonable concern, but, given the state of mass publications in the field, it is arguably a

higher priority of consideration to encourage improved rigor through standardization, at least at the current time.

So, how would a researcher go about engaging with the criteria listed in this monograph? To answer this, we provide an example form in Table 9 and Table 10, filling out a series of questions as if they were asked of two seminal publications, NASNet [325] and DARTS [171], which we stress have been selected for their importance to AutoDL, not for any perceived deficiency.

Now, crucially, because this form represents a deeper dive than the quantitative research-trend overviews described in Section 2.4, there are certain nuances to consider. For instance, one significant difference alluded to in the earlier section is that a detailed assessment should capture the quality of both an AutoDL process and the target DL model that it impacts. Accordingly, several criteria in the example form split their questions in such a manner. As to the rationale behind listed questions, this matter is elaborated on in the following discussion.

I. Novelty is a crucial prerequisite for the publication of any research in academia. For the most part then, this criterion does not need to be spelled out explicitly, as a strong peer-review system will control for its quality automatically. That stated, a recurring grievance in data science and beyond is that authors will often promote the novelty of their work, as justified within a narrow scope, without awareness of what exists across related disciplines or further back in academic history [248]. This limited perspective can be even more extreme. For example, a gradient-based method employed in NAS may solely be compared against other gradient-based NAS methodologies without appreciating the possible existence of associated studies in HPO. So, whereas an author may argue that the methodology is entirely new, the novelty may technically be its application to a search space of network architecture rather than a search space of training-algorithm hyperparameters. Of course, it would be ideal for AutoDL researchers to review parallel work in the space of AutoML/AutonoML and even further afield, but, given the eventual goal of developing an integrated end-to-end AutoDL system, one should at least be cognizant of similar approaches across all phases of the DL workflow in Figure 1. The questions in Table 9 reflect this bare minimum.

II. Solution quality refers primarily to the "correctness" of a DL model, which is traditionally the predominant focus of academic research; it is the obvious criterion to include and needs no in-depth justification. However, it must be noted that there are many different metrics by which this quality may be gauged, often depending on the DL problem that the model attempts to solve. For instance, accuracy often refers to classification tasks, mean average precision (mAP) is used for object detection, and so on. Even for the same type of task, a change in context may prioritize a different objective function. For example, maximizing top-1 accuracy may be required for high-stakes predictive analytics, e.g., skin disease classification, while top-n accuracy for $n \neq 1$ may be fine for other recommendation systems, e.g., preference prediction involving musical genres.

So, while the matter of supporting future comparisons, i.e., reproducibility, will be addressed shortly, researchers are still advised to be diligent when making their own comparisons with previous SoTA approaches. Indeed, the notion of "ceteris paribus", i.e., all other things being equal, is the ideal when assessing model correctness resulting from two competing AutoDL algorithms. Of course, some forms of equivalence, or lack thereof, are trivially clear; AutoDL algorithms intended for different phases of a DL workflow will have different aims. For instance, NAS and HPO works construct DL models from scratch and thus focus on their baseline quality, while data-engineering and deployment methods are best judged on how they modify the correctness of existing models. Similarly, although automated maintenance is complicated by a time dimension that makes it difficult to pick a representative solution-quality metric, related works still have the common aim of minimizing model deterioration.

Regardless, where obvious and not-so-obvious, the questions in Table 9 urge AutoDL researchers to make model-correctness claims with an eye to both equivalence and fairness, detailing evaluation

Table 9. [Part-1/2] An example self-assessment questionnaire for AutoDL algorithms.

	Question	Example response for NASNet [325]	Example response for DARTS [171]
I: Novelty	Is there significant innovation in this work?	Yes; the first cell-based search space in NAS.	Yes; the first differentiable search algorithm in NAS.
	Is this work completely distinct from existing research in Automated Data Engineering?	Yes; within reasonable scope.	Yes; within reasonable scope.
	Is this work completely distinct from existing research in NAS?	No, but without issue; reuses the search space in [325].	No, but without issue; reuses the search method in [324].
	Is this work completely distinct from existing research in HPO?	No, but without issue; the search method has been studied in HPO.	No, but without issue; the search method has been studied in HPO.
	Is this work completely distinct from existing research in Automated Deployment?	Yes; within reasonable scope.	Yes; within reasonable scope.
	Is this work completely distinct from existing research in Automated Maintenance?	Yes; within reasonable scope.	Yes; within reasonable scope.
II: Solution Quality	Does the contribution of this AutoDL method to model accuracy (or similar metric) significantly improve upon equivalent SoTA approaches?	No, but without issue; the largest NASNet model (of the time) achieves 82.7% top-1 accuracy on ImageNet, which is similar to previous SoTA model but with 40% reduction in FLOPs.	No; the best DARTS model achieves 2.76% error on CIFAR-10 with 3.3 million parameters, which is slightly worse than previous SoTA models.
	Is the reported contribution to accuracy (or similar metric) fairly representative?	Yes, decently; reported for CIFAR-10, ImageNet, COCO.	Yes, decently; reported for CIFAR-10, ImageNet, PTB, WikiText-2.
III: Efficiency	Does this AutoDL method converge with significant speed?	No; 2000 NVidia P100-hours to complete NAS on CIFAR-10.	Yes; 4 NVIDIA GTX 1080Ti GPU-days to complete NAS on CIFAR-10.
	Does the contribution of this AutoDL method to model efficiency (i.e., minimization of computational-resource usage) significantly improve upon equivalent SoTA approaches?	Yes; this NAS method can produce a model with 3.3 million parameters that achieves 2.65% error on CIFAR-10, but latency is not evaluated.	Yes; this NAS method can produce a model with 3.3 million parameters that achieves 2.76% error on CIFAR-10, but latency is not evaluated.
	Are the theoretical/practical efficiencies fairly analyzed?	Yes, mostly; the number of model parameters and FLOPs are analyzed, although there is no comment on practical speed.	Yes, mostly; the number of model parameters and FLOPs are analyzed, although there is no comment on practical speed.
IV: Stability	Do repeated runs of this AutoDL method produce results (e.g., in terms of accuracy metric) with minimal variance?	Unknown; analysis does not exist.	Yes, moderately; the results of four search runs are reported.
	Do repeated runs of this AutoDL method impact model structure (if applicable) in the same way?	Unknown; although NAS methods intrinsically influence model structure by virtue of construction, analysis does not exist.	Yes; different search runs may produce different models structures.
	Is the AutoDL method stable (i.e., not sensitive) for certain values of AutoDL-method hyperparameters?	Unknown; only brief mention that a learning rate of 3.5e-4 is relatively stable.	Unknown; analysis does not exist.
V: Interpretability	Is this AutoDL method explainable in how it impacts the target model?	Unknown; analysis does not exist.	Yes, decently; analyzed under the differentiable bi-level optimization framework.
	Are the results produced by the target model (after the AutoDL method takes effect) explainable?	Unknown; analysis does not exist.	Unknown; analysis does not exist.

Ideally, AutoDL algorithms should be evaluated w.r.t. all ten criteria. If a publication answers with a high quantity of thorough "yes" and justified "no" responses, it can be considered high-quality research with comprehensive consideration of all relevant issues in AutoDL. Seminal NAS works, NASNet [325] and DARTS [171], are used to exemplify usage of this questionnaire. All comparisons are made against the available literature of the time, i.e., July 2017 and June 2018. We note that the questions are neither exhaustive nor final.

contexts in depth. Far too often, incremental improvements are claimed as SoTA – one may speculate as to the reasons – without strong justification. Alternatively, superior model accuracies may not be adequately accompanied by an emphasis on the trade-offs involved. In actuality, it is not a great weakness for an AutoDL algorithm to lack accuracy improvements over SoTA models if the benefits of the technique are to be found elsewhere. The NASNet publication exemplifies this in Table 9, proving far more impressive in reaching similar-to-SoTA accuracy values via the construction of much smaller DNNs than contemporary hand-crafted models.

Table 10. [Part-2/2] An example self-assessment questionnaire for AutoDL algorithms.

	Question	Example response for NASNet [325]	Example response for DARTS [171]
VI: Reproducibility	Is all relevant experimental code involving this AutoDL method publicly available?	No.	Yes; links can be found in the paper.
	Is all relevant experimental code involving the models resulting from this AutoDL method publicly available?	Partially; the model definition and pre-trained weights can be found, but lack instructions on how to reproduce them.	Yes; at github.com/quark0/darts.
	Are all relevant experimental information for AutoDL-method and resulting models (e.g., hyperparameters, random seed, hardware, or platform) reported in the paper/code?	Yes, decently; most hyperparameters are mentioned, and hardware and platform are mentioned.	Yes; most information can be found in either paper or the released code.
VII: Engineering Quality	Is this AutoDL method implemented for public use (e.g., off-the-shelf scripts)?	Minimally; there are only two template scripts for evaluation usage.	Yes; the search, training, evaluation scripts can be found.
	Is the implementation code commented?	Partially.	Rarely.
	Is the implementation code tested?	Yes, mostly; however, there is a lack of detailed test reports.	No; lacks unit tests.
	Is the implementation code documented?	Partially; model evaluation is documented.	No.
	Is the implementation code modular, reusable, and extendable?	Yes; different config files can instantiate different versions of NASNet.	Yes; the minimal search cell can be reused.
VIII: Scalability	Are the theoretical/practical computational costs of this AutoDL method analyzed?	Yes, partially; empirical costs are analyzed, theoretical costs are not.	Yes, partially; empirical costs are analyzed, theoretical costs are not.
	Is it feasible to scale up to large-scale datasets?	No, but issue considered; sub-networks (i.e., cells) discovered for small-scale datasets can be transferred to large-scale datasets.	Unknown; the AutoDL method is efficient and is ideally able to scale up to ImageNet, but analysis does not exist.
	Is it feasible to scale up to large-scale models?	Unknown; analysis does not exist.	Unknown; the AutoDL method is efficient and is ideally able to scale up to ImageNet-scale models, but analysis does not exist.
	Is it feasible to scale up to large-scale search spaces?	Unknown; analysis does not exist.	Unknown; analysis does not exist.
IX: Generalizability	Can this AutoDL method generalize well to unseen datasets?	Yes; from CIFAR to ImageNet and COCO.	Yes; from CIFAR to ImageNet, and from PTB to WikiText-2.
	Can this AutoDL method generalize well to different target models?	Yes; can generalize to different macro structures.	Yes; can generalize to different macro structures.
	Can this AutoDL method generalize well to different kinds of search spaces?	No, but without issue; the proposed NASNet search space is intrinsic to the NAS algorithm.	Unknown; analysis does not exist.
X: Eco-friendliness	Is the energy consumption minimal for this AutoDL method?	Unknown; analysis does not exist.	Unknown; analysis does not exist.
	Is the energy consumption minimal for the target model when actively employed?	Unknown; analysis does not exist.	Unknown; analysis does not exist.
	Is the potential environmental impact of this AutoDL method (e.g., converted to carbon emissions) minimized?	Unknown; analysis does not exist.	Unknown; analysis does not exist.

Ideally, AutoDL algorithms should be evaluated w.r.t. all ten criteria. Please follow the same notation as Table 9 to read this table.

III. Efficiency is arguably the next highest priority for the AutoDL community after model correctness, emphasizing how most SoTA DNNs can be considered resource heavyweights, even in the current era of hardware. Generally, there are two primary considerations in this category: speed and memory. In the case of an AutoDL algorithm, these two issues are represented in how long an AutoDL algorithm takes to converge upon its solution, how much run-time memory is used, how many accelerators are required, and so on. Model training time and memory footprint are included in this assessment if part of an AutoDL process, e.g., for NAS, HPO, deployment-aware AutoDL, and certain forms of automated maintenance. However, some AutoDL algorithms are also responsible for inducing efficiency in a DL model, whether constructed from scratch or already in existence. In the latter case, this may not even involve any modifications to the direct structure of a DNN; intelligent exploitation in the category of automated data engineering can

boost inference speed by optimally managing datastreams. Whatever the case, the questionnaire in Table 9 motivates assessing the efficiency of both the AutoDL process and, where applicable, the DL outcome. Researchers will note that the speed and compactness of a DL solution often relates to (1) its number of parameters, (2) the memory footprint related to its activations, (3) its theoretical FLOPs usage, and (4) its practical inference speed.

Finally, we reiterate what was stated when discussing the correctness of a DL model; equivalence is key. Hardware and development/deployment environments should be diligently detailed to make any SoTA efficiency claims. For the same consideration, theoretical analyses and practical reporting are both beneficial. Only with a fair assessment does it become clear where, in relation to both correctness and efficiency, the Pareto front for AutoDL algorithms exists.

IV. Stability, predominantly of model correctness but also of any efficiency metric, is the first criterion that, as of the early 2020s, is usually overlooked by the AutoDL community. This is not simply due to negligence either, at least when granted the benefit of the doubt. Given how computationally expensive it is for even a single AutoDL run, it can often seem infeasible to apply the bare minimum commonly required for a stability assessment, i.e., a statistical sampling of performance followed by a report on associated variances. Nonetheless, the field of AutoDL suffers from this lack of statistical rigor, so much so that conclusions on SoTA performance have been challenged [160]. Metric-based analysis is thus recommended, where possible, to provide confidence in the reporting of any results. In the same vein, there are other more advanced analyses that can further inspire trust in published research. For instance, AutoDL algorithms that impact model structure, e.g., those related to NAS, may produce dramatically different networks. Stability in this case becomes even more complex to assess if those models are associated with indistinguishable performances, sometimes called "the Rashomon effect" [28]. In such a scenario, it is worth knowing whether the AutoDL algorithm in question is truly effective at optimization or whether the DL problem it is employed upon is similarly solvable for any network

Finally, the questionnaire in Table 9 also includes a higher-level question related to the hyperparameters of an AutoDL approach, although we avoid that terminology so as to avoid confusion with HPO. The point here is that the stability of an AutoDL algorithm may depend on both stochastic or deterministic elements, and it is worth assessing how sensitive the approach is to the latter, namely the settings selected for AutoML-method parameters.

V. Interpretability is one of the newer criteria that the AutoDL community has had to consider, reflecting the permeation of DL into broader society. It is no longer seen as acceptable in a variety of high-stakes settings for a DL model to hide its decision-making process within the complexity of its network, even if that complexity has often been associated with the performative strengths of DNNs. Admittedly, enforcing interpretability within a model is much more complex than acknowledging its value, and the topic is thus relatively nascent in DL [98]. Nonetheless, these requirements and associated efforts to address them have similarly diffused into AutoDL [26]. Thus, for this criterion, researchers should question whether the impact of an AutoDL algorithm and, if relevant, the DL model that it targets are both explainable. Certain theoretical analyses can also bolster associated commentary, such as the provision of ablation studies.

VI. Reproducibility is a core principle underlying the scientific method. Its inconsistent treatment has been a common concern in general ML for a while [207], and these concerns have likewise been mirrored in the field of AutoDL [160]. After all, claims that an AutoDL algorithm is accurate or efficient are pragmatically useless if they can never be replicated. Granted, while the criterion of solution quality requires a researcher to be diligent in comparing their method to those that have come before, and the criterion of stability arguably relates to assessing repeatability within the confines of the same experiment, the onus of confirming reproducibility belongs to the works that come after. Nonetheless, there are plenty of ways that a publication can support an ethos of

reproducibility, and most of these come down to a sufficient provision of details. Accordingly, the questions in Table 10 are motivated by considerations discussed elsewhere [164]. They essentially urge a researcher to consider whether they have provided comprehensive parameter values and ideally code for any presented experiments. This goes for both the AutoDL process and the DL model that it targets. So, while the NASNet publication assessed as an example does reasonably well, the lack of publicly available scripts around the operation of the NAS process is a weakness, even though it is understandable for a proof-of-principle work to focus on the end-product DL model as evidence of a well-performing AutoDL algorithm. In any case, researchers should also consider detailing the development/deployment environments that the experiments are run within.

VII. Engineering quality is a criterion that reflects the translation of AutoDL algorithms from theory to application. While reproducibility concerns code that can be used to verify the validity of certain claims, this criterion considers whether code exists that is usable beyond the contexts that were explored. Of course, if some form of implementation exists, experts with enough skill may eventually be able to hack it for their purposes after expending enough time and energy. However, given that the field of AutoDL is a proponent of automation, the worthiness of an algorithm can be judged, along one axis, by how accessible its implementation is to non-experts. How this human-computer interaction (HCI) should be managed and whether it should be allowed at all in certain contexts are topics for a separate discussion. Here, published codebases score well if they are informative, both internally via comments and externally via documentation, e.g., user guides. The codebase should also be robust and reliable, i.e., well-tested, and, ideally, there should be some degree of modularity that facilitates plug-and-play extension. After all, if a long-term goal of the AutoDL field is to have a fully integrated end-to-end system, the standard approach for achieving this is by fashioning together distinct pre-existing mechanisms that focus on different phases of a DL workflow.

VIII. Scalability is the first measure of extensibility promoted by this monograph, denoting how versatile an AutoDL approach is for standard variations of its default usage. Certainly, for proof-of-principle research, it is usually fine to test the operation of an algorithm on a small dataset, a small target DL model, and a small search space. However, real-world scenarios can face numerous datastreams to process, an extreme depth of network to capture all relevant features, a multitude of adaptive mechanisms to select from, and so on. Thus, it is worth asking whether an AutoDL mechanism is realistically suitable for scaling up in terms of data, model, and search space. Given that this is generally a matter of efficiency, Table 10 questions whether computational costs have been analyzed, e.g., formalized in "Big O" notation.

IX. Generalizability is the second measure of extensibility promoted by this monograph, denoting how versatile an AutoDL approach is for non-standard variations of its default usage. It mostly mirrors the criterion of scalability in that the focus is on dataset, target model, and the search space relevant to the algorithm. However, this time the proposed questions are more open-ended, gauging across these three aspects how broadly applicable a proposed approach is. There is of course no issue with an AutoDL method being specialized, e.g., the NASNet example in Table 10 is intrinsically wedded to a particular search space. Nonetheless, this criterion acknowledges that there is always value in general-purpose mechanisms, assuming the trade-offs, usually efficiency, are reasonable.

X. Eco-friendliness, as the final criterion suggested in this review, is a very new topic within data science, let alone AutoDL [201], that reflects the current concerns of global society. While traditionally ignored for the most part, the evolution of computational hardware has made this a much more pressing consideration; greater processing capabilities have not been able to dissociate from greater energy expenditures. Certainly, this is a problem as of the early 2020s due to the world relying heavily on non-renewable resources for power generation. Unnecessary energy usage

exacerbates the current global warming crisis via carbon emissions. Moreover, these effects are argued to become non-negligible once large-scale SoTA models are being trained and operated across extensive computer clusters by many developers and users, respectively. However, to be fair, a genuine assessment of this criterion is complex and comes down to trade-offs. Any negative evaluation of a target DL model should also not be doubled up for the AutoDL process, and vice versa. While we elaborate on this discussion in Section 9.2, it is sufficient here to recommend that researchers attempt separate power-usage records for the AutoDL algorithm and the target DL model. Estimating environmental impact is also worthwhile, even if simply applying a quick power-to-emission conversion based on a cited source.

In summary, this section has discussed the rationale behind ten criteria and associated example questions, in Table 9 and Table 10, that are designed to foster a comprehensive evaluation of an AutoDL algorithm, especially when considering its place within the field.

As a final point, the criteria have been ordered to reflect the mental process a reader may have when newly encountering an AutoDL publication. Such a sequence of thoughts may be as follows: "Have I seen this stuff in the abstract before? No? Ok, I will skim the paper. Does the proposed approach do a good job? Yes? Well, does it take forever to run? No? Ok, I will read further. Does it do its job consistently? Yes? Well, can I understand how the job is being done? Still yes? Seems too good to be true. I would not mind testing the results myself. Can I do this easily? Yes? I am impressed. Well, is there an implementation that I can use for my own applications? Yes? Can I scale it up? Yes? Can I use it for something else? Yes? My power bill (and the world) will survive this, right? Yes? Wow. This truly is an amazing AutoDL algorithm."

9.2 An Overarching Evaluation of AutoDL

Having dived deep into the criteria that form the basis of the assessment framework within this review, this section now zooms out once more for a final high-level perspective of AutoDL as of the early 2020s. Many of the following discussion points, again sorted by criterion, arise when considering all the overviews within the previous sections in an aggregate manner.

But first, it is worth a reminder that, with respect to the workflow-based reconceptualization of AutoDL in Figure 2, around three-quarters of surveyed work relates to model development, and the majority of this concerns NAS specifically. In essence, and as with AutoML [140], there certainly are intensifying conversations around just what the scope and long-term goals of AutoDL are, given its organic and unplanned evolution, but it must be acknowledged that the field is still dominated by a single sub-category. The AutoDL story is primarily a NAS story. At the same time, it is arguable that, because the other phases of the DL workflow are less well-explored, they may be more fertile for ongoing progress. With that context addressed, we now assess AutoDL through the lenses of particular criteria.

I. Novelty. An obvious take-away from Table 1, which lists seminal works in AutoDL, is that, excluding outliers, the field only truly became consolidated within the mid-to-late 2010s. Additionally, although the scattering of research makes conclusions difficult, it is reasonable to claim that NAS and DL-related HPO started up in about 2016, while the extension of the automation ethos into the data-engineering and deployment phases followed in around 2017/2018. As for automated maintenance, there has definitely been adaptation experimentation performed in parallel, but this research thread is not necessarily as advanced as the others; we are not aware of any DL model-adaptation mechanism that has the same standard of applicability.

Unsurprisingly though, elements of all AutoDL research are often inspired by previous work, some of it even external to the field. This is most obvious when the automated approaches in Table 1 are treated as computationally processed optimization problems, dissected into search space, search strategy, and boosting method for candidate evaluation. It follows that the foundations of all search

strategies are well established, having been utilized for decades with the broader ML sphere, as indicated by Table 4. This means that, details aside, there is a convenient familiarity when working with, for example, gradient-based methods in both NAS [63, 171] and DL-based HPO.

Search spaces, on the other hand, are far more diverse, differing even in the abstract between various phases of a DL workflow, if not sub-categories. For instance, automated data augmentation may sample sets of transformations, hardware search might optimize against series of programmable gates, and setting up low-shot learning for quick automated adaptation might have a search space of parameter-update rules. Given the importance of good search space design for optimal performance, with the possible exception of strategy-focused HPO, a lot of innovations often arise in this facet of AutoDL research. Granted, in research threads that have sufficient publication density, certain seminal works set the standards for such design, e.g., NASNet-like spaces [325], etc. These are usually tweaked by subsequent proposals rather than entirely replaced. Nonetheless, there is plenty of experimentation in the aspect of search spaces, even for NAS, and breakthroughs that improve the quality of AutoDL algorithms are possible.

Tactics for managing candidate evaluations are also a prime target for innovation, almost entirely driven by a focus on efficiency. Indeed, search spaces may differ across numerous scenarios, but almost every type of AutoDL algorithm needs to evaluate its target DL model after modifying the DL process in whatever way it does, otherwise there is no objective function to improve against. The obvious exception to this rule is filter-type data engineering, where features are transformed to optimize inherent properties of a dataset without involving the model in any way. However, in general, the point stands; training a DNN per iteration of an optimization is computationally heavy, and many novel techniques are regularly proposed to mitigate this. Nonetheless, as with search spaces, there are recurring types of approaches. Low-fidelity approximations, which may involve trimming some amount of data or reducing the training-time budget, are perhaps the oldest. Weight sharing is an alternative that is much more preferred for the modern DL model, especially given its scalability, which is ranked high in Table 3. It is very popular in NAS [206] despite its recent entrance into the field. Yet, once again, one could argue that weight sharing is not particularly novel, having been implicitly utilized within the gradient-based HPO method introduced at least as early as in the 1990s [149].

In truth, AutoDL is often both a reappropriation and customization of existing concepts and methodologies, frequently applied with ingenuity, and thus novelty should always be assessed in that light.

II. Solution quality. Current benchmarks for DL-model accuracy and related metrics are dominated by NAS, often in combination with other techniques. It is clear that automatically searching for DNNs can regularly outperform purely handcrafted networks, even if these differences become less compelling for DL tasks where the limits of model correctness are already approached. In such cases, the utility of an AutoDL algorithm becomes more reliant on whether the high accuracy is consistent or cherry-picked; this is a matter of stability. But the point remains that AutoDL has proven value in producing SoTA models.

There is, however, a broader discussion to be had on how solution quality is benchmarked. A common notion is to partake in a comprehensive leaderboard for a popular dataset, such as ImageNet for vision [226], SQuAD for NLP [211], and LibriSpeech for speech recognition [197]. This does allow for direct comparisons between the outcomes of competing AutoDL algorithms. However, such datasets may be amenable to certain characteristics of an algorithm that are useless or detrimental for other datasets, particularly in real-world applications. Unfortunately, there is no good solution here at the current time, given that benchmarking is an arduous process for large-scale AutoDL. It is simply worth highlighting that portions of the DL community put excessive weight on certain evaluations that may not be fairly representative of general solution quality.

In the other extreme, and with supporting evidence, there does exist skepticism around the performance of many NAS algorithms, e.g., whether they are better than random search [306]. While this concern relates to issues of reproducibility, it does perhaps support another assertion that this survey makes: the most important contributor in NAS to model accuracy is search space design, not search strategy. In saying this, it should also not be lost on the reader that the current search spaces yielding best accuracy are derived heavily from popular architectures that are manually designed. It is unclear whether there are better options out there, but this does suggest that, while humans cannot typically beat a NAS algorithm in selecting an optimal candidate DNN, their expert knowledge on general structure is not entirely worthless.

At this current stage, it is not clear whether these emergent principles are the same for other phases of a DL workflow, e.g., whether constraining a subset of FPGA configurations is more worthwhile than applying a different optimization method to hardware search. Admittedly, the reason expert knowledge is so useful in NAS may simply be because the appropriate search space has been arrived at after plenty of human exploration; the benefit of expert knowledge may not be available in other non-NAS contexts. Regardless, while AutoDL research is generally more rudimentary along the rest of the DL workflow, it has been shown that distinct mechanisms can work in concert to push solution quality even further. For example, a recent work achieved 87.3% accuracy on the challenging ImageNet dataset by combining, with the aid of manually designed hyperparameters, automated data engineering with NAS [264]. This reaffirms the value of pursuing an integrated end-to-end AutoDL system, even if no such empirically sound framework currently exists.

III. Efficiency. While accuracy is always an important goal for any form of ML, the ethos of efficiency is particularly compelling for AutoDL as opposed to AutoML, given the computational expense that arises from embracing the complexity of a DNN. Thus the drive for speed and, where applicable, compactness has traditionally come second only to solution correctness within AutoDL literature. In fact, the priorities may have even swapped over the last couple of years due to, one, diminishing communal interest in incremental accuracy improvements and, two, a shift in requirements as AutoML and AutoDL are progressively translated from academia to industry. The broader implications of this translation are worth discussing elsewhere.

As aforementioned, the push for efficiency applies to both the target DL model and the AutoDL process. Given that the latter is usually responsible for training and retraining a DNN, model-development efficiency is usually bundled in as part of the AutoDL processing time. However, the impact of the AutoDL process often outlives its operation and still needs to be accounted for, e.g., a deployment-aware model tuner has good value if, one, it works fast and, two, if the resulting model generates its predictions on a deployment environment in a timely manner. In effect, AutoDL research aims to account for theoretical/realistic inference speed and memory footprint as well. Regardless, simply in terms of AutoDL algorithms, the field has steadily made progress, at least in terms of preferred strategies. For instance, NAS has shifted from low-fidelity approximations to weight sharing, HPO has done the same but into hyper-gradient approximations, automated deployment prefers neural predictors over deployment simulation, and so on. These transitions do not guarantee their benefits though, as efficiency boosts of any form often come with trade-offs, usually in the form of assumptions weakening solution quality. It is thus a current focus within the AutoDL field to work out not just how to shift along a Pareto front but to push it out further.

IV. Stability. The impressive achievements of AutoDL in terms of both solution quality and model-specific efficiency have been tempered by a lack of trust in the results. This cynicism is partially driven by deficiencies in published scientific practice that go beyond the scope of this monograph, but certain limitations are also somewhat understandable in the context of DL due to the computational resources required. The limitation specific to this criterion is that many

works do not go to the extra effort of establishing a statistical analysis for presented results, which makes it difficult to grapple with the stochastic nature of an AutoDL algorithm. That stated, many publications are presently appealing for the reporting culture to be improved [65, 160, 164, 303], and an evaluation of stability is starting to be seen within an increasing amount of AutoDL works.

V. Interpretability. There is not much to say on this matter in AutoDL, given how the broader DL community is itself debating how best to instill some form of transparency in the DL process. However, the absence of substantial focus in this area will likely need some serious consideration at some point. Unlike AutoML, which is able to operate with explainable ML models, AutoDL is wedded to inherently complex connectionist structures. Given that the automation of any process is often associated with another layer of black-boxing, AutoDL may struggle for democratization if it does not somehow avert undue technical obscuration.

VI. Reproducibility. Ensuring that an AutoDL algorithm is repeatable faces the same computational-resource challenges as stability, only this time the onus is on peer review to confirm it. Thus, if a proposed method is to make a splash within the community, the proposer would be advised to facilitate such tests. Unfortunately, this is another widespread deficiency in AutoDL research. For instance, a recent investigation evaluated 12 NAS algorithms and found that most did not report enough details to fully reproduce the original results [160]. This concern has been echoed by many other researchers [65, 164]. In fact, some have proposed potential solutions within the scope of NAS, such as comprehensively filling out a questionnaire [164] or enabling the approach to be easily benchmarked [65, 303]. Early signs indicate that such appeals may be fruitful, with a steadily increasing proportion of new NAS papers providing both code and details.

Repeatability issues are unsurprisingly mimicked in other phases of a DL workflow, not just NAS. In general, they are somewhat alleviated by benchmarking suites [69, 71, 144]. However, while these works encourage good practice in researchers, many do focus on small-scale models and datasets. It thus remains an open question as to how the reproducibility of large-scale experiments, especially those requiring advanced hardware systems [215, 216], can also be guaranteed. Indeed, given their associated requirements for computational resources, there may be only two or three groups in the world that have the capability to do such experiments, and this does not even consider other limitations such as business restrictions or the availability of human expertise.

VII. Engineering quality. The upper echelons of DL experimentation are only accessible to entities with substantial computational resources, such as universities or, often more so, large corporations. Nonetheless, DL has permeated into broader society and, just as AutoML has been embraced by hobbyists and commercial vendors, AutoDL is likely to follow suit over time. There is, however, many differences between code scripted in academic settings and software written for real-world usage. Technical debt is not too much of an issue for one-off experiments, but it can cause substantial problems for real-world AI systems [240]. Moreover, the risks are an order of magnitude greater for AutoDL than DL, simply due to its complex higher-level nature; the former needs more advanced software/library supports.

Granted, as discussed earlier, certain features of AutoDL algorithms may be well-established, such as black-box/gray-box optimization libraries, which have a long history of developmental improvements [5, 94, 253]. In contrast, if AutoDL algorithms rely on more novel concepts, such as recent popular white-box optimization methods, the support may not be there. Fortunately, given the size of the AutoDL community, there is a good possibility of an implementation arising sooner or later. For instance, recent works have started to look into this issue for white-box AutoDL algorithms, providing examples as well [74, 312].

There are a number of new questions to ask once computational theory crosses over into software. Two of them are as follows: can AutoDL algorithms already implemented in these libraries be easily maintained, and can these libraries easily support the implementation of future AutoDL algorithms?

Certain researchers doubt it, concerned about the practical modularity of such algorithms [204, 216]. Worse yet, some high-level AutoDL libraries are tightly coupled with their associated low-level DL library [133, 319], and this constrains their utility. To address these issues, the development of newer libraries has started to explore the use of symbolic programming to keep dependencies minimal [76, 204].

In short, there is currently an impressive amount of activity in the AutoDL community on the engineering side. However, to date, no AutoDL packages have cemented themselves in popular use, e.g., in the same vein as PyTorch or Tensorflow for DL. It is also not clear how best to quantitatively compare AutoDL libraries beyond the minimum requirements proposed in Table 10.

VIII. Scalability. Given that this matter relates to efficiency, the ideal case for usability is if the time and space complexity of an AutoDL algorithm is linear, logarithmic, or constant with respect to the size/quantity of data used, the size of the target DL model, and the relevant search space. Now, at a baseline, the model-evaluation element of an AutoDL process is well controlled, as model training, if relevant, will often involve linear factors in time and/or memory with respect to data inputs and number of model layers. However, it is the search element of the process, whatever that may be according to the phase and sub-category of a DL workflow, that can easily blow out. For instance, for auto-augmentation, the decision to search for an n-length pipeline of data transformations can exponentiate processing time by n. Similarly, NAS can likewise face this exponentiation by simply increasing the depth of a target DNN to construct. Thus, either with conscious or subconscious focus on scalability, AutoDL research does often grapple with how to make search manageable, e.g., by using repeatable NAS cells, even it remains a point of debate whether such constraints block out better solutions [119]. Whatever the case, this issue will remain a challenge for AutoDL, and rigorous analysis of scalability must be done on a case-by-case basis.

IX. Generalizability. Like interpretability, this is a concept about which there is currently little to say within AutoDL as it is rarely rigorously considered. This is perhaps because the field of AutoDL is currently preoccupied with making algorithms work well in an intended context, rather than extending them to work in other contexts. This means that most AutoDL algorithms rely heavily on context-dependent assertions, especially white-box methods, which are preferable for their superior efficiency. The downside is that these are then difficult to transfer from one task to another, e.g., visual classification to speech recognition, let alone from one phase of a DL workflow to another, e.g., from NAS to searching for adaptive mechanisms. Granted, certain elements are more reusable than others, such as modular black-box optimization methods, but the state of computational hardware is not at a level where, on the whole, generalizability trumps efficiency.

X. Eco-friendliness. This notion has quickly attracted increasing attention within the DL community, even if the associated body of published literature currently remains small [156, 201] and many debates exist. When contemplating this matter, in the most general and ideal sense, it is advisable to consider what level of energy usage is most appropriate for a computational task. For instance, the diminishing returns and increasing power required to farm bitcoins is occasionally debated in the media, often accompanied by a dim view of the related profit-driven intent. On the other hand, DL tasks with broad social benefits, e.g., improved health outcomes, would generally be considered worthy of a higher resource expenditure. So, there is a baseline level for the energy value of every computational process, even if this is considerably difficult to quantify and highly subjective. Then perhaps the key to considering whether AutoDL is eco-friendly should be based on counterfactual considerations: what would happen if the same process was not automated?

Most AutoDL processes are done once, whereas their resulting impact on a target DL model can be felt for the duration of its deployment. If the kilowatt-hour cost of the algorithm is y and this reduces the cost of executing a model prediction by x, to be done k times over, then the eco-benefit of AutoDL is felt for $k > y/x$. Admittedly, the rules change for a persistent AutoDL process, such as

continuous monitoring and maintenance. In such a case, the costs of adaptation must be weighed up more carefully [321]. Regardless, analysis in this area is made challenging by how many levels of indirection exist between a line of code and a carbon emission. Different platforms are optimized in different ways, every hardware system appears to have its own unique setup, and checking power usage at a socket is complicated when AutoDL operations are done remotely or in a decentralized manner. Politics may obscure matters further, especially for corporations that are most likely to have the resources to cause significant impact, and even national grids vary in their proportion of renewable energy. So, while there is an increasing drive towards eco-friendly solutions in the community, it is unlikely an associated metric will be benchmarked widely any time soon.

9.3 Challenges and Opportunities

While the previous section emphasized that the AutoDL story has, up to this point, primarily been a NAS story, it does not have to remain that way. In fact, the abbreviation of AutoDL exists to make that distinction, acknowledging that automating high-level operations for DL does not end at constructing a neural network, even if NAS will always remain a core facet of the endeavor. Thus, given how nascent the entirety of the field is, the AutoDL community, however it continues to self-assemble, faces an arguably exciting opportunity to decide what matters most to it within the coming years.

Here, we briefly summarize the five most likely driving forces for future research, as extrapolated from the outcomes of this review. They are: (1) democratized AutoDL, (2) large-scale AutoDL, (3) persistent AutoDL, (4) integrated AutoDL, and (5) self-assembling AutoDL. The listing is in order of least-to-most challenging, although any such forecast is naturally subjective.

Democratized AutoDL. This is the path towards usability. It relies on making AutoDL approaches as accessible to the general public as possible. In principle, it is also the easiest path to progress along, as most of the challenges here are organizational, related to engineering quality, rather than fundamental. Certainly, there are already several AutoDL-related libraries in existence that have high-quality codebases [74, 188, 204, 319]. However, implementations are usually scattered and ad hoc; there is currently no automatic way to evaluate AutoDL code, even though this review has motivated several pertinent questions in Table 10. There is also a challenge to define protocols that can appropriately support AutoDL experiments and maintain related codebases. It does not serve the community well if every new AutoDL algorithm is released in an isolated library that is abandoned after a year. In short, as is progressively being shown for AutoML, there is plenty of interest in industry to leverage the capabilities of AutoDL, and plenty of real-world impact that can be achieved, if only AutoDL was easier to use.

Large-scale AutoDL. This is the path towards capability. It relies on finding ways to make AutoDL algorithms more powerful, capable of grappling with larger datasets, target models, and search spaces. The driving impetus here is as for democratization, specifically the potential for broad real-world usage beyond academia. However, while this path does not directly concern itself with who is employing AutoDL, it is concerned with what AutoDL will be employed upon, namely DL problems that do not hold back in scale. Granted, the automation of data preparation, architecture design, and deployment has already shown some good progress on large-scale classification datasets, and this is why such a direction is not profoundly challenging. Nonetheless, it is an ongoing process to push the power of AutoDL further without relying on computational resources to evolve in the background. Simply put, intelligent search space design and other cost-cutting tricks will need to be developed even further to extend the capability of AutoDL.

Persistent AutoDL. This is the path towards autonomy. It relies on developing robust AutonoDL mechanisms that can adapt DL models, in the absence of human control and oversight, to time-dependent performance deterioration. We stress that this is a weak form of autonomy, in that

the AutonoDL system is still performing the DL task that a human requires of it, but the system is nonetheless autonomously improving and maintaining its predictive/prescriptive ability. Of course, as this review has shown, the AutoDL community has not, by and large, put their focus on dynamic data environments; learning well in static environments is challenging enough. This is clear from the computational resources required for regular updates, as well as the fundamental challenges associated with connectionist structures, e.g., catastrophic interference. However, it is a major shortcoming to ignore this endeavor in the long run, especially as current global events have shown how detrimental it is to rely on fragile static models. On the plus side, research activity into adaptive learning, both recent and historical, has been both intensive and extensive, despite its scattered nature. Present efforts in topics such as domain adaptation and low-shot learning facilitate speedier model retraining for new contexts, and many proposals exist on how to elevate DL into a state of lifelong learning [198]. It will naturally take a concerted effort by sections of the AutoDL community to push forward an AutonoDL agenda, but the theoretical foundations are there to mitigate inherent challenges.

Integrated AutoDL. This is the path towards consolidation. It relies on researching and developing effective frameworks that are able to automate the entire DL workflow in Figure 1. It is a significant step up from existing works that usually only focus on one phase of the workflow, typically model development, even though it is becoming clear that, in terms of both usability and even performance, combining multiple mechanisms can be optimal [264]. Of course, the challenges of designing integrated architectures are numerous and have been discussed at length in the context of AutonoML [140]. Admittedly, ensuring that the constituent mechanisms are sufficiently general is a novel one to AutoDL. Perhaps, the design of an overarching framework should contain a switching capability that activates white-box algorithms for when DL problems are amenable, and black-box/gray-box algorithms for where white-box assumptions fail. Whatever the case, installing and using a single monolithic AutoDL system, rather than haphazardly cobbling together poorly interoperable mechanisms, remains an appealing goal within the field.

Self-assembling AutoDL. This is the path towards emergence. It relies on reassessing the handcrafted nature of AutoDL approaches themselves, seeking alternative ways to evolve better methods from scratch. As of such, it is the most speculative pathway on account of being a paradigm shift in the way that AutoDL is researched. Nonetheless, it is not a whimsical notion; many self-assembly algorithms are very robust, as they typically do not internalize assumptions about a solution space, i.e., in this case, what an AutoDL method should look like. Of these mechanisms, many researchers are most familiar with evolutionary procedures, typically involving population-based mechanics to evolve solutions, but there are also alternative forms of "algorithmic growth" that are based on iteratively unfolding complexity to meet the demands of a task. These latter processes have been discussed in the context of biological brain development [109]. Whatever the mechanism, the idea is that, constrained primarily by the primitives involved, e.g., basic functions, self-assembling AutoDL will gravitate towards whatever works well.

Now, there are obvious reasons why self-assembling AutoDL is considered the most difficult goal of the five listed in this section. A major prohibiting factor is computational complexity, made strongly evident when a DL model-optimizer is itself optimized. It is also largely unnecessary to consider holistic self-assembly when AutoDL is still focused on individual mechanisms, rather than how best to mesh them together [140]. Nonetheless, existing works have explored this direction in limited domains [170, 216], and the idea is attractive. While human expertise can prove beneficial, in that prescribed algorithms and search spaces are often speedy warm-starts towards good solutions, there is no guarantee that human biases do not obscure the very best solutions. Indeed, perhaps superior NAS algorithms combine unusual optimization methods with bizarre search spaces and unpredictably delegated model-evaluation boosts. A human would likely never consider this, but a

suitable self-assembling process may well stumble upon an emergent design pattern that is both novel and effective. So, although there are many more pressing aims for the field of AutoDL, such as consistently decent functionality, it would be remiss to not acknowledge that the high-level algorithms may *themselves* turn out even better if sought out by an automated process.

10   CONCLUSIONS

In this monograph, we have surveyed a vast body of literature, mostly recent, aiming to automate DL at a high level. Rather than focus solely on NAS, which is often treated as the be-all and end-all of this endeavor, we have emulated a previous review on AutonoML [140] by taking a broader perspective of AutoDL research and examining it with respect to an encompassing conceptual framework: a DL workflow. In the process of doing so, we have also noted deficiencies in the field stemming from both publication oversaturation and, more often than not, a lack of theoretical/experimental treatment that is both rigorous and holistic. We have thus also proposed an extensive set of criteria by which, within this work, assessments of algorithm-focused trends in AutoDL are made; we hope to stimulate further contemplation within the DL community as to how a clearer picture of progress can be better supported.

All the foundations for this review were established in Section 2, which not only discussed how the fields of AutoML and AutoDL relate and differ – key considerations included the structure of ML/DL solutions and the computational cost of training – but also elaborated the following notions:

- The DL workflow, which covers the phases of problem formulation and context understanding, data engineering, model development, deployment, and monitoring and maintenance.
- The assessment criteria, which cover novelty, accuracy, efficiency, stability, interpretability, reproducibility, engineering quality, scalability, generalizability, and eco-friendliness.

The subsequent content of the review and certain key conclusions are summarized as follows:

- Section 3 noted that any DL application starts with problem formulation, and there is certainly room for elements of this to be automated to some degree. However, it was acknowledged that the reliance on human involvement would be most challenging to reduce here and, as of such, this phase of a DL workflow has negligible presence in AutoDL research.
- Section 4 examined automated data engineering, categorizing research trends into the supplementary generation of data/labels and the intelligent exploitation of DL model inputs via data augmentation/selection. Although relatively untouched under the lens of automation in DL, data engineering appears to have greater long-term returns on model accuracy than model engineering, suggesting that attention to this area will grow over time. This may potentially improve, as a generalization, the relatively weak efficiencies of associated approaches.
- Section 5 examined NAS, dissecting algorithms for constructing neural networks into the spaces that they search, the optimization strategies that they use, and the tactics that they employ to mitigate expensive model training, if any. This is where the bulk of publications in AutoDL exists, so the topic is well advanced. Even so, the assessment within this review has identified gaps in knowledge here and there, mostly revolving around prerequisites to the maturation of an exploratory field. Specifically, while NAS research is responsible for many algorithms capable of producing high-performance DL models, it is generally not interpretable which elements of a NAS algorithm contribute to that performance, nor is it clear in which contexts that performance is reproducibly attainable.
- Section 6 examined HPO – in AutoDL this conventionally refers to optimizing training-algorithm hyperparameters – and categorized associated approaches by how much they leverage assumptions/knowledge about a training algorithm and hyperparameter space, i.e.,

black-box, gray-box, and white-box. Historically serving as the first expansion of AutoDL beyond NAS, the topic is similarly well studied, especially black-box HPO and, augmented by tactics to minimize the sampling/training of candidate models, gray-box HPO. White-box HPO is newer to the scene, folding HPO into model training for dramatic efficiency boosts, and its strong promise has yet to be validated by rigorous reproducibility studies.

- Section 7 examined automated deployment, with deployment-aware AutoDL and hardware search describing, with respect to a DL model and its deployment environment, which of the two is modified to suit the other one. Relatively untouched, in similar fashion to automated data engineering, the topic nonetheless possesses enough research literature to form a preliminary overview, and it seems likely that attention to this area will intensify in the future as AutoDL continues to be translated from predominantly academic settings to real-world applications. For now, the assessment found that progress in hardware search moves slower than in deployment-aware AutoDL, although this is understandable given that hardware is extremely non-standardized; this does not diminish the importance of the endeavor.
- Section 8 examined automated maintenance, although it acknowledged the challenges of DL, primarily relating to computational costs, that currently make model adaptation frequently infeasible, at least in online fashion. Nonetheless, while there is too little automation-specific research in this area for a substantial assessment, the importance of this phase of the DL workflow to many real-world applications warranted far more in-depth consideration than for automated problem formulation. Accordingly, scenarios for how dynamic data environments may change were discussed, and presently favored paradigms for efficiently training a model around such dynamics were reviewed. These approaches were categorized into domain adaptation, low-shot learning, and continuous learning.

Following the survey, Section 9 focused back on the assessment criteria proposed in this monograph, discussing a possible way in which AutoDL researchers may self-assess and report their theoretical/experimental investigations, so as to better foster a research environment that is conducive to accelerated progress in the field. Then, driven by the same criteria, the section concluded by presenting and discussing an overarching perspective of AutoDL as of the early 2020s, as well as a summary of how the field may evolve in pursuit of: usability, capability, autonomy, consolidation, and emergence.

Ultimately, this review concludes that AutoDL sits in an unusual position, having made exceptional progress in certain areas while also remaining far short of its potential. Granted, it is a nascent endeavor, and there are physical realities that keep general progress restrained; it is unlikely that an end-to-end integrated system for processing DL applications will be seen any time soon, even as the related field of AutoML seems to be moving in that direction [140]. Nonetheless, the trajectory of AutoDL in the broadest sense is also unnecessarily hindered by human factors, such as an all-too-common disregard, intentional or otherwise, for problems other than NAS, goals other than model accuracy, time periods other than the last few years, and so on. We hope that this survey challenges such narrow perspectives, supporting the DL community with an encompassing overview of "what is" and "what can be" in AutoDL.

REFERENCES

[1] Secret Labs AB, Fredrik Lundh, Alex Clark, and Other Contributors. 2011. Python Imaging Library (PIL).

[2] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. 2021. Zero-Cost Proxies for Lightweight NAS. In *International Conference on Learning Representations (ICLR)*.

[3] Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. 2019. Placeto: learning generalizable device placement algorithms for distributed machine learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 3981–3991.

[4] Alan Agresti. 2018. *An introduction to categorical data analysis.* John Wiley & Sons.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 2623–2631.

[6] Abbas Raza Ali, Marcin Budka, and Bogdan Gabrys. 2019. A meta-reinforcement learning approach to optimize parameters and hyper-parameters simultaneously. In *pricai*. 93–106.

[7] Abbas Raza Ali, Bogdan Gabrys, and Marcin Budka. 2018. Cross-domain meta-learning for time-series forecasting. *Procedia Computer Science* 126 (2018), 9–18.

[8] Robert Almgren and Neil Chriss. 2001. Optimal execution of portfolio transactions. *Journal of Risk* 3 (2001), 5–40.

[9] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 29.

[10] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. 2019. PUMA: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*. 715–731.

[11] Raul Astudillo and Peter Frazier. 2019. Bayesian optimization of composite functions. In *The International Conference on Machine Learning (ICML)*. 354–363.

[12] Sungyong Baik, Myungsub Choi, Janghoon Choi, Heewon Kim, and Kyoung Mu Lee. 2020. Meta-Learning with Adaptive Hyperparameters. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[13] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing Neural Network Architectures using Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

[14] Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2018. Accelerating neural architecture search using performance prediction. In *International Conference on Learning Representations (ICLR) Workshop*.

[15] Dana H Ballard. 1987. Modular learning in neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*. 279–284.

[16] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. 2018. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations (ICLR)*.

[17] Justin Bayer, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber. 2009. Evolving memory cell structures for sequence learning. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. 755–764.

[18] Shai Ben-David, John Blitzer, Koby Crammer, Fernando Pereira, et al. 2007. Analysis of representations for domain adaptation. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[19] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *The International Conference on Machine Learning (ICML)*. 550–559.

[20] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. 2020. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 14323–14332.

[21] Yoshua Bengio. 2000. Gradient-based optimization of hyperparameters. *Neural Computation* 12, 8 (2000), 1889–1900.

[22] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. 1990. *Learning a synaptic learning rule.* Citeseer.

[23] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *The International Conference on Machine Learning (ICML)*. 41–48.

[24] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 24.

[25] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)* 13, 2 (2012).

[26] A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter. 2018. CAVE: Configuration Assessment, Visualization and Evaluation. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*.

[27] George EP Box and David R Cox. 1964. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)* 26, 2 (1964), 211–243.

[28] Leo Breiman. 2001. Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statist. Sci.* 16, 3 (aug 2001), 199–231. https://doi.org/10.1214/ss/1009213726

[29] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2018. SMASH: one-shot model architecture search through hypernetworks. In *International Conference on Learning Representations (ICLR)*.

[30] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 33. 1877–1901.

[31] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. 2020. BATS: Binary architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[32] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*.

[33] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. 2018. Path-level network transformation for efficient architecture search. In *The International Conference on Machine Learning (ICML)*. 678–687.

[34] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*.

[35] Gail A Carpenter and Stephen Grossberg. 1987. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing* 37, 1 (1987), 54–115.

[36] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble selection from libraries of models. In *The International Conference on Machine Learning (ICML)*. 18.

[37] Bilge Celik and Joaquin Vanschoren. 2021. Adaptation strategies for automated machine learning on evolving data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 43, 9 (2021), 3067–3078.

[38] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41, 3 (2009), 1–58.

[39] Akshay Chandrashekaran and Ian R Lane. 2017. Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *The Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*. 477–492.

[40] Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. 2021. Searching the Search Space of Vision Transformer. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[41] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. 2016. Net2Net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations (ICLR)*.

[42] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *The USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 578–594.

[43] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. 2021. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations (ICLR)*.

[44] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. 2017. Learning to learn without gradient descent by gradient descent. In *The International Conference on Machine Learning (ICML)*. 748–756.

[45] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. 2019. DetNAS: Backbone Search for Object Detection. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 6642–6652.

[46] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-state Circuits* 52, 1 (2016), 127–138.

[47] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. 2021. DANCE: Differentiable Accelerator/Network Co-Exploration. In *The ACM/ESDA/IEEE Design Automation Conference (DAC)*. 337–342. https://doi.org/10.1109/DAC18074.2021.9586121

[48] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. 2013. Mitosis detection in breast cancer histology images with deep neural networks. In *Proceedings of the International Conference on Medical Image*

Computing and Computer Assisted Intervention (MICCAI). 411–418.

[49] Andrew R Conn, Katya Scheinberg, and Luis N Vicente. 2009. *Introduction to derivative-free optimization.* SIAM.

[50] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. 2019. AutoAugment: Learning augmentation strategies from data. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR).* 113–123.

[51] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. 2020. RandAugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR) Workshop.* 702–703.

[52] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. 2021. FBNetV3: Joint architecture-recipe search using neural acquisition function. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR).* 16276–16285.

[53] Erik A Daxberger, Anastasia Makarova, Matteo Turchetta, and Andreas Krause. 2020. Mixed-Variable Bayesian Optimization. In *International Joint Conferences on Artificial Intelligence (IJCAI).* 2633–2639.

[54] Kenneth De Jong. 2016. Evolutionary computation: a unified approach. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion.* 185–199.

[55] Georgi Dikov and Justin Bayer. 2019. Bayesian learning of neural network architectures. In *The International Conference on Artificial Intelligence and Statistics (AISTATS).* 730–738.

[56] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *International Joint Conferences on Artificial Intelligence (IJCAI).*

[57] Justin Domke. 2012. Generic methods for optimization-based modeling. In *The International Conference on Artificial Intelligence and Statistics (AISTATS).* 318–326.

[58] Xuanyi Dong, Junshi Huang, Yi Yang, and Shuicheng Yan. 2017. More is Less: A More Complicated Network with Less Inference Complexity. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR).* 5840–5848.

[59] Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. 2021. NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2021). https://doi.org/10.1109/TPAMI.2021.3054824 doi:10.1109/TPAMI.2021.3054824.

[60] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. 2021. AutoHAS: Efficient Hyperparameter and Architecture Search. In *International Conference on Learning Representations (ICLR) Workshop.*

[61] Xuanyi Dong and Yi Yang. 2019. Network Pruning via Transformable Architecture Search. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS).* 760–771.

[62] Xuanyi Dong and Yi Yang. 2019. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *Proceedings of the IEEE International Conference Computer Vision (ICCV).* 3681–3690. https://doi.org/10.1109/ICCV.2019.00378

[63] Xuanyi Dong and Yi Yang. 2019. Searching for A Robust Neural Architecture in Four GPU Hours. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR).* 1761–1770.

[64] Xuanyi Dong and Yi Yang. 2019. Teacher Supervises Students How to Learn from Partially Labeled Images for Facial Landmark Detection. In *Proceedings of the IEEE International Conference Computer Vision (ICCV).* 783–792.

[65] Xuanyi Dong and Yi Yang. 2020. NAS-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR).*

[66] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Sequential scenario-specific meta learner for online recommendation. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).* 2895–2904.

[67] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. 2016. RL2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779* (2016).

[68] Salvador Dura-Bernal, Benjamin A Suter, Padraig Gleeson, Matteo Cantarelli, Adrian Quintana, Facundo Rodriguez, David J Kedziora, George L Chadderdon, Cliff C Kerr, Samuel A Neymotin, Robert A McDougal, Michael Hines, Gordon MG Shepherd, and William W Lytton. 2019. NetPyNE, a tool for data-driven multiscale modeling of brain circuits. *eLife* 8 (apr 2019). https://doi.org/10.7554/elife.44494

[69] Katharina Eggensperger, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2014. Surrogate Benchmarks for Hyperparameter Optimization. In *The European Conference on Artificial Intelligence (ECAI) Workshop.* 24–31.

[70] Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. 2021. HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS).*

[71] Theresa Eimer, André Biedenkapp, Maximilian Reimer, Steven Adriaensen, Frank Hutter, and Marius Lindauer. 2021. DACBench: A Benchmark Library for Dynamic Algorithm Configuration. In *International Joint Conferences on Artificial Intelligence (IJCAI).* 1668–1674.

[72] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research (JMLR)* 20, 55 (2019), 1–21.

[73] Epic Games. 2019. Unreal Engine. https://www.unrealengine.com.

[74] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. In *The International Conference on Machine Learning (ICML) Workshop*.

[75] Facebook AI Research [n. d.]. Papers With Code. https://paperswithcode.com.

[76] Facebook Inc. 2021. PyTorch V1.10.0. https://github.com/pytorch/pytorch/releases/tag/v1.10.0.

[77] Stefan Falkner, Aaron Klein, and Frank Hutter. 2018. BOHB: Robust and efficient hyperparameter optimization at scale. In *The International Conference on Machine Learning (ICML)*. 1437–1446.

[78] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. 2018. Learning to teach. In *International Conference on Learning Representations (ICLR)*.

[79] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. 2020. Densely connected search space for more flexible neural architecture search. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 10628–10637.

[80] Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. In *Automated Machine Learning*. Springer, Cham, 3–33.

[81] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. 2017. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*.

[82] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *The International Conference on Machine Learning (ICML)*. 1126–1135.

[83] Alexander IJ Forrester, András Sóbester, and Andy J Keane. 2007. Multi-fidelity optimization via surrogate modelling. In *Mathematical, Physical and Engineering Sciences*, Vol. 463. 3251–3269.

[84] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *The International Conference on Machine Learning (ICML)*.

[85] Kunihiko Fukushima. 2013. Training multi-layered neural network neocognitron. *Neural Networks* 40 (2013), 18–31.

[86] Kunihiko Fukushima and Sei Miyake. 1982. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and Cooperation in Neural Nets*. Springer, 267–285.

[87] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–37.

[88] Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised domain adaptation by backpropagation. In *The International Conference on Machine Learning (ICML)*. 1180–1189.

[89] David Gelernter. 1993. *Mirror worlds: Or the day software puts the universe in a shoebox... How it will happen and what it will mean.* Oxford University Press.

[90] Ian Gemp, Georgios Theocharous, and Mohammad Ghavamzadeh. 2017. Automated data cleansing through meta-learning. In *AAAI Conference on Artificial Intelligence (AAAI)*. 4760––4761.

[91] Oguzhan Gencoglu, Mark van Gils, Esin Guldogan, Chamin Morikawa, Mehmet Süzen, Mathias Gruber, Jussi Leinonen, and Heikki Huttunen. 2019. HARK Side of Deep Learning–From Grad Student Descent to Automated Machine Learning. *arXiv preprint arXiv:1904.07633* (2019).

[92] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. 2019. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 7036–7045.

[93] David E Goldberg and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*. Vol. 1. 69–93.

[94] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. 2017. Google vizier: A service for black-box optimization. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1487–1495.

[95] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[96] Google. 2017. Edge TPU. https://cloud.google.com/edge-tpu.

[97] Stephen Grossberg. 2020. Toward Autonomous Adaptive Intelligence: Building Upon Neural Models of How Brains Make Minds. *IEEE Transactions on Systems, Man, and Cybernetics: Systems (TSMC)* 51, 1 (2020), 51–75.

[98] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2019. A survey of methods for explaining black box models. *ACM Computing Surveys (CSUR)* 51, 5 (2019), 1–42.

[99] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the European Conference on Computer*

Vision (ECCV). Springer, 544–560.

[100] David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *International Conference on Learning Representations (ICLR)*.

[101] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

[102] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations (ICLR)*.

[103] Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. 2021. Dynamic neural networks: A survey. *arXiv preprint arXiv:2102.04906* (2021).

[104] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation* 9, 2 (2001), 159–195.

[105] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *Proceedings of the IEEE International Conference Computer Vision (ICCV)*. 2961–2969.

[106] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 770–778.

[107] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the International Conference on World Wide Web (WWW)*. 173–182.

[108] Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations (ICLR)*.

[109] Peter Robin Hiesinger. 2021. *The Self-Assembling Brain: How Neural Networks Grow Smarter*. Princeton University Press. https://doi.org/doi:10.1515/9780691215518

[110] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2014. Distilling the knowledge in a neural network. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS) Workshop*.

[111] Geoffrey E Hinton and David C Plaut. 1987. Using fast weights to deblur old memories. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*. 177–186.

[112] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[113] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. 2001. Learning to learn using gradient descent. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. 87–94.

[114] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. 2008. Kernel methods in machine learning. *The Annals of Statistics* (2008), 1171–1220.

[115] John J Hopfield. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* 79, 8 (1982), 2554–2558.

[116] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5 (1989), 359–366.

[117] Harold Hotelling. 1933. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology* 24, 6 (1933), 417.

[118] Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. 2018. Evolved policy gradients. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[119] Hanzhang Hu, John Langford, Rich Caruana, Eric Horvitz, and Debadeepta Dey. 2018. Macro Neural Architecture Search Revisited. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS) Workshop*.

[120] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 4700–4708.

[121] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION)*. 507–523.

[122] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research (JAIR)* 36 (2009), 267–306.

[123] Ching-Lai Hwang and Abu Syed Md Masud. 2012. *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media.

[124] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *The International Conference on Machine Learning (ICML)*. 448–456.

[125] Alexey Grigorevich Ivakhnenko. 1971. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics (SMC)* 4 (1971), 364–378.

[126] Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: convergence and generalization in neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 8580–8589.

[127] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. 2017. Population based training of neural networks. *arXiv preprint arXiv:1711.09846* (2017).

[128] Kevin Jamieson and Ameet Talwalkar. 2016. Non-stochastic best arm identification and hyperparameter optimization. In *The International Conference on Artificial Intelligence and Statistics (AISTATS)*. 240–248.

[129] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*.

[130] Rodolphe Jenatton, Cedric Archambeau, Javier González, and Matthias Seeger. 2017. Bayesian optimization with tree-structured dependencies. In *The International Conference on Machine Learning (ICML)*. 1655–1664.

[131] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 2012. 3D convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35, 1 (2012), 221–231.

[132] Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. 2020. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 39, 12 (2020), 4805–4815.

[133] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1946–1956.

[134] Hadi S Jomaa, Josif Grabocka, and Lars Schmidt-Thieme. 2019. Hyp-RL: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527* (2019).

[135] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *The International Conference on Machine Learning (ICML)*. 2342–2350.

[136] Petr Kadlec and Bogdan Gabrys. 2009. Architecture for development of adaptive on-line prediction models. *Memetic Computing* 1, 4 (2009), 241–269.

[137] Kirthevasan Kandasamy, Gautam Dasarathy, Junier Oliva, Jeff Schneider, and Barnabás Póczos. 2016. Gaussian Process Bandit Optimisation with Multi-fidelity Evaluations. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 1000–1008.

[138] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[139] Zohar Karnin, Tomer Koren, and Oren Somekh. 2013. Almost optimal exploration in multi-armed bandits. In *The International Conference on Machine Learning (ICML)*. 1238–1246.

[140] David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. 2020. AutonoML: Towards an Integrated Framework for Autonomous Machine Learning. *arXiv preprint arXiv:2012.12600* (2020).

[141] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*.

[142] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526. https://doi.org/10.1073/pnas.1611835114

[143] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. 2020. Improving generalization in meta reinforcement learning using learned objectives. In *International Conference on Learning Representations (ICLR)*.

[144] Aaron Klein and Frank Hutter. 2019. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970* (2019).

[145] Ksenia Konyushkova, Sznitman Raphael, and Pascal Fua. 2017. Learning active learning from data. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 4228–4238.

[146] Bartosz Krawczyk, Leandro L Minku, Joao Gama, Jerzy Stefanowski, and Michał Woźniak. 2017. Ensemble learning for data stream analysis: A survey. *Information Fusion* 37 (2017), 132–156.

[147] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images.

[148] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 1097–1105.

[149] Jan Larsen, Lars Kai Hansen, Claus Svarer, and M Ohlsson. 1996. Design and regularization of neural networks: the optimal use of a validation set. In *IEEE Workshop on Neural Networks for Signal Processing (NNSP)*. 62–71.

[150] Jan Larsen, Claus Svarer, Lars Nonboe Andersen, and Lars Kai Hansen. 2002. Adaptive Regularization in Neural Network Modeling. In *Neural Networks: Tricks of the Trade*. https://doi.org/10.1007/3-540-49430-8_6

[151] John Lazzaro, Sylvie Ryckebusch, Misha Anne Mahowald, and Caver A Mead. 1988. Winner-take-all networks of O (n) complexity. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, Vol. 1.

[152] Yann LeCun. 1985. A learning scheme for asymmetric threshold networks. *Proceedings of Cognitiva* 85, 537 (1985), 599–604.

[153] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[154] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1, 4 (1989), 541–551.

[155] Ang Li, Ola Spyra, Sagi Perel, Valentin Dalibard, Max Jaderberg, Chenjie Gu, David Budden, Tim Harley, and Pramod Gupta. 2019. A generalized framework for population based training. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 1791–1799.

[156] Bo Li, Xinyang Jiang, Donglin Bai, Yuge Zhang, Ningxin Zheng, Xuanyi Dong, Lu Liu, Yuqing Yang, and Dongsheng Li. 2021. Full-Cycle Energy Consumption Benchmark for Low-Carbon Computer Vision. *arXiv preprint arXiv:2108.13465* (2021).

[157] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. 2021. HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark. In *International Conference on Learning Representations (ICLR)*.

[158] Ke Li and Jitendra Malik. 2017. Learning to optimize. In *International Conference on Learning Representations (ICLR)*.

[159] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2018. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research (JMLR)* 18, 1 (2018), 6765–6816.

[160] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*. 367–377.

[161] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M Robertson, and Yongxin Yang. 2020. DADA: Differentiable Automatic Data Augmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 580–595.

[162] Zhizhong Li and Derek Hoiem. 2017. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 40, 12 (2017), 2935–2947.

[163] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. 2019. Fast autoaugment. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[164] Marius Lindauer and Frank Hutter. 2020. Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research (JMLR)* 21, 243 (2020), 1–18.

[165] Seppo Linnainmaa. 1970. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki* (1970), 6–7.

[166] William A Little. 1974. The existence of persistent states in the brain. *Mathematical Biosciences* 19, 1-2 (1974), 101–120.

[167] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 806–814.

[168] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. 2019. Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 82–92.

[169] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[170] Hanxiao Liu, Andrew Brock, Karen Simonyan, and Quoc V Le. 2020. Evolving normalization-activation layers. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[171] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *International Conference on Learning Representations (ICLR)*.

[172] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, Xuanyi Dong, and Chengqi Zhang. 2021. Isometric propagation network for generalized zero-shot learning. In *International Conference on Learning Representations (ICLR)*.

[173] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Learning to propagate for graph meta-learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[174] Jonathan Lorraine, Paul Vicol, and David Duvenaud. 2020. Optimizing millions of hyperparameters by implicit differentiation. In *The International Conference on Artificial Intelligence and Statistics (AISTATS)*. 1540–1552.

[175] Ilya Loshchilov and Frank Hutter. 2016. CMA-ES for hyperparameter optimization of deep neural networks. In *International Conference on Learning Representations (ICLR) Workshop*.

[176] Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. 2016. Scalable gradient-based tuning of continuous regularization hyperparameters. In *The International Conference on Machine Learning (ICML)*. 2952–2960.

[177] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. 2018. Neural architecture optimization. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 7827–7838.

[178] Edward Ma. 2019. NLP Augmentation. https://github.com/makcedward/nlpaug.

[179] Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. 2019. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations (ICLR)*.

[180] Dougal Maclaurin, David Duvenaud, and Ryan Adams. 2015. Gradient-based hyperparameter optimization through reversible learning. In *The International Conference on Machine Learning (ICML)*. 2113–2122.

[181] Jorge G Madrid, Hugo Jair Escalante, Eduardo F Morales, Wei-Wei Tu, Yang Yu, Lisheng Sun-Hosoya, Isabelle Guyon, and Michèle Sebag. 2018. Towards AutoML in the presence of Drift: first results. In *The International Conference on Machine Learning (ICML) Workshop*.

[182] R Timothy Marler and Jasbir S Arora. 2004. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization* 26, 6 (2004), 369–395.

[183] Yannick Martel, Arne Roßmann, Eldar Sultanow, Oliver Weiß, Matthias Wissel, Frank Pelzel, and Matthias Seßler. 2021. Software Architecture Best Practices for Enterprise Artificial Intelligence. *INFORMATIK* (2021).

[184] Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Vol. 24. Elsevier, 109–165.

[185] Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics* 5, 4 (1943), 115–133.

[186] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. 2021. Neural Architecture Search without Training. In *The International Conference on Machine Learning (ICML)*.

[187] Donald Michie, David J Spiegelhalter, CC Taylor, et al. 1994. Machine Learning. *Neural and Statistical Classification* 13, 1994 (1994), 1–298.

[188] Microsoft Corporation 2018. Neural Network Intelligence. https://github.com/microsoft/nni.

[189] Mitar Milutinovic, Atılım Güneş Baydin, Robert Zinkov, William Harvey, Dawn Song, Frank Wood, and Wade Shen. 2017. End-to-end training of differentiable pipelines across machine learning frameworks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS) Workshop*.

[190] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V Le, and Jeff Dean. 2018. A hierarchical model for device placement. In *International Conference on Learning Representations (ICLR)*.

[191] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.

[192] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *The International Conference on Machine Learning (ICML)*. 2430–2439.

[193] Jonas Močkus. 1975. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*. Springer, 400–404.

[194] Luigi Nardi, David Koeplinger, and Kunle Olukotun. 2019. Practical design space exploration. In *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 347–358.

[195] Tien-Dung Nguyen, David Jacob Kedziora, Katarzyna Musial, and Bogdan Gabrys. 2021. Exploring Opportunistic Meta-knowledge to Reduce Search Spaces for Automated Machine Learning. In *International Joint Conference on Neural Network (IJCNN)*.

[196] Tong Niu and Mohit Bansal. 2019. Automatically Learning Data Augmentation Policies for Dialogue Tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1317–1323.

[197] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: an asr corpus based on public domain audio books. In *IEEE International conference on acoustics, speech and signal processing (ICASSP)*. 5206–5210.

[198] German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* 113 (2019), 54–71. https://doi.org/10.1016/j.neunet.2019.01.012

[199] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. 2015. Deep face recognition. In *Proceedings of the British Machine Vision Conference (BMVC)*. 41.1–41.12.

[200] Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. 2019. PABO: Pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In *The IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[201] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*

(2021).

[202] Karl Pearson. 1901. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

[203] Fabian Pedregosa. 2016. Hyperparameter optimization with approximate gradient. In *The International Conference on Machine Learning (ICML)*. 737–746.

[204] Daiyi Peng, Xuanyi Dong, Esteban Real, Mingxing Tan, Yifeng Lu, Gabriel Bender, Hanxiao Liu, Adam Kraft, Chen Liang, and Quoc Le. 2020. PyGlove: Symbolic Programming for Automated Machine Learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[205] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. 2018. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In *Proceedings of the EuroSys Conference (EuroSys)*. 1–14.

[206] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *The International Conference on Machine Learning (ICML)*. 4095–4104.

[207] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2021. Improving reproducibility in machine learning research: a report from the NeurIPS 2019 reproducibility program. *Journal of Machine Learning Research (JMLR)* 22 (2021).

[208] Jordan B Pollack. 1989. Implications of recursive distributed representations. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 527–536.

[209] J. Ross Quinlan. 1986. Induction of decision trees. *Machine learning* 1, 1 (1986), 81–106.

[210] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. 2020. Designing network design spaces. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 10428–10436.

[211] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[212] Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Searching for activation functions. *arXiv preprint arXiv:1710.05941* (2017).

[213] Sachin Ravi and Hugo Larochelle. 2017. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*.

[214] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017. A case for efficient accelerator design space exploration via bayesian optimization. In *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6.

[215] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*. 4780–4789.

[216] Esteban Real, Chen Liang, David So, and Quoc Le. 2020. AutoML-Zero: evolving machine learning algorithms from scratch. In *The International Conference on Machine Learning (ICML)*. 8007–8019.

[217] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-Scale Evolution of Image Classifiers. In *The International Conference on Machine Learning (ICML)*. 2902–2911.

[218] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *The International Conference on Machine Learning (ICML)*. 4334–4343.

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

[220] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[221] Martin Riedmiller and Heinrich Braun. 1993. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference on Neural Networks*. 586–591.

[222] Bernardino Romera-Paredes and Philip Torr. 2015. An embarrassingly simple approach to zero-shot learning. In *The International Conference on Machine Learning (ICML)*. 2152–2161.

[223] Binxin Ru, Ahsan Alvi, Vu Nguyen, Michael A Osborne, and Stephen Roberts. 2020. Bayesian optimisation over multiple continuous and categorical inputs. In *The International Conference on Machine Learning (ICML)*. 8276–8285.

[224] Binxin Ru, Pedro Esperanca, and Fabio Carlucci. 2020. Neural Architecture Generator Optimization. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[225] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[226] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander Berg, and Li Fei-Fei. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)* 115, 3 (2015), 211–252.

[227] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *arXiv preprint arXiv:1606.04671* (2016).

[228] Dymitr Ruta, Bogdan Gabrys, and Christiane Lemke. 2011. A generic multilevel architecture for time series prediction. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 23, 3 (2011), 350–359.

[229] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[230] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2016. Towards Automatic Composition of Multicomponent Predictive Systems. In *International Conference on Hybrid Artificial Intelligence Systems*. 27–39.

[231] Manuel Martin Salvador, Marcin Budka, and Bogdan Gabrys. 2019. Automatic Composition and Optimization of Multicomponent Predictive Systems With an Extended Auto-WEKA. *IEEE Transactions on Automation Science and Engineering (TASE)* 16, 2 (apr 2019), 946–959. https://doi.org/10.1109/tase.2018.2876430

[232] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 4510–4520.

[233] Tom Schaul and Jürgen Schmidhuber. 2010. Metalearning. *Scholarpedia* 5, 6 (2010), 4650.

[234] Jürgen Schmidhuber. 1987. *Evolutionary Principles in Self-referential Learning*. Ph. D. Dissertation.

[235] Jürgen Schmidhuber. 1992. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation* 4, 1 (1992), 131–139.

[236] Juergen Schmidhuber. 2002. Bias-optimal incremental problem solving. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[237] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61 (2015), 85–117.

[238] Jürgen Schmidhuber, Jieyu Zhao, and Nicol N Schraudolph. 1998. Reinforcement learning with self-modifying policies. In *Learning to learn*. Springer, 293–309.

[239] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[240] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 2503–2511.

[241] Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. 2019. Truncated back-propagation for bilevel optimization. In *The International Conference on Artificial Intelligence and Statistics (AISTATS)*. 1723–1732.

[242] Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data* 6, 1 (2019), 1–48.

[243] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. 2019. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[244] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. 2020. NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777* (2020).

[245] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[246] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*.

[247] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. 2018. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations (ICLR)*.

[248] Kate A Smith-Miles. 2009. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)* 41, 1 (2009), 1–25.

[249] Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 4077–4087.

[250] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 2951–2959.

[251] David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *The International Conference on Machine Learning (ICML)*. 5877–5886.

[252] David R So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. 2021. Primer: Searching for efficient transformers for language modeling. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[253] Qingquan Song, Haifeng Jin, and Xia Hu. 2022. *Automated Machine Learning in Action*. Manning Publications Co.

[254] Pablo Sprechmann, Siddhant M Jayakumar, Jack W Rae, Alexander Pritzel, Adria Puigdomenech Badia, Benigno Uria, Oriol Vinyals, Demis Hassabis, Razvan Pascanu, and Charles Blundell. 2018. Memory-based parameter adaptation. In *International Conference on Learning Representations (ICLR)*.

[255] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *The Annual Meeting of the Association for Computational Linguistics*.

[256] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeffrey Clune. 2020. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *The International Conference on Machine Learning (ICML)*. 9206–9216.

[257] Richard S Sutton. 1992. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI Conference on Artificial Intelligence (AAAI)*. 171–176.

[258] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[259] Mehmet Süzen. 2020. Equivalence in Deep Neural Networks via Conjugate Matrix Ensembles. *arXiv preprint arXiv:2006.13687* (2020).

[260] Mehmet Süzen, Joan J Cerdà, and Cornelius Weber. 2019. Periodic Spectral Ergodicity: A Complexity Measure for Deep Neural Networks and Neural Architecture Search. *arXiv preprint arXiv:1911.07831* (2019).

[261] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. 2014. Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896* (2014).

[262] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 1–9.

[263] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 2820–2828.

[264] Mingxing Tan and Quoc V Le. 2021. EfficientNetV2: Smaller models and faster training. In *The International Conference on Machine Learning (ICML)*, Vol. 139. 10096–10106.

[265] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 847–855.

[266] Sebastian Thrun and Tom M Mitchell. 1995. Lifelong robot learning. *Robotics and autonomous systems* 15, 1-2 (1995), 25–46.

[267] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*. 6000–6010.

[268] Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Richard Lewis, Janarthanan Rajendran, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. 2019. Discovery of useful questions as auxiliary tasks. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[269] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *The International Conference on Machine Learning (ICML)*. 1096–1103.

[270] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. 2020. FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 12965–12974.

[271] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Wenjun Zeng, and Tao Qin. 2021. Generalizing to Unseen Domains: A Survey on Domain Generalization. *arXiv preprint arXiv:2103.03097* (2021).

[272] Jane X Wang, Zeb Kurth-Nelson, Hubert Soyer, Joel Z Leibo, Dhruva Tirumala, Rémi Munos, Charles Blundell, Dharshan Kumaran, and Matt M Botvinick. 2017. Learning to reinforcement learn. In *Cognitive Science Society (CogSci)*.

[273] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 8612–8620.

[274] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. 2020. APQ: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 2078–2087.

[275] Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. 2018. Dataset distillation. *arXiv preprint arXiv:1811.10959* (2018).

[276] Christopher John Cornish Hellaby Watkins. 1989. Learning from delayed rewards. *PhD thesis, University of Cambridge* (1989).

[277] Andreas S Weigend and Neil A Gershenfeld. 1993. Results of the time series prediction competition at the Santa Fe Institute. In *IEEE International Conference on Neural Networks*. 1786–1793.

[278] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural predictor for neural architecture search. In *European Conference on Computer Vision*. Springer, 660–676.

[279] Colin White, Willie Neiswanger, and Yash Savani. 2021. BANANAS: Bayesian optimization with neural architectures for neural architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*.

[280] Martha White and Adam White. 2016. A Greedy Approach to Adapting the Trace Parameter for Temporal Difference Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 557–565.

[281] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[282] Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. 2019. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392* (2019).

[283] David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82.

[284] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 10734–10742.

[285] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. 2018. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090* (2018).

[286] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. 2018. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 41, 9 (2018), 2251–2265.

[287] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. 2019. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE International Conference Computer Vision (ICCV)*. 1284–1293.

[288] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2019. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*.

[289] Huan Xiong, Lei Huang, Mengyang Yu, Li Liu, Fan Zhu, and Ling Shao. 2020. On the number of linear regions of convolutional neural networks. In *The International Conference on Machine Learning (ICML)*. 10514–10523.

[290] Yunyang Xiong, Ronak Mehta, and Vikas Singh. 2019. Resource constrained neural network architecture search: Will a submodularity assumption help?. In *Proceedings of the IEEE International Conference Computer Vision (ICCV)*. 1901–1910.

[291] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2020. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations (ICLR)*.

[292] Zhongwen Xu, Hado van Hasselt, Matteo Hessel, Junhyuk Oh, Satinder Singh, and David Silver. 2020. Meta-gradient reinforcement learning with an objective discovered online. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[293] Zhongwen Xu, Hado van Hasselt, and David Silver. 2018. Meta-gradient reinforcement learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[294] Shen Yan, Colin White, Yash Savani, and Frank Hutter. 2021. NAS-Bench-x11 and the Power of Learning Curves. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*.

[295] Li Yang and Abdallah Shami. 2020. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415 (2020), 295–316.

[296] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *The ACM/ESDA/IEEE Design Automation Conference (DAC)*. 1–6.

[297] Ming Yang, Shuiwang Ji, Wei Xu, Jinjun Wang, Fengjun Lv, Kai Yu, Yihong Gong, Mert Dikmen, Dennis J Lin, and Thomas S Huang. 2009. Detecting human actions in surveillance videos. In *TREC Video Retrieval Evaluation Workshop*. Citeseer.

[298] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. 2017. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 5687–5695.

[299] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. 2018. NetAdapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 285–300.

[300] Xiao Yang, Weiqing Liu, Dong Zhou, Jiang Bian, and Tie-Yan Liu. 2020. Qlib: An AI-oriented Quantitative Investment Platform. *arXiv preprint arXiv:2009.11189* (2020).

[301] Zhaohui Yang, Yunhe Wang, Kai Han, Chunjing Xu, Chao Xu, Dacheng Tao, and Chang Xu. 2020. Searching for Low-Bit Weights in Quantized Neural Networks. In *Proceedings of the International Conference on Neural Information*

Processing Systems (NeurIPS).

[302] Chris Ying. 2019. Enumerating unique computational graphs via an iterative graph invariant. *arXiv preprint arXiv:1902.06192* (2019).

[303] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. 2019. NAS-Bench-101: Towards reproducible neural architecture search. In *The International Conference on Machine Learning (ICML)*. 7105–7114.

[304] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. BigNAS: Scaling up neural architecture search with big single-stage models. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 702–717.

[305] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations (ICLR)*.

[306] Kaicheng Yu, Christian Sciuto, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. 2020. Evaluating the search phase of neural architecture search. In *International Conference on Learning Representations (ICLR)*.

[307] Tong Yu and Hong Zhu. 2020. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689* (2020).

[308] Sheheryar Zaidi, Arber Zela, Thomas Elsken, Chris Holmes, Frank Hutter, and Yee Whye Teh. 2020. Neural ensemble search for performant and calibrated predictions. In *The International Conference on Machine Learning (ICML) Workshop*.

[309] Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. 2020. Understanding and robustifying differentiable architecture search. In *International Conference on Learning Representations (ICLR)*.

[310] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. 2018. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In *The International Conference on Machine Learning (ICML) Workshop*.

[311] Arber Zela, Julien Siems, and Frank Hutter. 2020. NAS-Bench-1Shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations (ICLR)*.

[312] Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, and Lidong Zhou. 2020. Retiarii: A Deep Learning Exploratory-Training Framework. In *The USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 919–936.

[313] Wei Zhang and Thomas G Dietterich. 1995. A reinforcement learning approach to job-shop scheduling. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, Vol. 95. 1114–1120.

[314] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 2423–2432.

[315] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. 2019. BayesNAS: A bayesian approach for neural architecture search. In *The International Conference on Machine Learning (ICML)*. 7603–7613.

[316] Yanqi Zhou, Xuanyi Dong, Berkin Akin, Mingxing Tan, Daiyi Peng, Tianjian Meng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2022. Rethinking Co-design of Neural Architectures and Hardware Accelerators. In *The Conference on Machine Learning and Systems (MLSys)*.

[317] Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. 2009. Multi-instance learning by treating instances as non-iid samples. In *The International Conference on Machine Learning (ICML)*. 1249–1256.

[318] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference Computer Vision (ICCV)*. 2223–2232.

[319] Lucas Zimmer, Marius Lindauer, and Frank Hutter. 2021. Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2021).

[320] Indre Zliobaite, Albert Bifet, Mohamed Gaber, Bogdan Gabrys, Joao Gama, Leandro Minku, and Katarzyna Musial. 2012. Next challenges for adaptive learning systems. *ACM SIGKDD Explorations Newsletter* 14, 1 (2012), 48–55.

[321] Indrė Žliobaitė, Marcin Budka, and Frederic Stahl. 2015. Towards cost-sensitive adaptation: When is it worth updating your predictive model? *Neurocomputing* 150 (feb 2015), 240–249. https://doi.org/10.1016/j.neucom.2014.05.084

[322] Indre Zliobaite and Bogdan Gabrys. 2014. Adaptive preprocessing for streaming data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 26, 2 (2014), 309–321.

[323] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. 2020. Learning data augmentation strategies for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 566–583.

[324] Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[325] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference Computer Vision Pattern Recognition (CVPR)*. 8697–8710.