Efficient Evolutionary Hyperparameter Optimization for Deep Learning

Aleksandr Chebykin

Efficient Evolutionary Hyperparameter Optimization for Deep Learning

Efficient Evolutionary Hyperparameter Optimization for Deep Learning

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus prof. dr. ir. Hester Bijl,
chair of the Board for Doctorates,
to be defended publicly on
Tuesday, 23 June 2026 at 12:30

by

Aleksandr CHEBYKIN

This dissertation has been approved by the promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. P.A.N. Bosman,	Delft University of Technology, Centrum Wiskunde & Informatica, *promotor*
Prof. dr. T. Alderliesten,	Delft University of Technology, Leiden University Medical Center, *promotor*
Independent members:	
Prof. dr. M.T.J. Spaan	Delft University of Technology
Prof. dr. G.C.H.E. de Croon	Delft University of Technology
Prof. dr. T.H.W. Bäck	Leiden University
Dr. J.N. van Rijn	Leiden University
Prof. dr. P.S. Cesar Garcia	Delft University of Technology, reserve member

This thesis was part of the research project DAEDALUS (project number 18373), funded via the Open Technology Programme of the Dutch Research Council (NWO), Elekta, and ORTEC LogiqCare. The research was carried out in collaboration with the Leiden University Medical Center (LUMC).

SIKS Dissertation Series No. 2026-39.
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Keywords:	hyperparameter optimization, deep learning, evolutionary algorithms, neural architecture search
Printed by:	Proefschriftspecialist
Front & Back:	Cover art created by the author working with ChatGPT (chatgpt.com), Nano Banana (aistudio.google.com), and Photoshop (photoshop.com).

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.
The source code associated with this thesis is available in 4TU.ResearchData at
https://data.4tu.nl/datasets/732d2ec8-6a24-4619-8a22-c7d286dce5d2.

With our worries left behind,
Haste stopped in its tracks.
Robots never halt their grind,
Humans can relax.

Yuri Entin*

*Poetic translation by the thesis author.

Contents

Summary

Artificial Intelligence (AI) is an idea, a set of research subfields, and, ultimately, a suite of technologies that is reshaping the world. AI systems are intended to solve problems that would otherwise require biological or human intelligence to address. Since the middle of the 2010s, breakthroughs in the AI subfield of *deep learning* enabled rapid progress in computer vision, natural language processing, generative modelling, and other areas.

The key idea of deep learning is to use sufficiently large quantities of data to set parameters of a *neural network* such that it will perform well on a target task. A neural network is a directed graph of parameterized operations that transform a numeric input into a numeric output. The parameters of a network can be optimized via gradient-based techniques, in contrast to the *hyper*parameters that are often manually set by an expert. Typical hyperparameter categories are the settings of the gradient-based optimizer, the choice of operations used in the network, and the structure of its computational graph. The latter two are commonly referred to as the *architecture* of a network, and optimizing them is called Neural Architecture Search (NAS).

Hyperparameters can strongly influence both the performance of a network on the target task, and its efficiency. Therefore, it is important to find good hyperparameter values. The goal of hyperparameter optimization algorithms is to automate this process, which is challenging in the deep learning context for several reasons. Firstly, to evaluate how good a set of hyperparameter values is, a network typically needs to be trained, which takes time and expensive hardware, thus restricting how many sets of hyperparameter values can be evaluated on a finite budget. Secondly, neural networks require many hyperparameters to be set, with each having many potential values that non-trivially interact with those of other hyperparameters, leading to large search spaces that may be difficult to optimize in. Finally, hyperparameter optimization is often multi-objective, that is, involving several conflicting objectives, such as maximizing performance of a network while minimizing its inference time.

Multi-objective problems are commonly addressed via Evolutionary Algorithms (EAs). In an EA, several solutions (a population) are optimized simultaneously, making them natural candidates to search for sets of solutions that represent different trade-offs in multi-objective problems. Other advantages of EAs are their ability to tackle large search spaces and the ease with which they can be parallelized (which is important for practical usage on modern hardware). Additionally, in order to reduce the inefficiency of EAs in terms of the number of evaluations of the objective function(s) required to reach convergence, these algorithms can be hybridized with approaches such as Bayesian optimization that can achieve excellent results within a budget of only a few evaluations.

The main goal of this thesis is to explore how EAs can be leveraged to perform hyperparameter optimization for deep learning effectively (so that the resulting networks achieve excellent performance) and efficiently (so that minimal computational effort would be required).

In Chapter 2, we build upon a prior efficient approach to NAS that searches for architectures by pretraining a single *supernetwork*. A supernetwork encompasses many possible architectures, which can be cheaply evaluated once the supernetwork has been trained. However, relying on a single supernetwork limits the diversity of solutions that can be discovered. Therefore, we experiment with using five different supernetworks to create cascades of multiple architectures in order to achieve better multi-objective trade-offs between image classification accuracy and the computational footprint. Our algorithm, ENCAS, outperforms state-of-the-art NAS and cascade search approaches on several datasets.

Our experience with supernetworks showed that there is a limited variety of architectures that can be discovered even if several supernetworks are considered, making the approach less attractive despite its efficiency. In Chapter 3, we propose an alternative, less limiting, approach to evolutionary NAS that remains relatively efficient. Specifically, we extend the Population Based Training (PBT) EA that is designed for optimizing non-architecture hyperparameters. PBT achieves efficiency via dynamic optimization: the networks are trained for a short time before poorly-performing solutions are replaced with copies of well-performing ones, and the hyperparameters are copied and perturbed. The efficiency is gained by continuing training the previous partially trained weights rather than restarting from scratch. A barrier to applying PBT to NAS is that weights cannot generally be reused across architectures. We address this issue by introducing crossover, in which a new architecture is created by mixing two well-performing architectures and inheriting the weights for each operation in the new architecture from the corresponding parent. Additionally, the weights are adapted (by reducing their magnitude and adding noise) to prevent getting stuck in local optima while still retaining previous information, which strongly improves the results in challenging settings of reinforcement learning and image generation.

Our extension of PBT to NAS in Chapter 3 relied on the standard PBT algorithm. However, several PBT variants have been published and reported as outperforming standard PBT and (a subset of) each other. Since no clear comparison between all the prominent variants was available, we ventured to carry out such a comparison ourselves in Chapter 4. We evaluate five PBT variants on a set of image classification and reinforcement learning tasks, finding that no PBT variant can consistently outperform all others. Additionally, we determine that some of the performance differences can be explained by the different degrees of greediness exhibited by the algorithms. We further observe that Bayesian PBT variants can achieve worse performance than even the standard PBT, which can be attributed to the effectiveness of their exploration making them greedier. Analyzing the theory underlying the Bayesian PBT variants, we find that they are guaranteed to asymptotically approach not the best solution, as previously postulated, but the greediest one.

One of the findings of Chapter 4 is that the step size of PBT variants can modulate their greediness and strongly influence their ultimate performance. Consequently, in Chapter 5, we develop a PBT variant that adjusts its step size automatically. Towards that goal, a stagnation detection mechanism is proposed, which, when triggered, causes a partial restart. Upon a restart, the step size is increased, the information about previously explored hyperparameters is used by a meta Bayesian optimization procedure to set new hyperparameters, and the partially-trained weights are adapted (similarly to Chapter 3, by

reducing their magnitude and adding noise). The newly proposed algorithm, on average, performs better than (or similarly to) five existing PBT variants with *tuned* step sizes and other baselines on eight image classification and reinforcement learning tasks.

Throughout this thesis, we consider how to efficiently optimize hyperparameters but any optimization incurs costs and adds complexity. Thus, arguably, the best hyperparameter optimization is no optimization at all. Avoiding optimization may be feasible under specific circumstances, such as those in Chapter 6, where distributed learning for medical image segmentation is explored. Image segmentation entails finding parts of an image that correspond to specific regions of interest (e.g., human organs in an MRI scan). For *medical* image segmentation, it can be difficult to gather a sufficiently large and diverse dataset without running into privacy concerns. An established alternative to collecting sensitive data is federated learning, where data is used for distributed training without sharing. We follow a different approach where *synthetic* data is generated at each hospital independently and then pooled together to be used for pretraining a general model. This model is distributed back to each hospital and fine-tuned on its specific data. The advantages of this approach include not requiring all hospitals to be online simultaneously, to share infrastructure, or to redo local computation when a new participant is added to the system. However, the neural networks for data synthesis and segmentation need to be trained at each hospital, potentially requiring either human intervention or automatic hyperparameter adaptation. We build upon prior methods known for robustness and introduce an approach that uses additional heuristics for hyperparameter adaptation to the properties of the data. Our approach is evaluated on three diverse datasets and found to perform well out-of-the-box, with pretraining on pooled synthetic data improving robustness to data shifts and performance on challenging data.

To summarize, this thesis is focused on efficient hyperparameter optimization for deep learning, largely via the design and application of EAs. Optimization of both the training hyperparameters and the architecture is addressed, as well as adapting hyperparameters without optimization. We conclude that evolutionary principles can be productively used to efficiently optimize hyperparameters and architectures of neural networks, thus helping accelerate their research and development (which could be further aided by making usage of hyperparameter optimization algorithms more convenient, as well as extending them to handle open-ended search spaces and very large models).

Samenvatting

Kunstmatige intelligentie (AI) is een idee, een geheel van onderzoekssubvelden en uiteindelijk een reeks technologieën die de wereld hervormen. AI-systemen zijn bedoeld om problemen op te lossen die anders biologische of menselijke intelligentie zouden vereisen. Sinds het midden van de jaren 2010 hebben doorbraken in het AI-subveld *deep learning* snelle vooruitgang mogelijk gemaakt in computervisie, natuurlijke taalverwerking, generatieve modelleren en andere gebieden.

Het kernidee van deep learning is om voldoende grote hoeveelheden data te gebruiken om de parameters van een *neuraal netwerk* zo in te stellen dat het goed presteert op een doeltaak. Een neuraal netwerk is een gerichte graaf van geparametriseerde operaties die een numerieke invoer omzetten in een numerieke uitvoer. De parameters van een netwerk kunnen worden geoptimaliseerd via gradiëntgebaseerde technieken, in tegenstelling tot de *hyper*parameters, die vaak handmatig door een expert worden ingesteld. Typische categorieën van hyperparameters zijn de instellingen van de gradiëntgebaseerde optimalisator, de keuze van operaties die in het netwerk worden gebruikt, en de structuur van de computationele graaf. De laatste twee worden gewoonlijk aangeduid als de *architectuur* van een netwerk, en het optimaliseren daarvan wordt Neural Architecture Search (NAS) genoemd.

Hyperparameters kunnen zowel de prestatie van een netwerk op de doeltaak als de efficiëntie ervan sterk beïnvloeden. Daarom is het belangrijk om goede waarden voor de hyperparameters te vinden. Het doel van hyperparameteroptimalisatie-algoritmen is om dit proces te automatiseren, wat in de context van deep learning om verschillende redenen uitdagend is. Ten eerste moet, om te evalueren hoe goed een verzameling hyperparameterwaarden is, een netwerk doorgaans worden getraind, wat tijd en dure hardware kost, en daarmee beperkt hoeveel hyperparameterconfiguraties binnen een eindig budget kunnen worden geëvalueerd. Ten tweede vereisen neurale netwerken dat veel hyperparameters worden ingesteld, waarbij elke hyperparameter vele mogelijke waarden kan aannemen die op niet-triviale wijze interageren met de waarden van andere hyperparameters. Dit leidt tot grote zoekruimtes die moeilijk te optimaliseren zijn. Ten slotte is hyperparameteroptimalisatie vaak multi-objectief, dat wil zeggen dat er meerdere, conflicterende doelstellingen spelen, zoals het maximaliseren van de prestatie van een netwerk terwijl de inferentietijd wordt geminimaliseerd.

Multi-objectieve problemen worden vaak aangepakt met behulp van Evolutionaire Algoritmen (EA's). In een EA worden meerdere oplossingen (een populatie) gelijktijdig geoptimaliseerd, wat ze natuurlijke kandidaten maakt om te zoeken naar verzamelingen van oplossingen die verschillende afwegingen in multi-objectieve problemen representeren. Andere voordelen van EA's zijn hun vermogen om grote zoekruimtes aan te pakken en de eenvoud waarmee ze kunnen worden geparalleliseerd (wat belangrijk is voor praktisch gebruik op moderne hardware). Om de inefficiëntie van EA's in termen van het aantal evaluaties van de doelfunctie(s) dat nodig is om tot convergentie te komen te vermin-

deren, kunnen deze algoritmen bovendien worden gehybridiseerd met benaderingen zoals Bayesiaanse optimalisatie, die uitstekende resultaten kunnen behalen met slechts een klein aantal evaluaties.

Het hoofddoel van dit proefschrift is te onderzoeken hoe EA's kunnen worden ingezet om hyperparameteroptimalisatie voor deep learning zowel effectief (zodat de resulterende netwerken uitstekende prestaties behalen) als efficiënt (zodat minimale rekeninspanning nodig is) uit te voeren.

In Hoofdstuk 2 bouwen we voort op een eerdere efficiënte benadering van NAS, die naar architecturen zoekt door één enkel *supernetwerk* te pretrainen. Een supernetwerk omvat vele mogelijke architecturen, die na het trainen van het supernetwerk goedkoop kunnen worden geëvalueerd. Het gebruik van één enkel supernetwerk beperkt echter de diversiteit van oplossingen die kunnen worden gevonden. Daarom experimenteren we met het gebruik van vijf verschillende supernetwerken om cascades van meerdere architecturen te creëren, met als doel betere multi-objectieve afwegingen te realiseren tussen nauwkeurigheid van beeldclassificatie en de computationele last. Ons algoritme, ENCAS, presteert beter dan state-of-the-art NAS- en cascade-zoekbenaderingen op verschillende datasets.

Onze ervaring met supernetwerken liet zien dat er slechts een beperkte variëteit aan architecturen kan worden ontdekt, zelfs als meerdere supernetwerken worden overwogen, wat de aanpak minder aantrekkelijk maakt ondanks de efficiëntie ervan. In Hoofdstuk 3 stellen we een alternatieve, minder beperkende benadering van evolutionaire NAS voor die relatief efficiënt blijft. Meer specifiek breiden we het Population Based Training (PBT) EA uit, dat ontworpen is voor het optimaliseren van niet-architectuur-hyperparameters. PBT bereikt efficiëntie via dynamische optimalisatie: de netwerken worden gedurende korte tijd getraind voordat slecht presterende oplossingen worden vervangen door kopieën van goed presterende oplossingen, waarbij de hyperparameters worden gekopieerd en geperturbeerd. De efficiëntie wordt verkregen door verder te trainen vanaf de eerder gedeeltelijk getrainde gewichten in plaats van steeds vanaf nul te beginnen. Een obstakel voor het toepassen van PBT op NAS is dat gewichten in het algemeen niet kunnen worden hergebruikt tussen architecturen. We pakken dit probleem aan door crossover te introduceren, waarbij een nieuwe architectuur wordt gecreëerd door twee goed presterende architecturen te mengen en de gewichten voor elke operatie in de nieuwe architectuur te laten erven van de corresponderende ouder. Bovendien worden de gewichten aangepast (door hun grootte te verkleinen en ruis toe te voegen) om te voorkomen dat ze vastlopen in lokale optima, terwijl eerdere informatie behouden blijft. Dit verbetert de resultaten sterk in uitdagende settings van reinforcement learning en beeldgeneratie.

Onze uitbreiding van PBT naar NAS in Hoofdstuk 3 was gebaseerd op de standaard PBT-algoritme. Er zijn echter verschillende PBT-varianten gepubliceerd, waarvan is gerapporteerd dat zij de standaard PBT en (een deelverzameling van) elkaar overtreffen. Aangezien er geen duidelijke vergelijking tussen alle prominente varianten beschikbaar was, hebben wij in Hoofdstuk 4 zelf een dergelijke vergelijking uitgevoerd. We evalueren vijf PBT-varianten op een verzameling beeldclassificatie- en reinforcement-learningtaken en constateren dat geen enkele PBT-variant alle andere consequent kan overtreffen. Daarnaast tonen we aan dat een deel van de prestatieverschillen kan worden verklaard door de verschillende mate van greediness die de algoritmen vertonen. We observeren

verder dat Bayesiaanse PBT-varianten slechtere prestaties kunnen leveren dan zelfs de standaard PBT, wat kan worden toegeschreven aan de effectiviteit van hun exploratie, die hen greedier maakt. Door de theorie achter de Bayesiaanse PBT-varianten te analyseren, constateren we dat zij asymptotisch niet de beste oplossing naderen, zoals eerder werd verondersteld, maar de oplossing die het meest greedy is.

Een van de bevindingen van Hoofdstuk 4 is dat de stapgrootte van PBT-varianten hun greediness kan moduleren en hun uiteindelijke prestaties sterk kan beïnvloeden. Navenant ontwikkelen we in Hoofdstuk 5 een PBT-variant die zijn stapgrootte automatisch aanpast. Met dat doel voor ogen wordt een stagnatiedetectiemechanisme voorgesteld dat, wanneer geactiveerd, een gedeeltelijke herstart veroorzaakt. Bij zo'n herstart wordt de stapgrootte vergroot, wordt informatie over eerder verkende hyperparameters gebruikt door een meta-Bayesiaanse optimalisatieprocedure om nieuwe hyperparameters in te stellen, en worden de gedeeltelijk getrainde gewichten aangepast (vergelijkbaar met Hoofdstuk 3, door hun grootte te verkleinen en ruis toe te voegen). Het voorgestelde algoritme presteert gemiddeld beter dan (of vergelijkbaar met) vijf bestaande PBT-varianten met *getunede* stapgroottes en andere baselines op acht beeldclassificatie- en reinforcement-learningtaken.

In dit proefschrift onderzoeken we hoe hyperparameters efficiënt kunnen worden geoptimaliseerd, maar elke vorm van optimalisatie brengt kosten en extra complexiteit met zich mee. Men zou dus kunnen stellen dat de beste hyperparameteroptimalisatie helemaal geen optimalisatie is. Het vermijden van optimalisatie kan haalbaar zijn onder specifieke omstandigheden, zoals die in Hoofdstuk 6, waar gedistribueerd leren voor medische beeldsegmentatie wordt onderzocht. Beeldsegmentatie houdt in dat delen van een beeld worden gevonden die overeenkomen met specifieke interessegebieden (bijvoorbeeld menselijke organen in een MRI-scan). Voor *medische* beeldsegmentatie kan het moeilijk zijn om een voldoende grote en diverse dataset te verzamelen zonder tegen privacykwesties aan te lopen. Een gevestigde alternatieve aanpak voor het verzamelen van gevoelige data is federatief leren, waarbij data wordt gebruikt voor gedistribueerde training zonder te worden gedeeld. Wij volgen een andere benadering, waarbij *synthetische* data onafhankelijk bij elk ziekenhuis wordt gegenereerd en vervolgens wordt samengevoegd om te worden gebruikt voor het pretrainen van een algemeen model. Dit model wordt vervolgens weer naar elk ziekenhuis verspreid en daar gefinetuned op de bijbehorende specifieke data. De voordelen van deze aanpak zijn onder meer dat niet alle ziekenhuizen tegelijkertijd online hoeven te zijn, geen infrastructuur hoeven te delen en lokale berekeningen niet hoeven te worden herhaald wanneer een nieuwe deelnemer aan het systeem wordt toegevoegd. De neurale netwerken voor datasynthese en segmentatie moeten echter bij elk ziekenhuis worden getraind, wat mogelijk menselijke interventie of automatische hyperparameteradaptatie vereist. We bouwen voort op eerdere methoden die bekend staan om hun robuustheid en introduceren een benadering die extra heuristieken gebruikt voor hyperparameteradaptatie aan de eigenschappen van de data. Onze aanpak wordt geëvalueerd op drie diverse datasets en blijkt out-of-the-box goed te presteren, waarbij pretraining op samengevoegde synthetische data de robuustheid tegen data shifts en de prestaties op uitdagende data verbetert.

Samenvattend richt dit proefschrift zich op efficiënte hyperparameteroptimalisatie voor deep learning, grotendeels via het ontwerp en de toepassing van EA's. Zowel de

optimalisatie van trainingshyperparameters als van de architectuur wordt behandeld, evenals het aanpassen van hyperparameters zonder optimalisatie. We concluderen dat evolutionaire principes productief kunnen worden ingezet om hyperparameters en architecturen van neurale netwerken efficiënt te optimaliseren, en zo kunnen bijdragen aan het versnellen van onderzoek en ontwikkeling op dit gebied (wat verder kan worden ondersteund door het gebruik van hyperparameteroptimalisatie-algoritmen te vereenvoudigen en ze uit te breiden naar open-ended zoekruimtes en zeer grote modellen).

1

Introduction

Artificial Intelligence (AI) is a broad research field aimed at building artificial systems for solving problems that would ordinarily require biological or human intelligence. The technologies resulting from the research in this field are also commonly called "AI". AI is widely applied in the modern world, with various forms of it used to diagnose disease (Abràmoff et al., 2018; Martinez-Gutierrez et al., 2023), optimize hardware (McDermott, 1982; Mirhoseini et al., 2021), play games (Campbell et al., 2002; Silver et al., 2018), and address many other problems (Jordan & Mitchell, 2015).

AI as a technology lacks a strict definition, with the products of different subfields being more or less synonymous with the term over the course of the last decades. Leaving aside many aspects of AI, in this thesis we focus on optimization (Section 1.1), learning (Section 1.2), and using the former to improve the latter (Sections 1.3 and 1.4).

1.1. Optimization

1.1.1. What is optimization?

Optimization entails maximizing (or minimizing) an objective by changing the decision variables on which it depends. For example, a drug's efficacy in curing a disease can be maximized by changing its molecular structure.

Formally, let $f : X \to \mathbb{R}$ be a function defined over a search space X that may include categorical, integer, and real-valued variables. A single-objective unconstrained optimization problem can be formulated as finding a solution $x^* \in X$ that maximizes the value of f:

$$x^* = \arg\max_{x \in X} f(x). \tag{1.1}$$

Many optimization problems are multi-objective, meaning that several conflicting objectives need to be optimized jointly. In the drug discovery example, while the drug's efficacy should be maximized, the severity and frequency of side effects should be minimized (minimization can be turned into maximization by negating the objective).

Formally, an L-objective optimization problem can be written as follows:

1

$$x^* = \arg\max_{x \in X} \boldsymbol{f}(x) = \arg\max_{x \in X} (f_1(x), f_2(x), \ldots, f_L(x)).$$

Since the objectives are conflicting, no single best solution usually exists, with different solutions achieving different trade-offs between objectives. However, some solutions can be strictly better than others, i.e., dominate them. A solution x_1 dominates x_2 if $\forall i \in \{1 \ldots L\}, f_i(x_1) \geq f_i(x_2)$ and $\exists i \in \{1 \ldots L\} : f_i(x_1) > f_i(x_2)$. Multi-objective optimization typically produces a set of non-dominated solutions, which is referred to as a trade-off front (or a Pareto front if it is globally optimal). A visualization is provided in Figure 1.1.

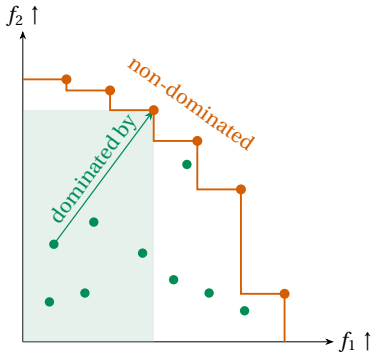


Figure 1.1: A trade-off front (orange points) for two objectives, with dominated points in green. The dominated region for one point on the trade-off front is shaded in light green. The orange line is the border of the space dominated by the points on the trade-off front.

1.1.2. Black-, gray-, white-box optimization

An important factor in optimization is the extent of available information about the objective function. In Equation (1.1), nothing is assumed to be known about f aside from the domain of its inputs. Such a scenario is called *black-box* optimization: in it, f is treated as an unknown mechanism that produces an output given the inputs. In this case, an optimization algorithm can only evaluate various inputs (more or less intelligently) until one corresponding to a satisfactory output is found, or the evaluation budget is exhausted.

If some additional information about f is available, the problem can be cast as *gray-box*. A gray-box setting relevant to this thesis is dynamic optimization, where the function f is additionally parameterized by time $t = 1, \ldots, T$. The goal is to maximize the final performance $f_T(x)$, but it is possible to partially evaluate x for $t' < T$ steps, producing intermediate results $f_1(x), \ldots, f_{t'}(x)$. If these intermediate results contain useful information about the final one, this information could be leveraged by the optimization algorithm to solve the problem more efficiently.

Finally, *white-box* optimization describes a setting where everything about the problem is known and understood. For the case of real-valued inputs and outputs, this entails that the analytical expression of f is known and a gradient of the objective with respect to the parameters can be computed. This allows the internal parameters of f to be directly

adjusted to maximize its value (by following the gradient), making this setting much more amenable to efficient optimization than black- or gray-box. However, in practice, the gradients can not always be computed, necessitating usage of black- and gray-box optimization algorithms.

1.1.3. Classes of optimization algorithms considered in this thesis

Real-valued white-box optimization problems are usually solved via casting them as minimization problems and using *gradient descent* or its variants. Gradient descent minimizes a function that has real-valued inputs by computing the gradient and taking a small step in the opposite direction: $\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$, where t denotes the number of the step and η is the *learning rate* that modulates the step size. There is a variety of improvements to this fundamental algorithm that introduce momentum (Polyak, 1964), efficiently compute second-order derivatives (Broyden, 1970), and include other changes (Nesterov, 1983; Duchi et al., 2011; Kingma & Ba, 2015).

Bayesian Optimization (BO) algorithms are designed for black-box optimization with expensive function evaluations. It is typically assumed that only a small number of evaluations (between 10 and 1000) can be performed. While gradient descent iteratively improves a single solution, a BO algorithm can choose a new solution to evaluate that is far from the previous ones. Key concepts of BO are the surrogate (a function approximating the unknown target function and the uncertainty around the approximation) and the acquisition function (a function defining for each solution how promising it is as the next evaluation point). BO operates by fitting the surrogate to the previous evaluation results, optimizing the acquisition function on the surrogate to find the best next point to evaluate, evaluating this point, and repeating the process. While efficient in terms of number of evaluations, BO requires additional computation (to fit the surrogate and optimize the acquisition function) (Snoek et al., 2012) and can run into difficulties with high-dimensional problems (Duque et al., 2024) and parallelizability (Song et al., 2024).

In contrast to BO algorithms, *Evolutionary Algorithms* (EAs) are evolution-inspired black- and gray-box optimization algorithms that are *inherently parallel*. In an EA, a *population* of several solutions is optimized simultaneously via operations that implement the two key procedures of EAs: selection and variation. Selection means removing poorly-performing solutions and propagating well-performing ones. Variation entails changing a solution, e.g., via random perturbation of the values or via recombining several solutions. EAs are a good fit for solving multi-objective problems (where the goal is to find a trade-off front), as they optimize a set of solutions, making them naturally suited to collectively explore different trade-offs between the objectives. EAs have also been shown to successfully address gray-box, dynamic, expensive, and other problems (Van Veldhuizen & Lamont, 1998; Branke & Schmeck, 2003; Singh & Deb, 2006; Bouter et al., 2017).

1.2. Learning

1.2.1. What is learning?

Learning is a rather vague concept that evades an exact all-encompassing definition. For the purpose of this thesis, we rely on the definition of Mitchell (1997): “a computer program learns from experience E with respect to some class of tasks C and performance

measure P if its performance at tasks in C, as measured by P, improves with experience E". In other words, learning is making use of experience to perform better on a specific class of tasks.

We restrict the definition from a computer program to a parametric function $f_\theta : X \to Y$, where X and Y are the input and the output spaces (that are finite-dimensional and may include a mix of categorical, integer, and real-valued variables). Performance of f_θ (as measured by P) is improved by optimizing the model parameters θ.

An important property of a learning problem is that generalization is expected. For instance, if a system learns to detect a disease in X-ray images, it should make accurate predictions for new, previously unseen images. Failure to do so while making correct predictions on the seen inputs is referred to as *overfitting*. Overfitting can be conceptualized as rote memorization taking place instead of learning. Thus, excellent performance on the seen inputs, while being perfectly satisfactory for optimization tasks, is not acceptable for learning (which motivates its definition referencing a *class* of tasks rather than one fixed deterministic task).

Several types of learning are commonly recognized. In *supervised learning*, a dataset of N inputs and corresponding outputs is available: $D = \{(x_i, y_i)\}_{i=1}^N$. For example, inputs can be X-ray images, and outputs can be binary labels indicating presence or absence of a disease. If outputs are categorical, like in this example, the task is considered to be *classification*. A numerical output (e.g., bone age in years) would make the task *regression*. In *segmentation* tasks, the output has the same resolution as the input, with each output pixel containing information about this location, such as presence of a specific organ.

In each of the above examples, the distribution of X remains the same (the inputs are X-ray images made in a particular environment on a particular patient cohort) while Y changes (e.g., specific categories for classification and positive reals within a range of possible human ages for regression), with the task defining the performance measure (e.g., accuracy for classification and mean squared error for regression).

A *reinforcement learning* problem can be conceptualized as an agent observing an environment (receiving $x \in X$) and executing actions (producing $y \in Y$) that affect the environment, the agent's observations, and its reward. Typically, an agent learns to maximize the discounted sum of rewards over the length of an episode (this serves as the performance measure P), which encourages pursuing both short- and long-term gains. An example of reinforcement learning is controlling appendages of a virtual human-shaped agent that must learn to run with a speed of 10 meters per second, guided by rewards proportional to the difference between its current speed and the target one.

Besides supervised and reinforcement learning, many other kinds of learning can be defined, including unsupervised and self-supervised. We refer an interested reader to Murphy (2022) for further details.

Since generalization is vital for learning, it needs to be carefully measured. In supervised learning, the dataset is often split into training, validation, and test sets[1]. The training set is used to update *parameters* during learning (which is commonly referred to as "training" the "model" f_θ), while the validation set is used to adjust *hyperparameters* (e.g., the class of functions to which f_θ belongs), and the test set is used to estimate the generalization performance of the final model.

[1] The same concept can be applied to, e.g., seeds of a reinforcement learning environment.

Learning and optimization are connected in many ways. Consider the optimization algorithms from Section 1.1.3: in BO, fitting a surrogate on observed data is a form of learning. Similarly, in some EAs, the population can be learned from to improve the variation operator and therefore the efficiency and effectiveness of the optimization procedure (Thierens & Bosman, 2011). Both of these examples demonstrate leveraging learning to improve optimization. On the other hand, optimization lies at the heart of learning, since parameters θ of f_θ are *optimized to improve performance P*. In this thesis, the focus is placed on further leveraging optimization to improve learning — specifically, deep learning.

1.2.2. Deep learning

Deep learning refers to a form of learning where the function f_θ takes the form of a many-layered artificial neural network. Historically, artificial neural networks were inspired by *biological* neural networks (Rosenblatt, 1958). However, the modern artificial neural networks used in the context of AI are not designed to reflect contemporary neuroscientific knowledge. We therefore consider them as mathematical objects and refer to them simply as "neural networks".

A neural network is a tuple of (G, O, Θ) where G is the computational graph (a directed graph defining the interaction of the inputs and the operations of the network), O is the list of operations used in the computational graph, and Θ is the list of parameters of all operations. G and O together are referred to as the *architecture* of the network. In this thesis, only architectures corresponding to directed acyclic graphs are considered.

A neural network processes its input by executing the operations within the computational graph. The simplest computational graph is a chain, which corresponds to sequential execution of the operations. More complex graphs may include divergence into multiple branches, convergence of multiple branches into one, and skip connections that funnel the output of an early operation to be the input of a later one (while skipping intermediate operations) (He et al., 2016).

A typical operation in a neural network is a linear layer that corresponds to a multiplication by a matrix of parameters (also called "weights" of a neural network). Multiplying an input row vector of N values by a matrix with dimensions (N, M) produces M output values. This can be conceptualized as M "neurons", each of which takes N inputs and multiplies them with its own set of N parameters to produce a scalar output. A linear layer, then, corresponds to a parallel execution of a single operation (with different parameters) over all inputs.

Activation functions can be used to introduce non-linearity into the network, thus increasing its representation capability. An activation function (such as a sigmoid or a rectified linear unit (Nair & Hinton, 2010)) is applied element-wise to the output of the preceding layer. Typically, activation functions are interspersed with linear layers. Prior to the application of an activation function, the activations can be normalized (to improve learning dynamics) via, e.g., layer normalization (Ba et al., 2016).

The simplicity and the inherent parallelizability of computation within each layer is a critical idea that contributed to the resurgence and domination of neural networks since the 2010s, as it enabled efficient implementation on Graphics Processing Units (GPUs),

which in turn allowed for increasingly large and therefore capable networks to be created that could learn from increasingly large datasets.

Solving a problem with deep learning requires formalizing the objective, collecting relevant data (or coding an interactive environment), defining the architecture of the network, establishing a training procedure, optimizing the parameters according to this procedure, evaluating the result, and potentially repeating this process. The optimization of parameters is described in Section 1.3.1, the optimization of architecture is addressed in Section 1.3.2, and the optimization of hyperparameters is discussed in Section 1.3.3.

1.3. Optimization for deep learning

1.3.1. Optimizing parameters

The parameters Θ of a neural network are typically optimized via gradient-descent-based methods, since the analytical expression for the network is known and the operations are chosen to be differentiable. The function to be optimized via gradient descent, commonly referred to as the loss function, also needs to be differentiable, and thus it may not exactly correspond to the target performance measure. For example, when classifying images, we may care about accuracy (that is based on discontinuous counting and therefore is non-differentiable) but use a differentiable surrogate such as the cross-entropy loss (Good, 1952; Goodfellow et al., 2016).

During training, the loss function $\ell(\Theta)$ is minimized using the gradient computed on (a random subset of) the training data: $\Theta_{t+1} = \Theta_t - \eta\nabla\ell(\Theta_t)$, where t is the step number, and η is the learning rate. However, since the task is *learning*, we generally do not aim to find parameter values that correspond to minimal *training* loss. On the contrary, the training performance can be sacrificed to improve the generalization performance on the validation and test sets. Introducing methods leading to this is known as *regularization*, which can take form of decaying weight values to zero, applying augmentations to inputs (e.g., randomly rotating images) (Cubuk et al., 2020), and others (Goodfellow et al., 2016). Regularization techniques essentially bias optimization towards solutions with desirable properties, such as generalization instead of memorization. For example, weight decay, by punishing large parameter values, limits the representation power of a network and therefore its ability to memorize the training data (Goodfellow et al., 2016). Similarly, image augmentations such as rotations hinder memorization by randomly varying inputs and pushing optimization to find weights that would make a classifier invariant to these modifications (e.g., a rotated cat should still be classified as a cat).

Parameter optimization is strongly influenced both by the training hyperparameters (such as the regularization hyperparameters, the specific gradient-based method used, and its hyperparameters) and the architecture (which defines the parameters to be optimized in the first place).

1.3.2. Optimizing architecture

As discussed in Section 1.2.2, an architecture of a neural network encompasses both its computational graph and the operations used within. Therefore, to properly define the search space for architecture optimization, one needs to specify what options are available

for both. Architecture optimization is commonly referred to as Neural Architecture Search (NAS) (Zoph & Le, 2017).

A search space can contain many or few operations, with each being more or less complex. Real et al. (2020) include many primitive operations into the search space and allow the algorithm to build upon the useful operations while ignoring the useless ones. A more restrictive (and more efficient) approach entails pre-selecting higher-level operations that are widely used in the literature: e.g., Zoph et al. (2018) included different convolution variants and max pooling into the search space. Finally, a single high-level pre-determined operation can be the only option, with the search space consisting only of its parameters (e.g., an inverted residual block (Sandler et al., 2018) was used by D. Wang et al. (2021b)).

Similar to operations, the computational graphs that are included into a search space can vary in complexity. In the simplest scenario, the graph can be largely predefined, with only minor variations allowed, such as searching for the number of times an operation can be repeated, thus influencing the depth of the network (H. Cai et al., 2020; D. Wang et al., 2021b). A larger search space can have the global structure of the graph fixed but consider it to be constructed of "cells" of limited size, within which the operations can be flexibly connected to each other (Zoph et al., 2018; Real et al., 2019). Finally, no structure can be predefined, enabling the algorithm to construct code out of the available operators without restrictions (Real et al., 2020).

The size and complexity of architecture search spaces varies widely. The inherent trade-off when designing a search space is between the extent to which it is possible to express any architecture and how difficult or time-consuming it is to find a well-performing one. More complex search spaces demand more computational resources, which in the case of NAS can reach tens of thousands of GPU-hours (Zoph & Le, 2017). Therefore, efficiency is a paramount concern.

To improve the efficiency of NAS, an algorithm that achieves better results with fewer architecture evaluations can be designed, an evaluation strategy for cheaper estimation of architecture quality can be developed, or the search space can be modified to be more amenable to efficient search. Improvements along all these lines have been pursued in an intertwined manner in the last years.

Initially, a recurrent neural network has been trained to generate architecture specifications via reinforcement learning based on full training of each architecture (Zoph & Le, 2017). Reducing the cost of evaluation by using proxies (such as training a downscaled version of the architecture) allowed the application of black-box approaches such as EAs (Real et al., 2019) and BO (C. Liu et al., 2018). Predicting long-term performance from short-term results allowed for further improved efficiency (Baker et al., 2018), as did proxy metrics aiming to estimate architecture quality with no training at all (White et al., 2021).

Another influential evaluation strategy is weight sharing via a supernetwork (Pham et al., 2018). A supernetwork is a superset of all possible architectures in a search space. Each specific architecture is a path within the supernetwork. The key idea is that two different architectures may be using the same operation at the same depth. The weights of this operation can then be shared between the architectures, saving the computational effort of independent training from scratch.

Supernetworks enabled efficient gradient-based NAS by applying a continuous relaxation to the search problem (H. Liu et al., 2019). Importantly, this restricts the search space and the potential architecture diversity, as the operations in the search space have to be amenable to gradient-based optimization.

Gradient-based supernetwork NAS is typically a "one-stage" process, since the weights are optimized simultaneously with the architecture. This improves efficiency but further restricts the kinds of architecture that can be found (since an architecture that initially performs poorly but could eventually reach the best result is unlikely to be found, as, based on its initial poor performance, weights associated with its operations are trained less, if at all).

An alternative two-stage supernetwork NAS approach amounts to pretraining a supernetwork without any search, and optimizing the architecture afterwards by a black-box algorithm leveraging the cheap (with no training required) evaluation of the pretrained subnetworks (H. Cai et al., 2020). This decoupled procedure allows for better architectures to be found without unduly biasing the search towards the architectures with good initial but subpar final performance.

A supernetwork upper-bounds the maximum size of a neural network architecture that can be found. This naturally predisposes supernetwork NAS to a multi-objective problem formulation, for example, maximizing performance while minimizing inference latency (H. Cai et al., 2020; D. Wang et al., 2021b).

Multi-objective supernetwork NAS achieved excellent performance-efficiency trade-offs (H. Cai et al., 2020; D. Wang et al., 2021b). However, architectures that can be discovered by a supernetwork NAS algorithm remain limited by the chosen supernetwork and the method used to train it. This motivates the following research question:

Research Question: Chapter 2

Can multiple supernetworks be leveraged to improve performance-efficiency trade-off fronts of neural networks on image classification tasks?

1.3.3. Optimizing hyperparameters

As mentioned previously, the parameters of a neural network are its weights (that are optimized during training (via gradient descent)). Hyperparameters define every other component of the network and its training procedure. Having introduced optimization of architecture hyperparameters in Section 1.3.2, here we focus on the hyperparameters of the training process, such as learning rate or input augmentation. Optimization of architectures, hyperparameters, and any other aspect of a machine learning pipeline is encompassed by the subfield of Automated Machine Learning (AutoML) (Hutter et al., 2019).

Traditionally, an algorithm would suggest hyperparameters that would then be used to fully train a model from scratch (Snoek et al., 2012). The simplest popular algorithm is grid search, where for each hyperparameter, several potential values are defined, and then all combinations of values of all hyperparameters are evaluated. Bergstra & Bengio (2012) showed that for deep learning, random search (where the hyperparameters are sampled randomly) is preferable. The best overall results with the fewest evaluations can

be found by BO algorithms (Snoek et al., 2012; Cowen-Rivers et al., 2022), although they may require careful setup and may struggle with high-dimensional search spaces.

Since some hyperparameter values can lead to poor performance early on, full training is not always necessary. As such, hyperparameter optimization can be cast from a black-box problem where only the final performance is optimized to a gray-box problem where performance at intermediate time points is available. Algorithms leveraging this information are commonly referred to as "multi-fidelity" algorithms (as the intermediate performance is only a partially precise estimate of the final performance) (Won et al., 2025). For example, in successive halving (Jamieson & Talwalkar, 2016), many networks are trained in parallel with different hyperparameters for a short time. The worst-performing fraction of them (half by default) is discarded, while the remaining networks continue training for twice as long, after which the procedure is repeated. Extensions of successive halving introduce asynchrony (Liam Li et al., 2020) and improved hyperparameter sampling via BO (Falkner et al., 2018) or EAs (Awad et al., 2021).

Multi-fidelity algorithms leverage information about intermediate results for efficiency but can produce only *static* hyperparameter values: one value per hyperparameter for the entire training run. However, dynamically changing hyperparameters has been shown to greatly improve performance in some settings (Loshchilov & Hutter, 2017; Tan & Le, 2021). Population Based Training (PBT) (Jaderberg et al., 2017) is an algorithm that is capable of efficiently optimizing hyperparameter values in a dynamic manner.

PBT is a gray-box EA that leverages information about intermediate performance differently to how multi-fidelity algorithms do it. PBT starts with multiple networks with different hyperparameters (a population) trained in parallel for several steps. After evaluation, a specific fraction of the worst-performing networks is discarded. Crucially, their weights are replaced with the *partially-trained* weights of well-performing networks, after which new hyperparameters are created by copying and perturbing the hyperparameters of these networks. The weight reuse leads not only to increased efficiency but also to an ability to optimize a schedule, since the partially-trained weights continue to be trained with modified hyperparameters. Thus, during the course of optimization, the weights are trained with an evolving schedule of hyperparameters.

PBT achieved excellent results in various hyperparameter optimization scenarios (Jaderberg et al., 2017). Yet by its nature, it is not directly applicable to architecture optimization, since it is not clear how the partially-trained weights could be reused in a new architecture (as it may contain a previously-unused operation requiring weights that are incompatible with the partially-trained ones). Several methods have been proposed (Franke et al., 2021; Wan et al., 2022) that were limited to relatively simple search spaces (i.e., including only a few options for architectures of no more than five layers) and reinforcement learning tasks, prompting us to pose the following research question:

Research Question: Chapter 3

Can PBT be extended from the optimization of training hyperparameters to general-purpose NAS in challenging search spaces?

In the context of optimizing hyperparameters of the training procedure, PBT has been extended in multiple ways. For instance, increasingly intricate BO components

have been introduced (Parker-Holder et al., 2020; Parker-Holder et al., 2021; Wan et al., 2022). Another extension, FIRE-PBT (Dalibard & Jaderberg, 2021), combats the greediness of PBT by introducing several subpopulations in which an objective function targeting long-term performance is optimized. Unfortunately, PBT variants have not always been clearly compared to each other: the selection of algorithms to compare to, as well as the underlying tasks to compare on, has not stayed consistent across publications, causing difficulties in understanding relative performance of the algorithms. To address this issue, the following research question is posed:

Research Question: Chapter 4

How do the PBT variants perform relative to each other on image classification and reinforcement learning tasks, and what are the limitations of these algorithms?

Step size is an important hyperparameter of any PBT variant: it defines for how many parameter optimization steps the underlying model is trained before it is evaluated and potentially replaced. Setting the step size too small could increase algorithm greediness while setting it too large reduces how much optimization an algorithm can do. The lack of a clear general rule of how to set this hyperparameter reduces efficiency and applicability of PBT variants, as potentially repeated execution of the algorithm with different step sizes can be required. This motivates us to ask:

Research Question: Chapter 5

Can a PBT variant be developed where the step size is automatically adjusted within a single run in an efficient manner?

1.4. Distributed learning via synthetic data

Deep learning requires data to succeed, with larger datasets leading to better performance (C. Sun et al., 2017). However, some kinds of data may be difficult to collect and utilize. An important case where data exists but is not straightforwardly available for learning is privacy-sensitive data, such as medical images in hospitals (Rieke et al., 2020). Access to this data is gated by bureaucratic procedures that can be difficult to overcome, thus reducing innovation and slowing down progress in finding cures and improving care.

The standard approach of making use of such data is federated learning (Konečný et al., 2016; Rieke et al., 2020; Adnan et al., 2022). In federated learning, a neural network is trained in a distributed way by simultaneously utilizing data in multiple locations to update the weights. Instead of sharing the original data, the weights of the models (or their gradients) are shared. Downsides of federated learning include the necessity of shared infrastructure and overall coordination between all the participants. At the same time, privacy concerns remain, as even gradients may leak enough information to reconstruct private data (Zhu et al., 2019).

A promising alternative is sharing *synthetic* data that faithfully reproduces important properties of real data while not inducing privacy risks (Goncalves et al., 2020). For a multi-site collaboration (where a site is, e.g., a medical center), synthetic data could be

independently and asynchronously generated at each site, and shared, allowing the data to be combined and utilized to train a more capable deep learning model than each site could on its own without requiring shared infrastructure for synchronous federated learning.

In order to create a synthetic dataset at each site, an appropriate generative network needs to be trained on the local data. This is likely to require adapting hyperparameters to the characteristics and the quantities of the available data. However, small dataset sizes and limited computation power at each site can make hyperparameter *optimization* infeasible (due to overfitting and computational requirements, respectively). However, hyperparameter *optimization* is merely a tool for automatically adjusting hyperparameters, not an end goal in itself. Arguably, the most efficient optimization is the one that is not run. Thus, the potential importance of automatic hyperparameter adaptation for distributed learning via synthetic data prompts us to pose the following, and final, research question in this thesis:

Research Question: Chapter 6

Can a distributed and asynchronous learning system based on synthetic data be constructed that would be applicable to varying types and quantities of data without manual hyperparameter adjustment?

2

Evolutionary Neural Cascade Search across Supernetworks

To achieve excellent performance with modern neural networks, having the right network architecture is important. Neural Architecture Search (NAS) concerns the automatic discovery of task-specific network architectures. Modern NAS approaches leverage supernetworks whose subnetworks encode candidate neural network architectures. These subnetworks can be trained simultaneously, removing the need to train each network from scratch, thereby increasing the efficiency of NAS.

A recent method called Neural Architecture Transfer (NAT) further improves the efficiency of NAS for computer vision tasks by using a multi-objective evolutionary algorithm to find high-quality subnetworks of a supernetwork pretrained on ImageNet. Building upon NAT, we introduce ENCAS — Evolutionary Neural Cascade Search. ENCAS can be used to search over multiple pretrained supernetworks to achieve a trade-off front of cascades of different neural network architectures, maximizing accuracy while minimizing FLOPs count.

We test ENCAS on common computer vision benchmarks (CIFAR-10, CIFAR-100, ImageNet) and achieve Pareto dominance over previous state-of-the-art NAS models up to 1.5 GFLOPs. Additionally, applying ENCAS to a pool of 518 publicly available ImageNet classifiers leads to Pareto dominance in all computation regimes and to increasing the maximum accuracy from 88.6% to 89.0%, accompanied by an 18% decrease in computation effort from 362 to 296 GFLOPs.

The contents of this chapter are based on the following publication: Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2022. Evolutionary Neural Cascade Search Across Supernetworks. In *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022,* 1038–1047. ACM. https://doi.org/10.1145/3512290.3528749.

2.1. Introduction

In recent years, deep neural networks have been successfully applied in domains ranging from text summarization (Brown et al., 2020) to medical image segmentation (Isensee et al., 2021). Much of this success has been enabled by task-specific neural network architectures that are designed manually while making use of expert knowledge. The research direction of Neural Architecture Search (NAS) (Zoph & Le, 2017) has the goal of making architecture design automatic and data-driven. Tremendous progress has been made since the first approaches: performance of the found models improved (H. Liu et al., 2019; Sharma et al., 2020), their size decreased (Pham et al., 2018; Dong & Yang, 2019a) (smaller models usually work faster and require less storage space), and the search process itself became much more efficient (with required GPU-hours decreasing from tens of thousands (Real et al., 2019) to single-digit numbers (Stamoulis et al., 2019)).

These search efficiency gains can mostly be attributed to the idea of weight sharing via a supernetwork (Pham et al., 2018) (with performance prediction (Baker et al., 2018; White et al., 2021) also playing a role). Instead of training the weights of each candidate architecture from scratch, a supernetwork is constructed such that each architecture in the search space is a subset of the supernetwork (see Figure 2.1). To evaluate the quality of an architecture, the weights of the relevant part of the supernetwork are copied. With various architectures potentially sharing the same operations (e.g., convolution (LeCun, Bengio, et al., 1995), attention (Bahdanau et al., 2015)), the amount of training needed decreases drastically.

However, by requiring that each architecture is a path within a supernetwork, the supernetwork approach inherently limits the diversity of architectures that can be produced. Thus, the manual choice of which supernetwork to use for the automated NAS procedure plays a large role, as it restricts the search space before the search algorithm even starts. With the growing number of available supernetworks, the issue of choosing the supernetwork is becoming increasingly important, and yet, to the best of our knowledge, there exists no method taking it into account.

There are many ways to improve neural network performance that are different from NAS. Ensembling (Rokach, 2010) is one such technique that involves passing the same input through several different models and combining their predictions to get a better final prediction. It has been shown to work well if the models' mistakes are independent (Esposito & Saitta, 2003), which is helped by the models being different from each other (Wichard et al., 2002). NAS has been used (M. Chen et al., 2021a; Zaidi et al., 2021) to produce models that together make a good ensemble. Modern supernetwork-based approaches seem very fitting to this purpose because they do not incur additional training costs for ensembles of arbitrarily large size (once a supernetwork is trained, weights for trained subnetworks can be extracted from it and used without additional retraining[1]).

Cascading (Viola & Jones, 2001) is a particular case of ensembling with the focus on efficiency: whereas an ensemble requires that every input is processed by every model, a cascade proceeds in a sequential manner, invoking a larger model only if the predictions of smaller models are not confident enough (judged by the confidence function, see Section 2.2.3 for details). Thus, easy inputs consume less computational resources,

[1] This holds for modern state-of-the-art approaches (H. Cai et al., 2020; Sharma et al., 2020; D. Wang et al., 2021b) but does not hold for all, especially older, approaches (H. Liu et al., 2019; Stamoulis et al., 2019).

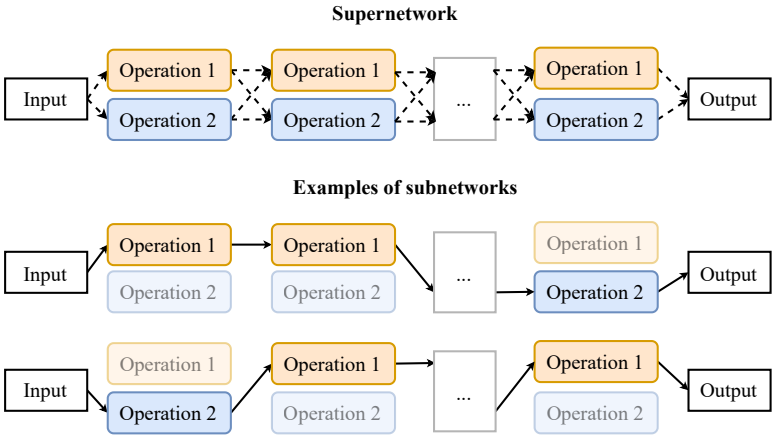


Figure 2.1: A schematic of a generic supernetwork.

while harder inputs can still be predicted well by utilizing every model in the cascade. Despite the potential of cascades to produce efficient and effective models, they remain underexplored in the context of deep learning (X. Wang et al., 2020), and in particular no work has yet been done on combining NAS with cascade search.

To perform any kind of NAS, efficient search algorithms are indispensable. Evolutionary Algorithms (EAs) are particularly fit to the task, as they do not require the search space to be continuous, are known to solve real-world problems efficiently (Chiong et al., 2012), and excel in solving multi-objective and dynamic problems (Van Veldhuizen & Lamont, 1998; Branke & Schmeck, 2003; Singh & Deb, 2006).

Driven by the observations above, in this chapter we present an algorithm called Evolutionary Neural CAscade Search (ENCAS). ENCAS is supernetwork-based and designed to take advantage of various pretrained supernetworks. ENCAS can search over a user-specified set of arbitrary supernetworks that may have e.g., different operations or numbers of layers (the only restriction being that subnetworks extracted from a supernetwork require no retraining). Our algorithm is multi-objective with the goal of finding models on the optimal effectivity-efficiency trade-off front.

The main contributions of the chapter are threefold:

- This work is the first to research the combination of NAS and cascades.
- This work is the first to investigate the feasibility of using several different supernetworks in NAS. We explore whether the additional diversity they provide is helpful for creating cascades.
- The ENCAS algorithm is introduced to search for efficient and effective cascades.

2.2. Related work

2.2.1. Neural architecture search

The first methods to learn neural network architectures trained each candidate architecture from scratch, taking tens of thousands of GPU hours (Zoph & Le, 2017). Pham et al. (2018) introduced the ideas of weight sharing and supernetworks, which drastically decreased search costs. Multiple algorithms followed, most famously DARTS (H. Liu et al., 2019) that reformulated the problem of operation selection as a continuous one, achieving great efficiency.

However, supernetwork weights were discarded after NAS, with the final model being retrained from scratch (because it led to better results). This becomes costly when more than one model is required, e.g., considering both server-based and smartphone-based deployment. OnceForAll (OFA) (H. Cai et al., 2020) introduced an algorithm for training supernetwork weights such that they could be used in extracted subnetworks without any further training. AttentiveNAS (D. Wang et al., 2021b) and AlphaNet (D. Wang et al., 2021a) introduced training techniques that lead to even better performance (albeit using a different search space).

Neural Architecture Transfer (NAT) (Lu et al., 2021) is an approach for fine-tuning a pretrained supernetwork for smaller datasets. The key difference from the previous approaches is that the architectures are adapted together with the weights in a multi-objective search procedure, trading off size and accuracy. The main idea is training only subnetworks close to the currently known trade-off front. To this end, a population of networks is evolved by Non-dominated Sorting Genetic Algorithm III (NSGA-III) (Deb & Jain, 2013), a prominent many-objective EA. In this work, we aim to reproduce and to build upon NAT. It should be noted that NAT uses two sets of supernetwork weights that share the same search space[2]; in contrast, in this work we are primarily interested in supernetworks that represent different search spaces.

2.2.2. Search for architectures of neural ensembles

Neural Ensemble Search (Zaidi et al., 2021) (NES) is the first approach to search architectures of neural ensembles. It trained multiple networks separately, without weight sharing. The follow-up work, Multi-headed Neural Ensemble Search (MH-NES) (Narayanan et al., 2021), utilizes weight sharing by having different ensemble members share first layers of the network. MH-NES achieves gains in robustness, search efficiency and model efficiency. Neural Ensemble Architecture Search (NEAS) (M. Chen et al., 2021a) is a similar approach with the key idea of gradual removal of the least promising operations from the supernetwork.

Neural Ensemble Search via Sampling (NESS) (Shu et al., 2021) is supernet-based but does not require the ensemble models to have the same first layers. For this, a supernetwork is first trained, then subnetworks are sampled via novel sampling algorithms.

Our algorithm, ENCAS, substantially differs from all the ones discussed above. Firstly, ENCAS searches for architectures of cascades, for which no prior work exists (to the best of our knowledge). Secondly, ENCAS is truly multi-objective, with a single run producing

[2] This was not clear to us from (Lu et al., 2021), but it was explained to us in private communication with the authors.

multiple networks on a trade-off front of model size and performance, whereas NES, MH-NES, and NESS do not directly optimize model efficiency; NEAS uses model size as a constraint in single-objective optimization, which thus needs to be run once for each target model size. Thirdly, none of the existing algorithms take advantage of pretrained supernetworks, while we purposefully design our algorithm to rely on them, bearing in mind that pretraining plays a huge role in the success of deep learning approaches (Donahue et al., 2014). Finally, all existing algorithms require the user to specify the ensemble size in advance. Our algorithm has the *maximum* cascade size as a hyperparameter, meaning that it can output cascades with fewer networks if adding more networks does not improve performance.

2.2.3. Cascades of neural networks

As mentioned, cascading is a way of efficiently combining multiple available models. It requires the user to choose a confidence function and confidence thresholds. The confidence function estimates how confident a model is in its prediction for a specific input. An example of that could be the maximum predicted probability or the gap between the largest and the second-largest logit values (Streeter, 2018). The confidence thresholds are used to decide when to stop evaluation and to return the current output. A cascade operates sequentially in the following way: the current model makes a prediction and a confidence value for it is determined by the confidence function; if this value is above the confidence threshold for the current model, the cascade is terminated, otherwise the process is repeated for the next model. Note that the output of a cascade can be either the output of the last used model (Streeter, 2018) (i.e., the outputs of unconfident models are discarded), or the averaged outputs of all the used models (X. Wang et al., 2020). We follow X. Wang et al. (2020) in using the averaged outputs.

Cascades are popular in machine learning (Kukenys & McCane, 2008; Z. Xu et al., 2014), but in deep learning there are only a few works utilizing them (Angelova et al., 2015; Z. Cai et al., 2015). Recently, X. Wang et al. (2020) pointed out that cascades can dominate single models in terms of performance, efficiency, and training time. Cascades of X. Wang et al. (2020) were constructed via an exhaustive search of a small search space of predefined networks.

GreedyCascade (Streeter, 2018) achieved good results by designing an efficient greedy algorithm for cascade selection from a somewhat larger number of networks. GreedyCascade has a fundamental limitation: by construction, it cannot produce a cascade that would perform better than the best model in the model pool. Our algorithm does not have this limitation in its design. In addition, GreedyCascade scales quadratically in the number of networks.

Our algorithm searches in a search space that is substantially larger than ever used before for cascade search (it contains hundreds of networks instead of dozens). Also, ENCAS is multi-objective and requires only a single run to create the trade-off front, unlike the exhaustive search procedure of X. Wang et al. (2020).

2.2.4. Evolutionary algorithms

An EA is a population-based optimization algorithm that relies on the ideas of *(1)* fitness-based selection and *(2)* variation (most often mutation and crossover, i.e., information

transfer between solutions in the population). EAs achieve SOTA results on a variety of benchmark and real-world problems (Chiong et al., 2012; Bouter et al., 2019; Petchrompo et al., 2022).

In this work, we use NSGA-III (Deb & Jain, 2013) (as part of NAT) and the Multi-objective Gene-pool Optimal Mixing Evolutionary Algorithm (MO-GOMEA) (Luong et al., 2014). NSGA-III relies on non-dominated sorting and pre-supplied reference points to keep the population spread-out in the objective space. The two key ideas behind MO-GOMEA are linkage learning (which leads to dependent variables being exchanged between solutions as a single group) and Gene-pool Optimal Mixing (which ensures that crossover always leads to a fitness improvement). Since we use the algorithm without any modifications, we refer the interested reader to (Thierens & Bosman, 2011; Luong et al., 2014) for details. We choose to use this algorithm because it performs well in many problems (Luong et al., 2015; Rodrigues et al., 2016) and because it does not require setting any hyperparameters (such as population size, crossover type, or mutation rate).

2.3. Methods

2.3.1. Searching for cascades of dynamic size

Let us assume that a supernetwork has been trained via the NAT procedure. In addition to the supernetwork weights, the procedure generates a trade-off front of network architectures. The architectures from this front will be used in ENCAS.

Creating a cascade of a specific size out of a predefined model pool comes down to selecting the appropriate sequence of models and their confidence thresholds. Let us focus on the models first. Each out of N models can be encoded as a categorical value $1..N$. To consider cascades with fewer models (or even a single model), we add the value 0 to encode a "no operation" model — a model that does nothing. Then, a solution to the problem of searching for a cascade of maximum size k can be encoded as a list of k values, each ranging from 0 to N. In the interest of robustness we do not search for the weights of the models in the cascade, and take a simple average of their outputs instead.

As to the confidence threshold values, they are encoded as $k-1$ additional categorical variables. In our experiments we use 51 possible values from 0.0 to 1.0 with step size 0.02. If all thresholds are equal to 1, the cascade becomes an ensemble, since the confidence of any model will always be smaller than 1, and thus all the models will be used.

A visual representation of a solution can be seen in Figure 2.2.

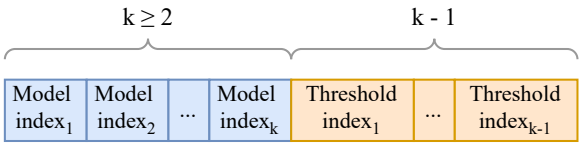


Figure 2.2: Representation of a solution for ENCAS.

To evaluate a solution, every subsequent model is used to only update probabilities of inputs for which previous models were not confident (once the confidence for an input is above the current threshold, cascading stops). The final probabilities are used as

cascade predictions to evaluate its performance. The FLOPs of a cascade are computed as a weighted sum of the FLOPs of all the models in the cascade, where each weight is the fraction of the total number of inputs that this model was used on. Since the models in the model pool are known in advance, their outputs on the validation set can be precomputed, which leads to fast search times of under 1 GPU-hour even on a large dataset (e.g., ImageNet) and with hundreds of base models to choose from.

Note that this approach is trivial to extend to multiple supernetworks by adding the models from the trade-off fronts of all the supernetworks to the model pool. Since the algorithm relies on network outputs, the problem of different supernetworks having different operations is side-stepped.

Any multi-objective search algorithm can be run to maximize validation accuracy and minimize FLOPs. We use MO-GOMEA (Luong et al., 2014). Pseudocode of ENCAS is listed in Algorithm 2.1.

Algorithm 2.1 ENCAS

Input: Supernetworks $\{S_i\}$, possible thresholds $\{t_i\}$, maximum cascade size k

model_pool = { }
for S_i in $\{S_i\}$ **do**
    trade_off_front$_i$ = NAT(S_i)
    model_pool = model_pool ∪ trade_off_front$_i$
fitness_func = make_fitness_func(model_pool, $\{t_i\}$, k)
# make_fitness_func defines the procedure for decoding and encoding the values (see Figure 2.2), and for evaluating a solution, i.e., a cascade (see Section 2.2.3).
cascades = MO-GOMEA(fitness_func)
return cascades # the trade-off front

Empirically, we observed that ENCAS finds hundreds of cascades. To reduce that number and to combat overfitting, the trade-off front found by ENCAS is filtered: we traverse the non-dominated front from least accurate to most accurate cascades, and a cascade is kept if its accuracy on the validation set after rounding to the first significant digit is higher than the accuracy of the previous cascade.

In ENCAS, only the models from the trade-off front of each supernetwork are considered. This means that ENCAS works with models that are very good on their own, but it also means that it cannot create cascades of models that might be subpar individually but extremely good together. Models with weights that complement each other in this way may not even exist in separately-trained supernetworks, so ideally the training of a supernetwork and the cascade search should happen simultaneously. To investigate whether this idea is feasible, we construct a version of ENCAS called ENCAS-joint (indicating that training and search are performed jointly).

ENCAS-joint extends NAT to training several different supernetworks simultaneously. Whereas in NAT a solution represents an architecture of a single model, in ENCAS-joint a solution represents architectures of all the models in a cascade, their target positions, and the threshold values. Confidence thresholds are restricted to 10 possible values from 0.1 to 1.0 with step size 0.1 to decrease the search space size; the confidence threshold of

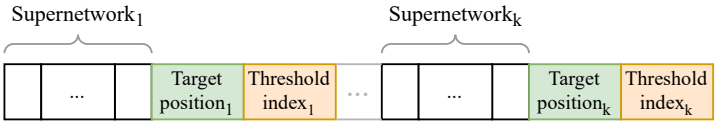


Figure 2.3: Representation of a solution in ENCAS-joint.

the last network is not used. These per-supernetwork representations are concatenated to encode the whole cascade. Figure 2.3 visualizes the encoding.

Note that our encoding of the order of the networks is generally inefficient (i.e., a permutation would be more efficient), but since we use a small number of supernetworks (5), the number of possible orderings is quite small, and our Cartesian encoding should suffice.

Before evaluating a cascade, the networks are ordered (with ties broken arbitrarily), after which the cascade is evaluated as usual. NAT requires defining a surrogate that predicts the fitness of an architecture. To extend this to multiple supernetworks in a simple way, we create such a surrogate for each supernetwork used, and an additional surrogate that combines outputs of supernetwork-wise surrogates for a prediction for the whole cascade. Yet another surrogate is used to estimate the FLOPs count for a cascade from FLOPs of individuals models, target positions, and thresholds (this is necessary because changing thresholds or the order of networks impacts not only its performance but also the FLOPs count). Each of the surrogates is a Radial Basis Function (RBF) (Broomhead & Lowe, 1988) ensemble, the same as in NAT.

Each supernetwork is trained separately based on which subnetworks from it are present in the population. In order not to disadvantage supernetworks that might require more training to achieve good accuracy, we train all the supernetworks for an equal number of steps. To avoid tuning hyperparameters of NSGA-III to the new scenario, we exchange it for MO-GOMEA, for which no hyperparameters are tuned.

Note that this approach has a limitation of not allowing a cascade to contain several models from the same supernetwork. Therefore, it is possible to further improve results by running ENCAS on the supernetworks trained by ENCAS-joint (we refer to this combination as ENCAS-joint+). In the next section we compare all versions of our algorithm.

2.4. Experiments

We conduct experiments on established computer vision benchmark datasets: ImageNet (ILSVRC2012) (Russakovsky et al., 2015), CIFAR-10 (Krizhevsky & Hinton, 2009), and CIFAR-100 (Krizhevsky & Hinton, 2009). In our experiments, we consider the bi-objective problem of maximizing top-1 accuracy while minimizing the FLOPs count. Note that hyperparameter selection and all search procedures were performed on the validation subsets, while the experimental results are reported on the test sets. This means that a trade-off front that is monotonous when evaluated on the validation set often becomes non-monotonous when evaluated on the test set. Since one should not use the test set to select models, we show all the models, even if they are dominated once the test accuracy is considered.

The validation sets for CIFAR-10 and CIFAR-100 consist of 10,000 images randomly split off from the training set (that contain 50,000 images; test sets contain 10,000 images). For ImageNet, we rely on the pretrained networks that use the whole training set and report the results on the ILSVRC2012 validation set, as is established practice, since the true test set is not publicly available. Images seen during training cannot be used during the search because their activations have different statistics (Streeter, 2018), and for comparability our results should be reported on the ILSVRC2012 validation set (which is treated as the test set). As the actual validation set, we therefore used 20,683 images from ImageNetV2 (Recht et al., 2019), which is a dataset designed to match the ImageNet collection procedure as closely as possible[3]. We would prefer to avoid using this additional data, but cannot; as the number of these images is only 1.6% of the ImageNet training set size, we assume that the unfair advantage we gain by using it is negligible.

We use the normalized hypervolume indicator (Zitzler et al., 2003) as a metric of the quality of a trade-off front (see Appendix 2.E for details). Every experiment is run 10 times, mean and standard deviation are reported. We plot the median run (in terms of hypervolume), along with a shaded area delimited by the worst and best fronts achieved over all the runs. Appendix 2.A contains our hyperparameters. Search time is measured on a single Nvidia 2080TI GPU. For statistical testing we use the Wilcoxon signed-rank test (Wilcoxon, 1992) with Bonferroni correction (Dunn, 1961) (target p-value=0.01, 20 tests, corrected p=0.0005, mentions of statistical significance in the text imply smaller p, all p-values are reported in Appendix 2.F).

Our code is public[4]. We have worked with our own implementation of NAT because we did not have access to the authors' code of NAT that was used for the NAT article.

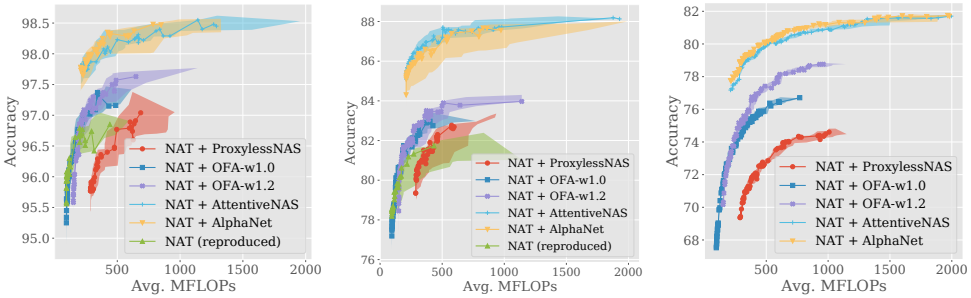
We are interested in utilizing pretrained supernetworks, as training one from scratch takes on the order of thousands of GPU-hours (H. Cai et al., 2020). Unfortunately, many papers do not release either code or pretrained weights. As more supernetworks become available in the future, the value of searching across supernetworks should only increase.

In our experiments we rely on five different supernetworks pretrained on ImageNet: AttentiveNAS (D. Wang et al., 2021b), AlphaNet (D. Wang et al., 2021a), ProxylessNAS (H. Cai et al., 2019), OFA-w1.0 (H. Cai et al., 2020), OFA-w1.2 (H. Cai et al., 2020). All of them are built from inverted residual blocks (Sandler et al., 2018). Moreover, ProxylessNAS, OFA-w1.0, OFA-w1.2 have the same search space, with only width multipliers being different (to get the actual number of neurons in a layer, the base number of neurons is multiplied by the width multiplier). AttentiveNAS and AlphaNet are from the same search space, but the weights were trained via different approaches.

To adapt a supernetwork to CIFAR-10 and CIFAR-100, we apply the NAT procedure for each supernetwork separately. This produces trade-off fronts of models, which in the following sections will be used for cascade construction. For CIFAR-10 and CIFAR-100 we also reproduce NAT with its original hyperparameters using OFA-w1.0 and OFA-w1.2 (due to computational constraints, we do not reproduce NAT for ImageNet). For ImageNet, there is no need to further update the weights, however the trade-off front still needs to

[3] ImageNetV2 contains three sets of 10,000 images that were collected slightly differently, we use images from all three sets: removing duplicates gives 20,683 images.

[4] https://github.com/AwesomeLemon/ENCAS



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Figure 2.4: Results of running NAT (Lu et al., 2021) with different supernetworks on CIFAR-10, CIFAR-100, and ImageNet.

be found. For this reason, we run a version of NAT with no training and no reevaluation of the already evaluated networks.

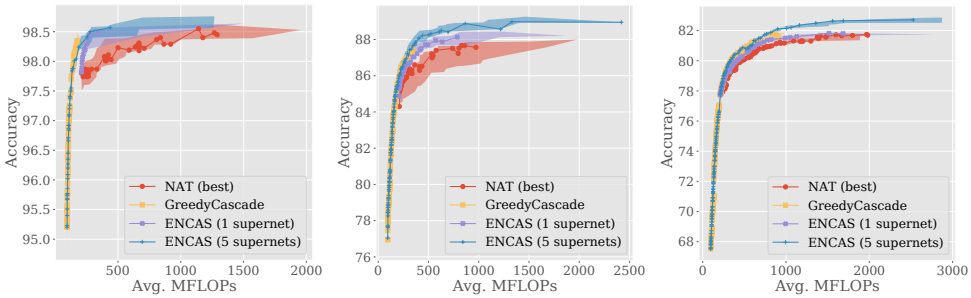
Results of using NAT with each supernetwork are presented in Figure 2.4. It can be seen that the choice of the supernetwork impacts the resulting trade-off front significantly, with supernetworks that perform better on ImageNet also performing better on CIFAR-10 and CIFAR-100, as expected (Kornblith et al., 2019). Additionally, our reproduced NAT achieves results inferior to those reported by Lu et al. (2021), even after we introduced changes that were not in the article but suggested by the authors in private communication (see Appendix 2.A). This prompted us to look for better hyperparameters, which are used in all our experiments (including those in Figure 2.4). With these hyperparameters, search time is 30 GPU-hours for OFA-w1.0, OFA-w1.2, or ProxylessNAS, and 45 GPU-hours for AlphaNet or AttentiveNAS.

2.4.1. Cascading best NAT results

Figure 2.5 and Tables 2.1, 2.2, 2.3 show that using ENCAS on the NAT results with the single best supernetwork leads to the models on the fronts becoming more efficient across all datasets, with hypervolumes increasing (statistically significant; difference in maximum accuracy is not statistically significant). Visually, the effect is small on ImageNet, and noticeable on CIFAR-10 and CIFAR-100.

2.4.2. Cascading all NAT results

The next question is whether using supernetworks other than the best one will improve the results of ENCAS. As shown in Figure 2.5, the results are strongly improved, with the differences in hypervolume and maximum accuracy to the best NAT baseline (and to ENCAS with 1 supernetwork) being statistically significant. We hypothesize that inclusion of better and more diverse supernetworks would make the gap even larger. Search time of ENCAS is 1 GPU-hour (with approximately 300 base models). Since we report all the cascades found by ENCAS (several dozen), we do not name them, but for the ease of reference we name a subset (see Appendix 2.D).



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Figure 2.5: Comparing ENCAS to the baselines.

Table 2.1: ImageNet performance. "HV" denote hypervolume, "acc." denotes top-1 accuracy. The method producing the highest accuracy is in bold.

Method	HV	Max acc.	Max MFLOPs
EfficientNet B0–B3 (Tan & Le, 2019)	0.464	81.6	1800
EfficientNet C0–C3 (X. Wang et al., 2020)	0.482	82.2	1800
MobileNetV3 (Howard et al., 2019)	0.396	76.6	350
OFA (#75) (H. Cai et al., 2020)	0.451	80.0	595
NEAS (M. Chen et al., 2021a)	0.458	80.0	574
NSGANetV2 (Lu et al., 2020)	0.477	80.4	593
AlphaNet (Sharma et al., 2020)	0.490	80.8	709
BigNAS (J. Yu et al., 2020)	0.480	80.9	1040
NAT (best) (Lu et al., 2021)	$0.506_{\pm 0.001}$	$81.69_{\pm 0.02}$	$1962_{\pm 33}$
GreedyCascade (Streeter, 2018)	$0.520_{\pm 0.002}$	$81.72_{\pm 0.11}$	$927_{\pm 36}$
ENCAS (1 supernet)	$0.509_{\pm 0.001}$	$81.78_{\pm 0.05}$	$1929_{\pm 424}$
ENCAS (5 supernets)	$\mathbf{0.537_{\pm 0.001}}$	$\mathbf{82.72_{\pm 0.10}}$	$2616_{\pm 221}$

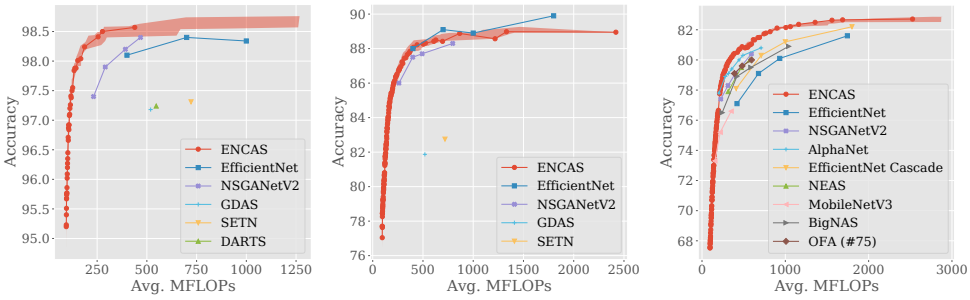
2.4.3. Comparison to SOTA

We compare ENCAS with the SOTA cascade search algorithm GreedyCascade (Streeter, 2018) by applying it to the same model pool. As can be seen in Figure 2.5, ENCAS matches the performance of GreedyCascade for smaller FLOPs on all datasets, and can find cascades with better accuracy than the best baseline model, which GreedyCascade cannot do. Note that the runtime of both algorithms (under 1 hour) is negligible in comparison to the supernetwork adaptation time (tens of hours). Differences in hypervolume and maximum accuracy between ENCAS and GreedyCascade are statistically significant.

Our results are also compared with those of previous efficient NAS algorithms (see Tables 2.1, 2.2, 2.3). Figure 2.6 shows that the trade-off fronts produced by ENCAS dominate other NAS approaches under 1.5 GFLOPs across the datasets. But it can also be seen that for CIFAR-100 while ENCAS is on par with EfficientNet-B0 to B2, it is outperformed by B3, even though the supernetworks we use outperform EfficientNet B0 to B3 on ImageNet.

Table 2.2: CIFAR-100 performance. "HV" denotes hypervolume, "acc." denotes top-1 accuracy. The method producing the highest accuracy is in bold.

Method	HV	Max acc.	Max MFLOPs
EfficientNet B0–B3 (Tan & Le, 2019)	0.664	89.9	1800
NSGANetV2 (Lu et al., 2020)	0.658	88.3	796
GDAS (Dong & Yang, 2019b)	—	81.87	519
SETN (Dong & Yang, 2019a)	—	82.75	722
NAT (reproduced) (Lu et al., 2021)	$0.523_{\pm 0.012}$	$81.53_{\pm 0.54}$	$682_{\pm 298}$
NAT (best) (Lu et al., 2021)	$0.659_{\pm 0.004}$	$87.94_{\pm 0.23}$	$1277_{\pm 287}$
GreedyCascade (Streeter, 2018)	$0.665_{\pm 0.005}$	$87.25_{\pm 0.25}$	$341_{\pm 30}$
ENCAS (1 supernet)	$0.663_{\pm 0.005}$	$88.09_{\pm 0.21}$	$1051_{\pm 353}$
ENCAS (5 supernets)	$0.699_{\pm 0.003}$	$88.96_{\pm 0.17}$	$1401_{\pm 420}$
ENCAS-joint	$0.681_{\pm 0.005}$	$88.55_{\pm 0.19}$	$6678_{\pm 1735}$
ENCAS-joint+	$\mathbf{0.701_{\pm 0.005}}$	$\mathbf{89.10_{\pm 0.22}}$	$1750_{\pm 418}$



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Figure 2.6: Comparing ENCAS with SOTA NAS algorithms.

This may occur because training an individual network is much easier than training a supernetwork (in our experience, training a supernetwork is hard due to subnetworks having to share both weights and hyperparameters).

Note that we do not compare search times of different algorithms because the corresponding publications often report times that are not comparable due to e.g., using different hardware, not accounting for supernetwork training or final network retraining time. A fair comparison would require us to run all the algorithms, for which we lack compute. For future reference, the runtime associated with each part of our pipeline is mentioned in the section describing it.

2.4.4. Joint training and cascade search

Is joint weight training and search of cascade architectures beneficial? In Figure 2.7 we can see that ENCAS-joint finds a trade-off front that is worse than the one found by ENCAS.

Table 2.3: CIFAR-10 performance. "HV" denotes hypervolume, "acc." denotes top-1 accuracy. The method producing the highest accuracy is in bold.

Method	HV	Max acc.	Max MFLOPs
EfficientNet B0–B2 (Tan & Le, 2019)	0.863	98.4	1000
NSGANetV2 (Lu et al., 2021)	0.904	98.4	468
GDAS (Dong & Yang, 2019b)	—	97.18	519
SETN (Dong & Yang, 2019a)	—	97.31	722
DARTS (H. Liu et al., 2019)	—	97.24	547
NAT (reproduced) (Lu et al., 2021)	$0.899_{\pm 0.003}$	$96.80_{\pm 0.13}$	$361_{\pm 119}$
NAT (best) (Lu et al., 2021)	$0.911_{\pm 0.002}$	$98.46_{\pm 0.08}$	$1390_{\pm 274}$
GreedyCascade (Streeter, 2018)	$0.935_{\pm 0.002}$	$98.31_{\pm 0.05}$	$191_{\pm 16}$
ENCAS (1 supernet)	$0.911_{\pm 0.002}$	$98.45_{\pm 0.09}$	$698_{\pm 321}$
ENCAS (5 supernets)	$0.941_{\pm 0.002}$	$98.60_{\pm 0.09}$	$749_{\pm 298}$
ENCAS-joint	$0.935_{\pm 0.002}$	$98.68_{\pm 0.04}$	$4858_{\pm 1126}$
ENCAS-joint+	$\mathbf{0.943_{\pm 0.002}}$	$\mathbf{98.68_{\pm 0.08}}$	$1060_{\pm 444}$

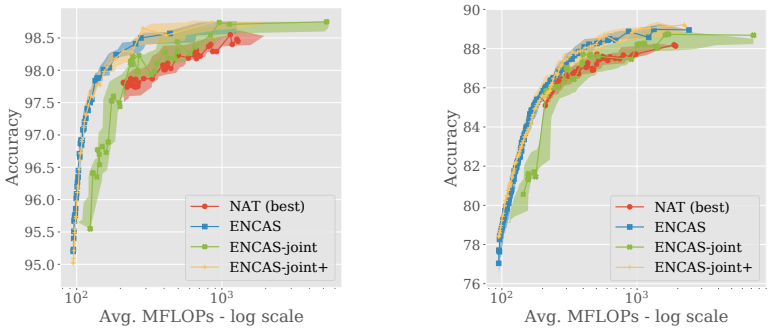


Figure 2.7: Investigating benefits of joint training and search.

This likely happens due to the increased size of the search space. However, the trade-off front found by ENCAS-joint is better than the best NAT one.

We further see that ENCAS-joint+ (running ENCAS on the supernetworks trained by ENCAS-joint) improves upon ENCAS-joint on both CIFAR-10 and CIFAR-100. But is it better than running ENCAS on separately trained supernetworks? Although ENCAS-joint+ appears to outperform ENCAS in terms of hypervolume and maximum accuracy (see Tables 2.2, 2.3), these results are not statistically significant.

Note that these experiments are not performed on ImageNet due to limitations in computational power. Training supernetworks jointly is computationally intensive in general (search time of ENCAS-joint is 240 GPU-hours) while also lacking flexibility, as adding or removing a network means restarting the whole process from scratch. Given inconclusiveness of improvements brought by ENCAS-joint+, we recommend using ENCAS and separate training of supernetworks (see Section 2.6 for further discussion).

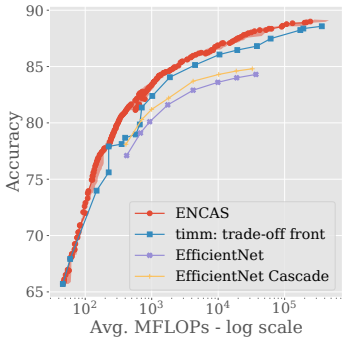


Figure 2.8: ENCAS discovers a dominating trade-off front on ImageNet by searching for cascades of 518 `timm` models (from which only the models on the trade-off front are shown).

2.4.5. Applying ENCAS to SOTA ImageNet models

ENCAS relies on the architectures discovered via supernetwork-based NAS. However, using a supernetwork means that these architectures are necessarily not very large. Because of this, NAS results are typically evaluated in the context of a mobile phone setting, which is usually taken to mean ≤ 600 MFLOPs.

However, nowadays there are hundreds of large well-performing ImageNet-pretrained models available online. Can our good results be extended from the mobile phone setting to dominating the complete trade-off front? To answer this question, we take 518 ImageNet models from the PyTorch Image Models (`timm`)library (Wightman, 2019), and run our search procedure on them.

As shown in Figure 2.8, this indeed leads to a dominating trade-off front. The increase of 0.4 percentage points in the maximum ImageNet performance leads to our largest cascade achieving the highest ImageNet accuracy of publicly available models ($89.01_{\pm 0.10}$), while simultaneously decreasing FLOPs by 18% (from 362 GFLOPs to $296_{\pm 77}$ GFLOPs). Our cascades outperform those in (X. Wang et al., 2020), in large part thanks to the ability of our algorithm to use a search space containing hundreds of models, which is not feasible for the exhaustive search approach used in (X. Wang et al., 2020).

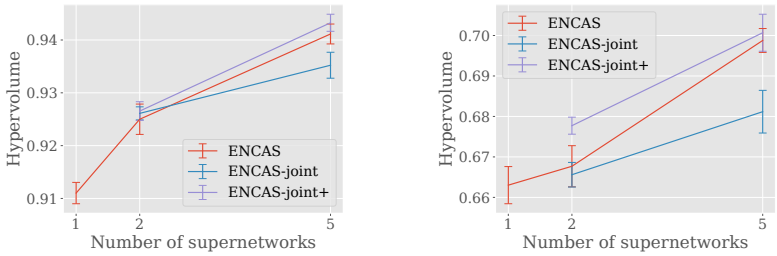
2.5. Additional experiments

In this section we further investigate the impact of using more than one supernetwork. Due to space constraints, a comparison to ensembles is provided in Appendix 2.B and a comparison to random search is provided in Appendix 2.C.

2.5.1. Impact of increasing the number of supernetworks

Figure 2.9 shows the impact of increasing the number of supernetworks used in ENCAS from 1 to 2 to 5. A trend of increasing hypervolume can be clearly observed.

For joint training (ENCAS-joint), the hypervolume also increases, but not as much. This can be explained by the increase in the search space that every additional supernet-



(a) CIFAR-10 (b) CIFAR-100

Figure 2.9: Impact of increasing the number of supernetworks in ENCAS, ENCAS-joint, and ENCAS-joint+.

work brings. Interestingly, the hypervolume obtained with ENCAS-joint+ grows about as fast as with ENCAS, which we interpret to mean that the joint training and search over an increasing number of supernetworks is beneficial for weights while harmful for the simultaneous search (given the same search budget). The larger search space may require a larger search budget to achieve better results, and therefore ENCAS-joint may have higher potential to benefit from more compute. Running ENCAS using the weights trained by ENCAS-joint (i.e., ENCAS-joint+) realizes the benefit of better weights and ameliorates the downside of a larger search space.

2.5.2. Is using different supernetworks better than using the best one trained several times?

Experiments in section 2.4.2 demonstrated that using several supernetworks is better than using just the best one. But is the source of the effect the diversities of architectures found, or just the increased quantity of networks with different weights? To answer this question, we train (via NAT) the best supernetwork 5 times with different seeds on CIFAR-10 and CIFAR-100 and apply ENCAS to the resulting trade-off fronts.

In Figure 2.10 (left) we see that using different runs of the best supernetwork barely increases hypervolume, in contrast to using different supernetworks. If we inspect the trade-off fronts in Figure 2.10 (right) for 5 supernetworks and for 5 seeds of the best supernetwork on CIFAR-10, we can see that the difference is in low-FLOPs models that are missing from the best supernetwork but are present in other supernetworks. Therefore, using diverse supernetworks is helpful for obtaining a larger trade-off front coverage. However, on the side of the most accurate networks, using multiple restarts of NAT with the best supernetwork is sufficient to get close to the best performance, unlike what could be expected, since the diversity in architectures and weights is arguably lower.

2.6. Discussion

While our approach achieves good results with limited resource usage, it still suffers from limitations. Notably, it relies on pretrained supernetworks, which are currently not very diverse, architecture-wise. Additionally, these supernetworks need to allow extraction

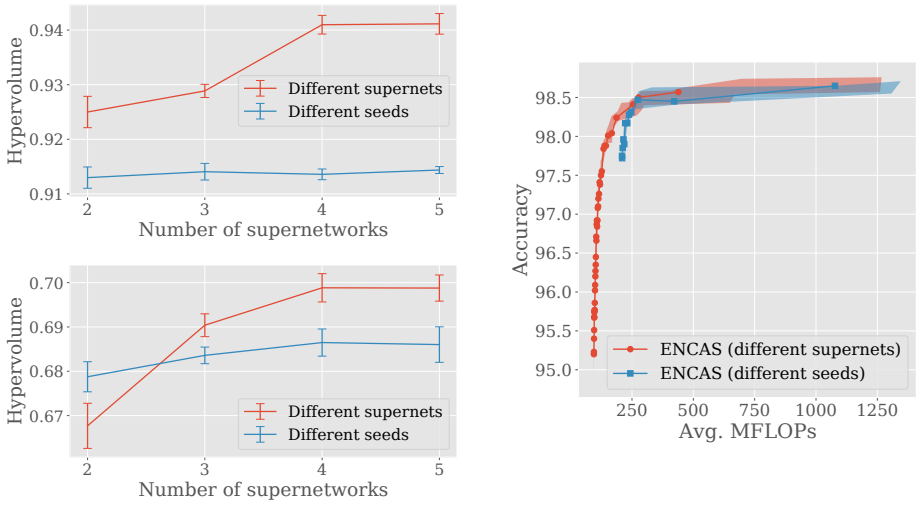


Figure 2.10: Comparison between hypervolume when using different supernetworks, or the best supernetwork trained with different seeds on *(top left)* CIFAR-10, *(bottom left)* CIFAR-100. The plot on the *(right)* shows comparison between trade-off fronts of 5 supernetworks, or 5 seeds of the best supernetwork on CIFAR-10.

and usage of subnetworks without retraining in order for our approach to work, which limits their selection even further.

In our experiments, we find that performing search after the supernetwork weights have been adapted is not much worse than joint training and search. This can mean that there is not a lot of benefit to be gained by fine-tuning architecture choices of different cascade components to each other; alternatively, perhaps ENCAS-joint was simply not able to realize these benefits, for instance because it may require more computational resources than we used in our experiments.

This chapter has demonstrated the benefits brought by the usage of cascades. This reinforces the main thesis of (X. Wang et al., 2020): researchers should pay more attention to cascades. However, the warnings of (X. Wang et al., 2020) should also be repeated, as they apply to any cascade approach, including ours: the decrease in FLOPs brought by cascades can be realized either when processing images one-by-one, or when processing a large amount of images offline. The benefits are not realized in online batch processing: once a batch has been created, due to the parallel nature of GPU accelerators, processing a part of a batch takes approximately the same resources as processing the whole batch.

2.7. Conclusion

In this chapter, we considered the automatic creation of cascades of deep neural networks. We developed an effective algorithm called ENCAS that builds upon the literature on efficient NAS by searching for cascades across pretrained supernetworks either simultaneously with weight training or after weight training. ENCAS is the first NAS algorithm that

searches for cascade architectures. It does so by solving the multi-objective optimization problem of finding well-performing small cascades with the help of an EA (MO-GOMEA).

ENCAS was found to outperform SOTA efficient NAS approaches on several image classification datasets. Its search procedure can also be applied to an arbitrary model pool. By applying it to well-performing publicly available ImageNet models, we achieved a dominating trade-off front on ImageNet.

Finally, we find that searching for neural network architectures in more than one pre-trained supernetwork is beneficial despite the limited diversity of the currently available supernetworks, which is expected to only increase with time.

Acknowledgments

We thank Arkadiy Dushatskiy for insightful discussions, Marco Virgolin for introducing us to (X. Wang et al., 2020), Zhichao Lu for sharing his knowledge about NAT and parts of the NAT codebase.

Appendix

2.A. Hyperparameters

In this section, hyperparameters for our experiments are listed.

Reproducing NAT: initial learning rate 0.01, optimizer Stochastic Gradient Descent with Nesterov momentum 0.9 (Nesterov, 1983), training for 30 meta-iteration of 5 epochs each, cosine learning rate schedule (Loshchilov & Hutter, 2017), train batch size 96, validation batch size 150, CutOut (Devries & Taylor, 2017) with size 56, RandAugment (Cubuk et al., 2020) with the policy "rand-m9-mstd0.5", 10,000 surrogate evaluations for each meta-iteration, the surrogate is an ensemble of 500 RBFs (Broomhead & Lowe, 1988), archive size 300, sampling a network to train by constructing a distribution of possible architecture choices from archive and sampling each choice from this distribution.

Better NAT results (only differences are listed): cosine cyclic learning rate schedule with period of 5 epochs, CutOut is replaced with CutMix (Yun et al., 2019), Stochastic Weight Averaging (Izmailov et al., 2018) of models from the last 20 meta-iterations (i.e., epochs 150, 145,.. 55), sampling a network to train by directly sampling an architecture from the archive.

ENCAS-joint: 5 epochs per supernetwork per iteration, the rest follows "Better NAT results".

Techniques suggested to us privately by the NAT authors to improve NAT performance are: usage of CutOut, RandAugment, and Riesz s-energy (Blank et al., 2020) for reference point generation instead of the Das-Dennis method (Das & Dennis, 1998).

ENCAS was run for 600,000 evaluations. For experiments with `timm` an Nvidia A100 card was used for its superior VRAM size.

2.B. Evolutionary neural ensemble search

In this section we briefly discuss a restriction of ENCAS to ensemble search, an algorithm we name Evolutionary Neural ENsemble Search (ENENS). Whereas ENCAS searches for a sequence of models and confidence thresholds, ENENS searches only for the models, as confidence thresholds are not needed in an ensemble. The overall procedure is a simplified version of the one in ENCAS (Section 2.3.1): a solution representation consists of model indices (thresholds are no longer a part of the representation); a solution is evaluated by averaging the predictions of all the models in it; the FLOPs count is the sum of FLOPs counts of all the models in the ensemble. The rest of the algorithm is unchanged.

2.B.1. Comparison with existing neural ensemble search algorithms

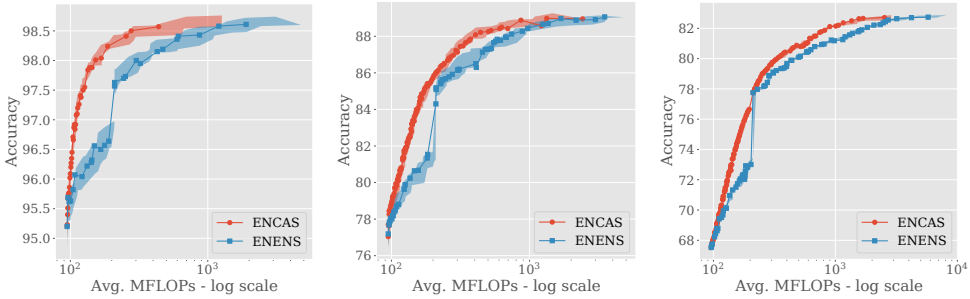
The existing algorithms for ensemble architecture search do not use pretrained supernetworks. Being able to use pretrained supernetworks is an important advantage in deep learning, an advantage that is realized by ENENS, which can be seen to exhibit superior performance in terms of top-1 accuracy, as reported in Table 2.4.

2.B.2. Cascade vs ensemble

The comparison of ENCAS and ENENS in Figure 2.11 shows that the front of cascades dominates for most FLOPs values. The largest ensembles perform very similarly, or, in

Table 2.4: Top-1 accuracy of different algorithms for search of network architectures for an ensemble. For our experiment, mean and standard deviation across 10 seeds are reported.

Method	CIFAR-10	CIFAR-100	ImageNet
NES (Zaidi et al., 2021)	92.4	76.5	—
MH-NES (Narayanan et al., 2021)	—	80.35	—
NESS (Shu et al., 2021)	97.64	85.45	77.7
NEAS (M. Chen et al., 2021a)	—	—	80.0
ENENS	$98.62_{\pm 0.06}$	$88.99_{\pm 0.09}$	$82.80_{\pm 0.07}$



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

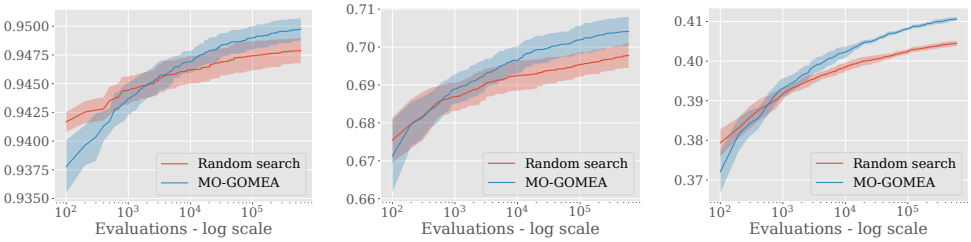
Figure 2.11: The trade-off front of cascades is better than the trade-off front of ensembles on the test set.

case of ImageNet, perhaps perform slightly better, even though technically they could be found by ENCAS as special cases of cascades. Best we can tell, this does not happen because in the largest-FLOPs regime ENCAS finds smaller cascades and ensembles (than ENENS) that perform better on the validation set, but lose a bit of performance on the test set. The larger ensembles found by ENENS do not face this issue. The small increase in accuracy comes at the cost of thousands more MFLOPs, however, which means that ensembles should be considered only when efficiency is of no concern.

2.C. Comparison to random search

Random search is a strong baseline for NAS (Li & Talwalkar, 2019). Consequently, we replace MO-GOMEA in ENCAS with random search to investigate whether a simpler search algorithm will suffice. In Figure 2.12 we can see that while random search achieves better results in the first several hundred evaluations of search, it is overtaken by MO-GOMEA after that, demonstrating the benefit of the more sophisticated search approach in terms of the optimization problem it is tasked to solve.

However, Table 2.5 shows that the difference in test performance over all the datasets is small (not statistically significant). Better performance of MO-GOMEA on the validation



(a) CIFAR-10 (b) CIFAR-100 (c) ImageNet

Figure 2.12: MO-GOMEA outperforms random search in terms of hypervolume of the validation trade-off fronts.

Table 2.5: Comparison of using MO-GOMEA and random search in ENCAS (test set performance, mean and standard deviation are computed across 10 seeds).

Method	Hypervolume	Max accuracy	Max MFLOPs
	CIFAR-10		
Random	$0.941_{\pm 0.002}$	$98.61_{\pm 0.08}$	$814_{\pm 307}$
MO-GOMEA	$0.941_{\pm 0.002}$	$98.60_{\pm 0.09}$	$749_{\pm 298}$
	CIFAR-100		
Random	$0.698_{\pm 0.003}$	$88.91_{\pm 0.15}$	$1298_{\pm 389}$
MO-GOMEA	$0.699_{\pm 0.003}$	$88.96_{\pm 0.17}$	$1401_{\pm 420}$
	ImageNet		
Random	$0.535_{\pm 0.001}$	$82.66_{\pm 0.07}$	$2708_{\pm 588}$
MO-GOMEA	$0.537_{\pm 0.001}$	$82.72_{\pm 0.10}$	$2614_{\pm 221}$

set but not the test set means that overfitting to the validation set occurs. Combatting overfitting is an avenue for future work.

2.D. ENCAS models for reference

In the main text, we avoid naming specific models on the trade-off front, as we believe that it is helpful to have NAS produce tens or hundreds of models that may differ only slightly but that allow users more flexibility in choosing which one to use. Nonetheless, to make referencing our models easier, here we specify FLOPs and top-1 accuracy of selected models. A model is included if its MFLOPs value is closest to a value divisible by 100 — this way we select a representative and spread-out sample of the trade-off front while avoiding cherry-picking. The models are taken from the median run (by hypervolume as evaluated on the test set).

Table 2.6: CIFAR-10, selected models.

Name	MFLOPs	Accuracy
ENCAS@100	100	96.19
ENCAS@186	186	98.23
ENCAS@276	276	98.49
ENCAS@439	439	98.57

Table 2.7: CIFAR-100, selected models.

Name	MFLOPs	Accuracy
ENCAS@100	100	78.98
ENCAS@202	202	85.85
ENCAS@299	299	87.14
ENCAS@405	405	88.08
ENCAS@503	503	88.26
ENCAS@607	607	88.42
ENCAS@694	694	88.42
ENCAS@865	865	88.88
ENCAS@1217	1217	88.57
ENCAS@1330	1330	88.98
ENCAS@2419	2419	88.95

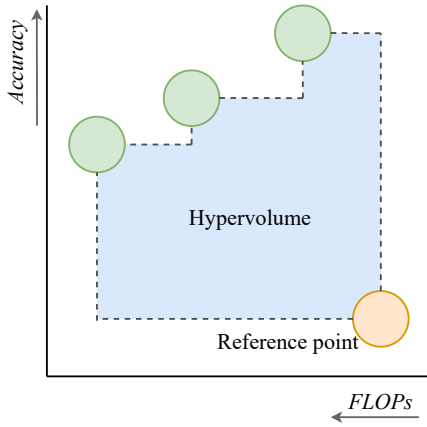


Figure 2.13: Hypervolume indicator for the task of maximizing accuracy and minimizing FLOPs. Green circles correspond to the solutions on the trade-off front.

Table 2.8: ImageNet, selected models.

Name	MFLOPs	Accuracy
ENCAS@100	100	68.19
ENCAS@211	211	77.75
ENCAS@301	301	79.74
ENCAS@387	387	80.43
ENCAS@517	517	80.79
ENCAS@615	615	81.34
ENCAS@690	690	81.49
ENCAS@817	817	81.87
ENCAS@900	900	82.11
ENCAS@1006	1006	82.14
ENCAS@1060	1060	82.20
ENCAS@1160	1160	82.36
ENCAS@1357	1357	82.49
ENCAS@1559	1559	82.63
ENCAS@1687	1687	82.66
ENCAS@2526	2526	82.71

2.E. Hypervolume

The hypervolume indicator measures the volume encapsulated between the trade-off front and a reference point in the objective space (Zitzler et al., 2003). Hypervolume is a useful metric of the quality of a trade-off front.

Since hypervolume is measured relative to a reference point, its absolute value has no meaning, as changing the reference point changes it. However, if the reference point is fixed, relative differences between the hypervolumes of different algorithms can indicate which of them finds a better trade-off front. Note however that hypervolume, while useful, does not perfectly catch every aspect that is important when comparing fronts. Especially when values are close, other measures or visual front inspection are advised.

A visual representation of hypervolume can be seen in Figure 2.13. Normalized hypervolume means hypervolume divided by its largest possible value. In our experiments the reference point is fixed at (MFLOPs=4000, accuracy=60).

2.F. Statistical testing

Table 2.9 lists p-values for one-sided Wilcoxon pairwise rank tests with Bonferroni correction (null hypothesis is that Algorithm 1 is worse than Algorithm 2).

All benchmark datasets (CIFAR-10, CIFAR-100, ImageNet) are tested together to increase sample size. P-values below a significance threshold of 0.0005 are highlighted (target p-value=0.01, 20 tests, corrected p=0.0005). P-values are rounded for visualization purposes. Unless specified otherwise, 5 supernetworks are used. Experiments are separated into two groups: those in the main text and those in the appendix.

Table 2.9: P-values of conducted experiments (values below the significance threshold of 0.0005 are highlighted).

Algorithm 1	Algorithm 2	Split	Hypervolume	Max accuracy
ENCAS (1 supernetwork)	NAT (best)	test	5.6e-6	0.0008
ENCAS	NAT (best)	test	8.7e-7	8.7e-7
ENCAS	ENCAS (1 supernetwork)	test	8.7e-7	8.7e-7
ENCAS	GreedyCascade	test	8.7e-7	8.7e-7
ENCAS-joint+	ENCAS-joint	test	1.9e-6	0.0008
ENCAS-joint+	ENCAS	test	0.0133	0.0093
ENCAS	ENCAS (5 seeds of the best supernet)	test	9.5e-7	0.9385
ENCAS	ENENS	test	8.7e-7	0.9829
ENCAS with MO-GOMEA	ENCAS with Random search	val	8.7e-7	1.8e-6
ENCAS with MO-GOMEA	ENCAS with Random search	test	0.0247	0.0512

3

Shrink-Perturb Improves Architecture Mixing during Population Based Training for Neural Architecture Search

In this work, we show that simultaneously training and mixing neural networks is a promising way to conduct Neural Architecture Search (NAS). For hyperparameter optimization, reusing the partially trained weights allows for efficient search, as was previously demonstrated by the Population Based Training (PBT) algorithm. We propose PBT-NAS, an adaptation of PBT to NAS where architectures are improved during training by replacing poorly-performing networks in a population with the result of mixing well-performing ones and inheriting the weights using the shrink-perturb technique. After PBT-NAS terminates, the created networks can be directly used without retraining. PBT-NAS is highly parallelizable and effective: on challenging tasks (image generation and reinforcement learning) PBT-NAS achieves superior performance compared to baselines (random search and mutation-based PBT).

The contents of this chapter are based on the following publication: Alexander Chebykin, Arkadiy Dushatskiy, Tanja Alderliesten & Peter A. N. Bosman. 2023. Shrink-Perturb Improves Architecture Mixing During Population Based Training for Neural Architecture Search. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland*, 372: 381–388. Frontiers in Artificial Intelligence and Applications. IOS Press. https://doi.org/10.3233/FAIA230294.

3.1. Introduction

Neural Architecture Search (NAS) is the process of automatically finding a neural network architecture that performs well on a target task (such as image classification (H. Liu et al., 2019), natural language processing (Klyuchnikov et al., 2022), image generation (C. Gao et al., 2020)). One of the key questions for NAS is the question of efficiency, since evaluating every promising architecture by fully training it would require an extremely large amount of computational resources.

Many approaches have been proposed for increasing the search efficiency: low-fidelity evaluation (Zoph & Le, 2017; Shim et al., 2021), using weight sharing via a supernetwork (Pham et al., 2018; H. Cai et al., 2020), estimating architecture quality via training-free metrics (Abdelfattah et al., 2021; Mellor et al., 2021). Typically, each approach has two stages: first, finding an architecture efficiently, then, training it (or its scaled-up version) from scratch. This final training usually requires a manual intervention (e.g., if an architecture of a cell is searched, determining how many of these cells should be used), which diminishes the benefit of an automatic approach (potentially, this could also be automated, but we are not aware of such studies in the literature). Ideally, an architecture itself (not its proxy version) should be searched on the target problem, with the search result being immediately usable after the search (such single-stage NAS approaches exist but are limited: e.g., they restrict potential search spaces (S. Hu et al., 2020) or require costly pretraining (H. Cai et al., 2020; D. Wang et al., 2021b)).

For the task of hyperparameter optimization (which is closely related to NAS), effective and efficient single-stage algorithms exist in the form of Population Based Training (PBT) (Jaderberg et al., 2017) and its extensions (Dalibard & Jaderberg, 2021; Liang et al., 2021). The key idea of PBT is to train many networks with different hyperparameters (a population) in parallel: as the training progresses, worse networks are replaced by copies of better ones (including the weights), with hyperparameter values explored via random perturbation. PBT is highly efficient due to the weight reuse, and due to its parallel nature: given a sufficient amount of computational resources, running PBT takes approximately the same wall-clock time as training just one network.

Determining the best way to adapt PBT to NAS is an open research question (Dalibard & Jaderberg, 2021): if a network architecture has been perturbed, the partly-trained weights cannot be reused (because, e.g., weights of a convolutional layer cannot be used in a linear one). The naive approach of initializing them randomly does not work well (see Section 3.5.3), and existing algorithms extending PBT to NAS (Franke et al., 2021; Wan et al., 2022) sidestep the issue at the cost of parallelizability or performance (see Section 3.2.2).

We propose to adapt PBT to NAS by modifying the search to rely not on random perturbations but on mixing layers of the networks in the population. An example of this principle is combining an encoder and a decoder from two different autoencoder networks, ultimately obtaining a better-performing network. In this setting, the source of the weights for the changed layers is natural: they can be copied from the parent networks. Furthermore, we explore if additionally adapting the copied weights with the shrink-perturb technique (Ash & Adams, 2020) (reducing weight magnitude and adding noise) is helpful for achieving a successful transfer of a layer from one network to another.

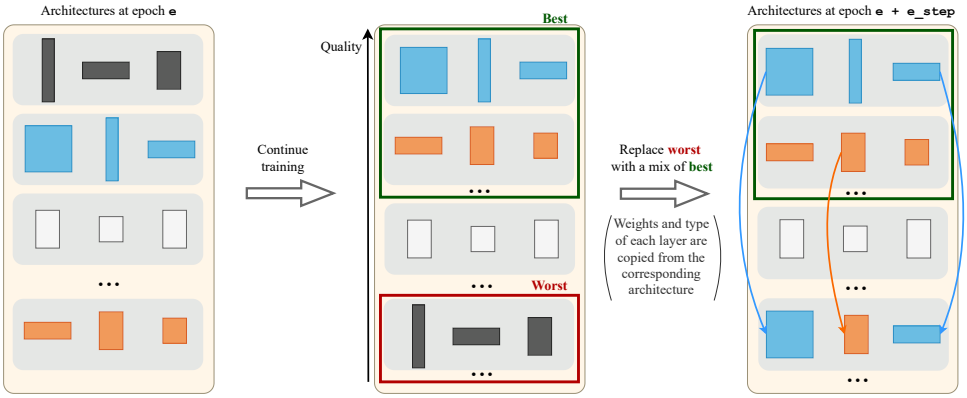


Figure 3.1: In each iteration of PBT-NAS, architectures in the population continue training for several epochs and then are sorted by performance. Every architecture from the bottom percentile is replaced with a mix of two architectures from the top percentile. During mixing, each layer is copied from one of these two architectures (weights from the architecture with worse performance are shrink-perturbed). Different shapes represent different types of layers.

For many standard tasks (such as image classification), single-objective NAS algorithms are matched by (or show only a small improvement over) the simple baseline of random search (Li & Talwalkar, 2019; K. Yu et al., 2020). In order to make the potential benefit of PBT-NAS clear, experiments in this chapter are conducted in two challenging settings: Generative Adversarial Network (GAN) training, and Reinforcement Learning (RL) for visual continuous control. We further advocate for harder tasks and search spaces in Section 3.6.

While our approach could potentially be extended to include hyperparameter optimization, this chapter is focused on architecture search.

The contributions of this work are threefold:

1. We propose to conduct NAS by training a population of different architectures and mixing them on-the-fly to create better ones (while inheriting the weights).

2. We investigate if applying shrink-perturb (Ash & Adams, 2020) to the weights is a superior technique for weight inheritance compared to copying or random reinitialization.

3. Integrating these ideas, we introduce PBT-NAS, an efficient and general NAS algorithm, and evaluate it on challenging NAS search spaces and tasks.

3.2. Related work

3.2.1. Neural architecture search

NAS is the automatic process of finding a well-performing neural network architecture for a specific task. Already in early NAS work (Zoph & Le, 2017), efficiency concerns played

a role: candidate architectures were trained for only a few epochs. Similar low-fidelity search methods save compute by using fewer layers (Lu et al., 2019), or only a subset of the data (Shim et al., 2021). Another way to save compute is by utilizing a training-free metric to perform NAS without any training (Abdelfattah et al., 2021; Mellor et al., 2021).

ENAS (Pham et al., 2018) introduced the idea of weight sharing: all candidate architectures are viewed as subsets of a supernetwork, with the weights of the common parts reused across the architectures. The final architecture is scaled up and trained from scratch. This approach greatly decreased cost of the search to just several GPU-days. DARTS (H. Liu et al., 2019) further increased efficiency by continuously relaxing the problem. Many approaches build upon DARTS by e.g., reducing memory usage (Y. Xu et al., 2020) or improving performance (X. Chen et al., 2021). AdversarialNAS (C. Gao et al., 2020) extends the approach to GAN training, outperforming previous algorithms (X. Gong et al., 2019).

In OnceForAll (H. Cai et al., 2020), a supernetwork is pretrained such that subnetworks would perform well without retraining, in AttentiveNAS (D. Wang et al., 2021b) and AlphaNet (D. Wang et al., 2021a) performance is further improved. These approaches are a good fit for multi-objective NAS (where in contrast to single-objective NAS, multiple architectures with different trade-offs between objectives such as performance and latency are searched). However, the costs for the proposed pretraining reach thousands of GPU-hours. Additionally, in any supernetwork approach, the diversity and size of the architectures are restricted by the supernetwork.

Our approach of exchanging layers and weights between different networks is distinct from the supernetwork-based weight sharing. The weights in the supernetwork are constrained to perform well in a variety of subnetworks, while in our approach, after the weights have been copied to the network with a novel architecture, they can be freely adapted to it, independently of what happens to their original version in the parent network.

The general idea of creating new architectures by modifying existing ones and reusing the weights has been explored in NAS approaches (Elsken et al., 2019; H. Jin et al., 2019) relying on network morphisms(T. Chen et al., 2016; Wei et al., 2016). Network morphisms are operators that change the architecture of a neural network without influencing its functionality. Although (Elsken et al., 2019; H. Jin et al., 2019) successfully used morphisms, the idea was later challenged (Wen et al., 2020) with experiments demonstrating that random initialization of new layers is superior to morphisms. Morphisms are different from our work: while they create a new architecture by modifying one existing architecture, we seek to mix two distinct architectures and reuse their weights.

3.2.2. Population based training

In hyperparameter optimization, hyperparameters of neural network training, such as learning rate or weight decay, are optimized. Bayesian optimization algorithms (Hutter et al., 2011; Falkner et al., 2018) are commonly used for sequentially evaluating promising hyperparameter configurations. Other approaches include Evolutionary Algorithms (Loshchilov & Hutter, 2016; Liang et al., 2021), and random search (Bergstra & Bengio, 2012), a simple but reasonably good baseline.

In contrast to the approaches that train weights for each hyperparameter configuration from scratch, PBT (Jaderberg et al., 2017) reuses partly-trained weights when exploring hyperparameters (see Section 3.1 for short description and (Jaderberg et al., 2017) for details).

To the best of our knowledge, two algorithms were proposed for including architecture search into PBT: SEARL (Franke et al., 2021) and BG-PBT (Wan et al., 2022). In SEARL, the architecture is modified by mutation, which can add a linear layer, add neurons to an existing layer, change an activation function, or add noise to the weights. We use a SEARL-like mutation as a baseline. In BG-PBT, there are multiple generations; in each generation, network architectures are sampled, initialized with random weights, and their training is sped up via distillation from the best network of the previous generation. This approach adds complexity in the form of multiple generations and using distillation (that would require adaptation to each setting, e.g., GAN training). Both SEARL and BG-PBT were proposed exclusively for RL tasks, while we construct PBT-NAS to be a general NAS algorithm.

3.2.3. Combining several neural networks into one

Neural networks can be combined in various ways. In evolutionary NAS (Lu et al., 2019) where weights are trained from scratch for each considered architecture, crossover is performed between encodings of architectures. Alternatively, there exist methods combining only the weights of networks that have the same architecture (Uriot & Izzo, 2020; Ainsworth et al., 2023). Naively averaging the weights leads to a large loss in performance (Ainsworth et al., 2023), which motivated these approaches to align neurons so that they would represent similar features. Averaging weights without alignment is possible if the weights of the networks are closely related. The idea of model soups (Wortsman et al., 2022) is to start with a pretrained model, fine-tune it with different sets of hyperparameters, and greedily search which of the fine-tuned models to average.

In our approach, we mix different architectures *together with the weights* during training, in contrast to evolutionary NAS algorithms combining only architecture encodings, and training the weights from scratch. We also avoid the additional complexity of aligning neurons, instead we continue to train the created network, and allow the gradient descent procedure to adapt the neurons to each other (which is facilitated by shrink-perturb (Ash & Adams, 2020), see Section 3.3.3).

3.3. Method

3.3.1. Problem setting

The goal of single-objective NAS is to find a network architecture α^* from a search space Ω that maximizes an objective function f after training the weights θ:

$$\alpha^* = \arg\max_{\alpha \in \Omega} f(\alpha; \theta) \tag{3.1}$$

Ω typically includes network architecture properties such as the number of layers, types of each layer, and its hyperparameters (e.g., convolution size). We will describe an architecture α by M categorical variables $\{x_i\}_{i=0..M-1}$, each taking l_i possible values. Note that in the case where more than one architecture is searched for (e.g., generator and

Algorithm 3.1 PBT-NAS

Input: search space Ω, number of variables M, population size N, number of epochs e_total, step size e_step, selection parameter τ, probability p of replacing a layer, parameters λ, γ of shrink-perturb

$pop \leftarrow \{N$ random architectures from $\Omega\}$
$e \leftarrow 0$
while $e < e\_total$ **do**
  for $i \leftarrow 0$ **to** $N-1$ **do** # in parallel
    train pop_i for e_step epochs
    $pop_i.fitness \leftarrow$ `evaluate`(pop_i)
  sort pop by $fitness$
  $best\_nets \leftarrow$ the best τ% nets
  $worst\_indices \leftarrow$ indices of the worst τ% nets
  for j **in** $worst\_indices$ **do**
    $pop_j \leftarrow$ `create_architecture`$(best\_nets, p, M, \lambda, \gamma)$
    # the result of mixing, see Algorithm 3.2
  $e \leftarrow e + e\_step$

discriminator of a GAN), we consider, for simplicity, α to include architecture parameters of all architectures.

3.3.2. Algorithm overview

In our algorithm, PBT-NAS, we follow the general structure of PBT, where N networks are trained in parallel[1]. In each iteration of the algorithm, every network is trained for e_step epochs. Then, each of the worst τ% of the networks is replaced by a mix of two networks from the best τ% (according to the objective function f). Over time, better architectures are created. Mixing architectures during training is the key component of PBT-NAS. In Section 3.3.3, we motivate the choice to do NAS by mixing networks. Further details of how we mix architectures are given in Section 3.3.4.

A visual representation of one iteration of PBT-NAS is shown in Figure 3.1, and the pseudocode is listed in Algorithm 3.1.

3.3.3. Key question when modifying architecture during training: where to get the weights from?

PBT relies on random perturbations of hyperparameters for exploring the search space while the network weights are being continuously trained. This works well when searching for hyperparameters that can be replaced independently of the weights: e.g., after changing the learning rate, the training can continue with the same weights.

However, searching for an architecture means introducing changes that impact the weights, e.g., changing the type of a layer from linear to convolutional. After such a change, the training process is disturbed: the weights of one type of layer cannot be used in another one.

[1] Note that since each network has a different architecture, it has a different training speed, so to avoid biasing the search towards models that require less training time, we use the synchronous variant of PBT.

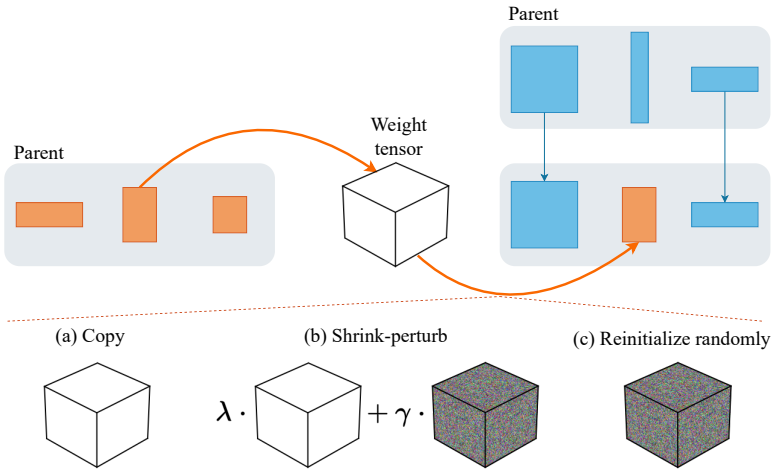


Figure 3.2: Three potential operations to perform on the weight tensor when copying the corresponding layer from the parent.

To follow the paradigm of PBT and continue training the network after an architectural change, the source of the weights needs to be determined. We consider three potential approaches (Figure 3.2).

One approach is initializing the new weights randomly. Intuitively, this could be problematic, as replacing weights of a whole layer with random ones can substantially disrupt the learned connections between neurons across the whole network.

Instead of being initialized randomly, the weights of the modified part can come from another network in the population. If the new value of the type of the layer is not generated randomly but copied from another solution, the corresponding layer weights can be copied from it too. Straightforward weight copying may be better than random initialization but it faces the following issue: even though the layers at the same depth of different networks should perform similar transformations (when trained on the same task), the actual data representations in each network are likely to be different. The copied weights would need to be adapted to a different representation space, but it might be difficult for gradient descent to adapt them quickly.

Shrink-perturb (Ash & Adams, 2020) is potentially helpful in this scenario. It was motivated by the observation that in online learning, continuing training from already trained weights when new data comes in can be worse than retraining from scratch using all the available data. Shrink-perturb consists of modifying the weights of a neural network by *shrinking* (multiplying by a constant λ) and *perturbing* them (adding noise multiplied by a constant γ; a new initialization of the network architecture is used as the source of noise).

Applying shrink-perturb to the copied weights is the middle ground between copying the weights as-is, and initializing them randomly. This preserves some useful information

Algorithm 3.2 `create_architecture`

Input: set of networks to potentially mix $nets$, probability p of replacing a layer, number of variables M, parameters λ, γ of shrink-perturb

$net_1, net_2 \leftarrow$ randomly sample from $nets$
if $net_1.fitness < net_2.fitness$ **then**
  $net_1, net_2 \leftarrow net_2, net_1$ # sort by fitness
$net_{new} \leftarrow$ `copy`(net_1)
for $i = 0$ **to** $M-1$ **do** # iterate over architecture variables
  if `random_uniform`$() < p$ **then**
    $net_{new}.\alpha_i \leftarrow net_2.\alpha_i$ # copy the value of the variable
    if $\exists net_2.W^i$ **then**
      # if the variable is a layer, copy and modify its weights
      $W_{new} \leftarrow$ `copy`$(net_2.W^i)$
      `shrink_perturb`$(W_{new}, \lambda, \gamma)$
      $net_{new}.W^i \leftarrow W_{new}$
return net_{new}

in the weights, while also potentially making their adaptation to the new architecture easier.

3.3.4. Mixing networks

Algorithm 3.2 shows our procedure for creating a new network. Firstly, two parent networks are randomly sampled from the top τ percentile of the population. An offspring solution is created by copying the better parent, and replacing with probability p each layer with the layer from the worse parent (including the weights, which are shrink-perturbed). Our mixing is a version of uniform crossover (Syswerda et al., 1989) where only one offspring solution is produced. Note that our mixing requires that layers in the same position can be substituted for each other (i.e., the output can be used as the input of the next layer), with the architecture remaining valid after a layer is replaced. We further discuss this limitation in Section 3.6.

Unlike existing approaches to combining neural networks (see Section 3.2.3), we do not expect (or need) the new network to perform well right away. Instead, it will be trained for several epochs in the next iteration of PBT-NAS, the same as the other networks in the population.

3.4. Experiment setup

3.4.1. General

We evaluate PBT-NAS on two tasks known to require careful tuning of network architecture and hyperparameters: GAN training and RL for visual control. In these settings, architecture can strongly influence performance (Sinha et al., 2020; Gui et al., 2023). We consider non-trivial architecture search spaces, see Sections 3.4.2 and 3.4.3. We would like to emphasize that achieving a state-of-the-art result on the chosen tasks is not our goal, instead we aim to demonstrate the feasibility of architecture search via simultaneous training and architecture mixing on tasks where performance strongly depends on architecture.

Hyperparameters of PBT-NAS are population size N, step size e_step, selection parameter τ (we use the default value from PBT, 25%, in all experiments), probability p of replacing a layer (which is also set to 25%). We aim to avoid unnecessary hyperparameter tuning to see if our approach is robust enough to perform well without it and to save computational resources.

The experiments were run in a distributed way, the details on used hardware and on GPU-hour costs of experiments are given in Appendix 3.F. The algorithms used the amount of compute equivalent to training N networks. Every experiment was run three times, we report the mean and standard deviation of the performance of the best solution from each run. We use the Wilcoxon signed-rank test (Wilcoxon, 1992) with Bonferroni correction (Dunn, 1961) for statistical testing (target p-value 0.05, 4 tests, corrected p 0.0125, mentions of statistical significance in the text imply smaller p, all p-values are reported in Appendix 3.C). Our code is available at https://github.com/AwesomeLemon/PBT-NAS, it includes configuration files for all experiments.

3.4.2. GANs

In AdversarialNAS (C. Gao et al., 2020), the authors describe searching for a GAN architecture (for unconditional generation) in a search space where random search achieved poor results — this motivated us to adopt this search space, which we refer to as `Gan`. In AdversarialNAS, both generator and discriminator architectures are searched for but we noticed that in the official implementation, the searched discriminator is discarded, and an architecture from the literature (X. Gong et al., 2019) is used instead. This prompted us to create an extended version of the search space (which we call `GanHard`) that includes discriminator architectures resembling the one manually selected by the AdversarialNAS authors. AdversarialNAS cannot be used to search in `GanHard` because some of the options cannot be searched for via continuous relaxation (one example is searching whether a layer should downsample: since output tensors with and without downsampling have different dimensions, a weighted combination cannot be created).

Specifics of search spaces are not critical for our research, so we give condensed descriptions here, see Appendix 3.E and our code for more details.

In `Gan`, operations for three DARTS-like (H. Liu et al., 2019) cells are searched (each cell is a Directed Acyclic Graph (DAG) with operations on the edges; in contrast to DARTS, each cell may have a different architecture). Additionally, inspecting the code of AdversarialNAS showed that the output of some cells is pointwise summed with a projection of a part of a latent vector. Each such projection is a single linear layer mapping a part of a latent vector to a tensor of the same dimensionality as the cell output: (#channels, width, height). These projections contain many parameters and are therefore an important part of the architecture. In the code of AdversarialNAS, these projections are adjusted for each dataset. As to the discriminator, the architecture from (X. Gong et al., 2019) is used.

Next, we describe `GanHard`. In `GanHard`, the parameters of the projections in the generator can be searched for. We additionally treat the layer mapping latent vector to the input of the generator as a projection, since it is conceptually similar. The projections (one per cell) can be enabled or disabled, except for the first one (connected to generator input) which is always enabled. A projection can take as input either the whole latent vector or the corresponding one-third of it (the first third of the vector for the first projection,

etc.). There are three options for the spatial dimensions of the output of a projection: *target* (equal to the output dimensions of the corresponding cell), *smallest* (equal to the input dimensions of the first cell), and *previous* (equal to the output dimensions of the previous cell). Since the projection output is summed with the cell output pointwise, the dimensions need to match, which is not the case for the last two options. To upsample the tensor to the target dimensions, either bilinear or nearest neighbour interpolation is used, which is also a part of the search space. Finally, given that a projection is a large linear layer with potentially millions of parameters (which makes overfitting plausible), we introduce an option for a dropout layer in the projection, with possible parameters $0.0, 0.1, 0.2$.

The discriminator search space in `GanHard` is based on the one in AdversarialNAS, the discriminator has 4 cells (each being a DAG with two branches), each cell has a downsampling operation in the end. However, we noticed that the fixed architecture from (X. Gong et al., 2019) that is used for final training of AdversarialNAS downsamples only in the first two cells. Additionally, in the first cell, the input is downsampled rather than the output. We amend the discriminator search space to contain a similar architecture. Firstly, we search whether each cell should downsample or not. Secondly, we add options for downsampling operations that are performed at the start of each branch rather than at the end of them. To enrich the search space further, we add two more nodes to each cell.

The number of variables in `Gan` is 21 and the size of the search space is $\approx 3.4 \cdot 10^{19}$. In `GanHard`, there are 72 variables (32 for the generator, 40 for the discriminator), and the size of the search space is $\approx 2.9 \cdot 10^{53}$.

Following AdversarialNAS, we run the experiments on CIFAR-10 (Krizhevsky & Hinton, 2009) and STL-10 (Coates et al., 2011), using both `Gan` and `GanHard`. In AdversarialNAS, the networks were trained for 600 epochs. We reduce that number to 300 epochs to save computation time (preliminary experiments showed diminishing returns to longer training), for the other hyperparameters, the same values as in AdversarialNAS are used.

The Frechet Inception Distance (FID) (Heusel et al., 2017) is a commonly used metric for measuring GAN quality. We use its negation as the objective function, computing it on 5,000 images during the search. For reporting the final result, the FID for the best network is computed on 50,000 images. We additionally report the Inception Score (IS) (Salimans et al., 2016), another common metric of GAN quality. The idea behind both FID and IS is to compare representations of real and generated images.

Based on preliminary experiments, the population size N is set to 24, and e_step is set to 10.

3.4.3. RL

We build upon DrQ-v2 (Yarats et al., 2022), a model-free RL algorithm for visual continuous control. DrQ-v2 achieves great results on the Deep Mind Control benchmark (Tassa et al., 2018), solving many tasks. Searching for architectures for solved tasks is not necessary, therefore for our experiments we chose tasks where DrQ-v2 did not achieve the maximum possible performance: Quadruped Run, Walker Run, Humanoid Run.

DrQ-v2 is an actor-critic algorithm with three components: *1)* an encoder that creates a representation of the pixel-based environment observation, *2)* an actor that, given the representation, outputs probabilities of actions, and *3)* a critic that, given the represen-

tation, estimates the Q-value of the state-action pair (the critic contains two networks because double Q-learning is used (Hasselt, 2010)).

We design the search space to include the architectures of all the components of DrQ-v2. Each network has three searchable layers. For the encoder, the options are Identity, Convolution {3x3, 5x5, 7x7}, ResNet (He et al., 2016) block {3x3, 5x5, 7x7}, Separable convolution {3x3, 5x5, 7x7}. For the actor and both networks of the critic, the available layers are Identity, Linear, and Residual (Bjorck et al., 2021) with multiplier 0.5 or 2.0. Additionally, we search whether to use Spectral Normalization (Miyato et al., 2018) (for each network separately) and which activation function to use in each layer (options: Identity, Tanh, ReLU, Swish). We also search the dimensionality of representation: in DrQ-v2 it was set to either 50 or 100 depending on the task, we have 25, 50, 100, and 150 as options.

There are 36 variables in total, the search space size is $\approx 4.6 \cdot 10^{21}$.

The hyperparameters of DrQ-v2 are used without additional tuning. For the Walker and Humanoid tasks, DrQ-v2 uses a replay buffer of size 10^6. Our servers do not have enough RAM to allow for such a buffer size when many agents are training in parallel, therefore for these tasks, we use a shared replay buffer (proposed in (Franke et al., 2021)): different agents can learn from the experiences of each other. To run in a distributed scenario with no shared storage, the buffers are only shared by the networks on the same machine. For fairness, all the baselines also use a shared buffer per machine.

Based on preliminary experiments, the population size N is set to 12. For the Quadruped and the Walker tasks, the DrQ-v2 agent used $3 \cdot 10^6$ frames. We use the same number of frames per agent, which means that N times more total frames are used. For the Humanoid task, $3 \cdot 10^7$ frames were used in DrQ-v2, we use only $1.5 \cdot 10^7$ per agent to save computation time. For uniformness of notation with GANs, we also use "epoch" in the context of RL, one epoch is defined as 10^4 frames. This means that 300 epochs are used for the Quadruped and Walker tasks, the same as for GANs, and e_step is also set to 10. Similar to BG-PBT (Wan et al., 2022), our preliminary experiments showed that having longer periods without selection at the start of the training is beneficial, therefore during the first half of the training, e_step is doubled from 10 to 20 epochs for the Quadruped and Walker tasks. Since for Humanoid only half the training is performed (in terms of frames per agent), the step size is fixed at 100 epochs (scaled up from 10 proportionally to the increase in the number of frames).

3.4.4. Baselines

We consider two general baselines that parallelize well and that can search in the proposed challenging search spaces.

1. **Random search.** N architectures are randomly sampled and trained.

2. **SEARL-like mutation**. In order to fairly evaluate the performance of a mutation-based architecture search approach like SEARL (Franke et al., 2021), we replace the mixing operator of PBT-NAS with the mutation operator from SEARL, adapting it to be applicable to both GAN and RL settings: with equal probability, either *(a)* one variable in the architecture encoding is resampled, *(b)* weights are mutated using the procedure from SEARL, or *(c)* no change is performed.

AdversarialNAS is a specialized baseline only capable of searching in `Gan`. In (C. Gao et al., 2020), the performance for only one seed was reported. We run the official implementation with 5 seeds and report the mean and standard deviation of performance.

3.5. Results

3.5.1. PBT-NAS vs. the baselines

As can be seen in Table 3.1, PBT-NAS achieves the best performance among all tested approaches in all GAN settings[2]. The improvements in FID over both random search and SEARL-based mutation are statistically significant. Despite the claim of (C. Gao et al., 2020) that random search performs poorly in `Gan`, we find that the gap between it and AdversarialNAS (C. Gao et al., 2020) on CIFAR-10 is small, and the difference between all algorithms is overall not large. The decreased performance of random search in `GanHard` shows that `GanHard` is indeed a more challenging search space. The results of searching in this space for CIFAR-10 and STL-10 show a clear improvement of PBT-NAS over the baselines in terms of FID. IS is better in the majority of settings.

PBT-NAS is also the best among alternatives on RL tasks, achieving better anytime performance, as shown in Figure 3.3 (the improvements in score over both random search and SEARL-based mutation are statistically significant). For Walker Run, there is no meaningful difference between algorithms, as the task is solved by all tested approaches, demonstrating that for differences between performance of the algorithms to be clear, both the RL task and the search space need to be of significant complexity.

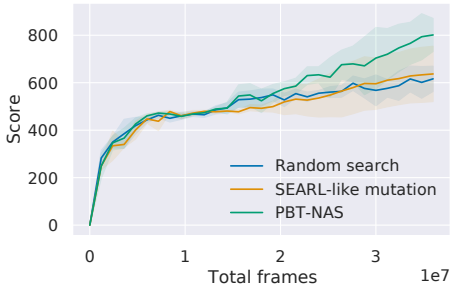
Table 3.1: Results for GAN training (mean ± st. dev.). The best value in each column is in bold.

Algorithm	CIFAR-10				STL-10	
	`Gan`		`GanHard`		`GanHard`	
	FID ↓	IS ↑	FID ↓	IS ↑	FID ↓	IS ↑
AdversarialNAS	$12.29_{\pm 0.80}$	$8.47_{\pm 0.14}$	—	—	—	—
Random search	$13.39_{\pm 0.28}$	$8.22_{\pm 0.15}$	$16.79_{\pm 0.97}$	$7.80_{\pm 0.14}$	$28.58_{\pm 1.77}$	$9.33_{\pm 0.18}$
SEARL-like mutation	$13.78_{\pm 1.02}$	$8.38_{\pm 0.05}$	$15.72_{\pm 2.22}$	$\mathbf{8.26}_{\pm 0.21}$	$26.94_{\pm 0.93}$	$9.66_{\pm 0.27}$
PBT-NAS	$\mathbf{12.21}_{\pm 0.16}$	$\mathbf{8.63}_{\pm 0.17}$	$\mathbf{13.25}_{\pm 1.64}$	$8.25_{\pm 0.27}$	$\mathbf{25.11}_{\pm 0.94}$	$\mathbf{9.71}_{\pm 0.09}$

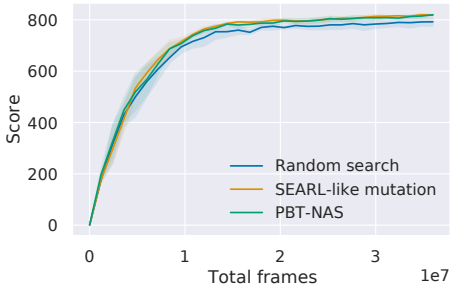
3.5.2. Mixing networks is better than cloning good networks

In order to show that creating new architectures makes a difference, we run a "No mixing" ablation: every component of PBT-NAS is kept the same, except that a new model is created by mixing a well-performing model with itself (rather than with another well-performing model). This way, no new architecture is produced, but the other benefits of PBT-NAS remain (e.g., replacing poorly-performing networks with well-performing ones). As seen in Table 3.2, this degrades the performance, clearly showing the impact that creating a better architecture can have.

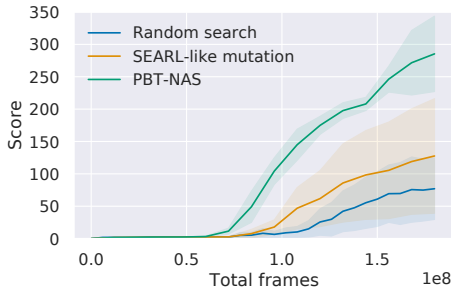
[2]When searching in `Gan` for STL-10, we faced reproducibility issues (despite using the official implementation), see Appendix 3.D for results and discussion.



(a) Quadruped Run



(b) Walker Run



(c) Humanoid Run

Figure 3.3: Results for RL tasks, mean ± st. dev. (shaded area).

Table 3.2: Results of ablation studies (mean ± st. dev.). The best value in each column is in bold.

Algorithm	FID ↓ (CIFAR-10, `GanHard`)	Score ↑ (Quadruped Run)
PBT-NAS (default)	$\mathbf{13.25}_{\pm 1.64}$	$\mathbf{801}_{\pm 70}$
No mixing	$14.90_{\pm 1.29}$	$672_{\pm 53}$
Shrink-perturb coefficients:		
[1, 0] — copy exactly	$14.94_{\pm 0.56}$	$699_{\pm 4}$
[0, 1] — reinitialize randomly	$15.06_{\pm 2.59}$	$532_{\pm 62}$

Table 3.3: The effect of using shrink-perturb in random search.

Use shrink-perturb in random search	FID ↓ (CIFAR-10, `GanHard`)	Score ↑ (Quadruped Run)
No (default)	$16.79_{\pm 0.97}$	$616_{\pm 53}$
Yes	$22.39_{\pm 2.64}$	$498_{\pm 30}$

3.5.3. Shrink-perturb is the superior way of weight inheritance

Table 3.2 shows that copying weights from the donor without change (shrink-perturb parameters $[1,0]$), or replacing them with random weights (shrink perturb $[0,1]$) leads to worse results in comparison to the usage of shrink-perturb. Thus, in our settings, using shrink-perturb is the best method to inherit the weights. The default parameters of shrink-perturb from (Zaidi et al., 2022) ($[0.4,0.1]$) worked well in PBT-NAS without any tuning.

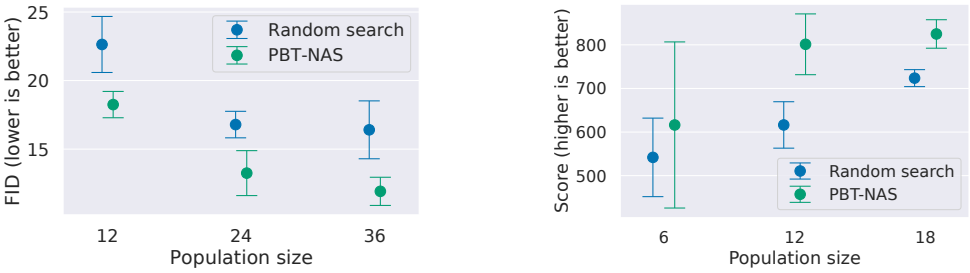
In (Zaidi et al., 2022), shrink-perturb was found to benefit performance, thus raising the question if using it gives PBT-NAS an unfair advantage that is not related to NAS. In order to test this, we added shrink-perturb to random search. As shown in Table 3.3, performance deteriorates, indicating that using shrink-perturb with default parameters in our setting is not helpful outside the context of NAS.

3.5.4. Increasing population size improves performance

We design our algorithm to be highly parallel and scalable. Figure 3.4 demonstrates that as the population size increases, the performance strictly improves (although diminishing returns can be observed). Given enough GPUs, the increased population size will not meaningfully increase wall-clock time, since every population member can be evaluated in parallel.

3.5.5. Model soups

As mentioned in Section 3.2.3, the idea of a model soup (Wortsman et al., 2022) is to improve performance by averaging weights of closely-related neural networks. As such, it seems like an especially good fit for the PBT setting: although the networks in the population start from different weights (and different architectures in the case of PBT-NAS), as worse networks are replaced by offspring of better networks, the population gradually converges. Since creating a model soup is done after training and requires a negligible amount of computation (evaluating at most N models), its inclusion into



(a) CIFAR-10, `GanHard` (FID ↓)

(b) Quadruped Run (Score ↑)

Figure 3.4: Impact of scaling population size, mean ± st. dev.

PBT-like algorithms could give an almost free performance improvement. Therefore, we create model soups following the greedy algorithm from (Wortsman et al., 2022).

Table 3.4 shows that soups improve GAN FID by approximately 0.4 points. For RL, however, there is no improvement when both the encoder and the actor are averaged (Table 3.5). We hypothesize that different actors may have dissimilar internal representations implementing different behaviour logic, unlike the encoders that only convert pixel inputs into representations. Therefore, we tried to separately average encoders, or actors. The results with averaged encoders are the best overall but they still do not lead to improved performance. For Walker Run, the task where performance is saturated, there is no difference between settings.

Previously, soups were only demonstrated for classification tasks, so it is interesting to see that they could also be beneficial in GANs. While no improvement was seen for RL, the fact that only the vision-related network, the encoder, could be averaged without large performance degradation hints at the limitations of the technique.

3.6. Discussion

We have introduced PBT-NAS, a NAS algorithm that creates new architectures by simultaneously training and mixing a population of neural networks. PBT-NAS brings the efficiency of PBT (designed for hyperparameter optimization) to NAS, providing a novel way to search for architectures. As computation power grows, especially in the form of multiple affordable GPUs, having parallelizable and scalable algorithms such as PBT-NAS becomes more important. At the same time, this computation power is not limitless, and reusing the partly-trained weights during architecture search is important from the perspective of search efficiency.

Currently, a large amount of effort in single-objective NAS research is directed at searching classifier architectures in cell-based search spaces, which are quite restrictive, and where random search achieves competitive results (Li & Talwalkar, 2019; Antoine Yang et al., 2020). We think that pivoting to more challenging search spaces and tasks could lead to NAS having a larger impact (e.g., in constructing state-of-the-art architectures, which is still mostly done by hand), and to comparisons between NAS algorithms leading to clearer differences. In Section 3.5.1, we showed that PBT-NAS could search in the

Table 3.4: The difference in metrics between a model soup and the best individual model (GAN), mean ± st. dev.

Dataset	GanHard	
	Δ FID ↓	Δ IS ↑
CIFAR-10	$-0.48_{\pm 0.34}$	$0.07_{\pm 0.05}$
STL-10	$-0.35_{\pm 0.26}$	$0.04_{\pm 0.25}$

Table 3.5: The difference in score between a model soup and the best individual model (RL), mean ± st. dev.

What to average	Δ Score ↑		
	Quadruped	Walker	Humanoid
Encoder	$-6_{\pm 10}$	$-1_{\pm 4}$	$-23_{\pm 19}$
Actor	$-196_{\pm 275}$	$-4_{\pm 7}$	$-244_{\pm 32}$
Both	$-142_{\pm 193}$	$0_{\pm 6}$	$-223_{\pm 20}$

challenging `GanHard` space, where an existing efficient algorithm, AdversarialNAS, could not be applied.

One limitation of exchanging layers during training is the requirement that different layer options (in the same position) need to be interoperable: the activation tensors they produce should be possible for the next layer to take as input (so that after replacing a layer, the architecture remains valid). This means that the number of neurons can be searched only when it does not influence the output shape. This could be addressed by e.g., duplicating neurons if there are too few of them and removing excessive ones if there are too many. Another limitation arises due to the greedy nature of PBT-NAS: architectures are selected based on their intermediate performance, and, therefore, sub-optimal architectures can be selected when early performance of an architecture is not representative of the final one.

Achieving good performance in different tasks with minimal hyperparameter tuning is a desirable property for a NAS algorithm. We used hyperparameters from the literature without tuning both in GAN training and in RL, as well as relying on default selection strategy from PBT. PBT-NAS outperformed baselines despite using these default values, tuning them could potentially further improve the results.

3.7. Conclusion

In this chapter, we designed and evaluated PBT-NAS, a novel way to search for an architecture by mixing different architectures while they are being trained. We find that adapting the weights with the shrink-perturb technique during mixing is advantageous compared to copying or randomly reinitializing them.

PBT-NAS is shown to be effective on challenging tasks (GAN training, RL), where it outperformed considered baselines. At the same time, it is efficient, requiring training of

only tens of networks to explore large search spaces. The algorithm is straightforward, parallelizes and scales well, and has few hyperparameters.

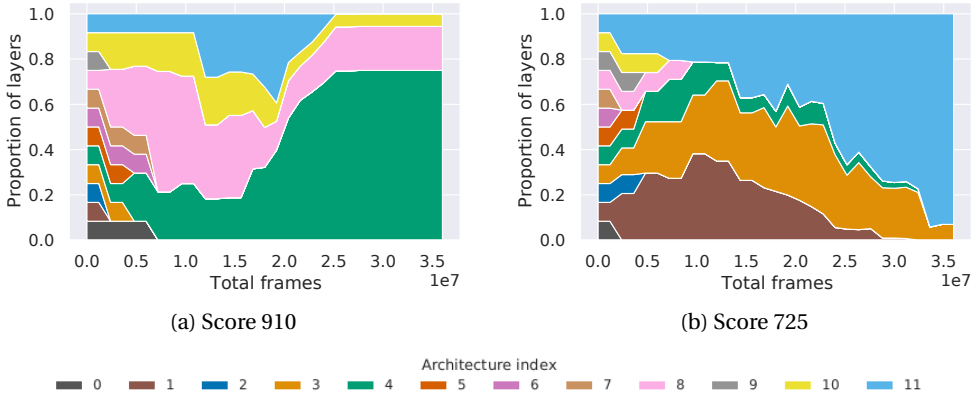
While in this work only NAS was considered, in the future, PBT-NAS could be adapted to simultaneously search for hyperparameters of neural network training, and of the algorithm itself, both of which would be necessary in order to fully automate the process of neural network training.

Appendix

3.A. Visualizing search progress

In order to visualize the search process, we track the origin of all the layers in the population. Initially, the layers of the i-th population member are specified to have origin i. When a new network is created by mixing different networks, its layers will have different origins. Over time, as worse-performing networks are replaced with the results of mixing better-performing ones, the successful layers constitute an increasing proportion of all layers in the population.

We visualize the experiments on the Quadruped Run task that achieved the best (Figure 3.5a) and the worst (Figure 3.5b) scores (out of the three seeds). In the experiment in Figure 3.5a, many architectures are successfully mixed, with the final population containing layers coming from several initial architectures. On the contrary, in the experiment in Figure 3.5b, one architecture largely takes over, with a small part inherited from a second architecture.



(a) Score 910

(b) Score 725

Figure 3.5: Proportion of layers from different initial architectures in the population over time (Quadruped Run). Experiments with the best (a) and worst (b) scores are shown.

3.B. Additional Bayesian optimization baseline

In this section we compare PBT-NAS to an additional baseline, Bayesian Optimization Hyperband (BOHB) (Falkner et al., 2018). BOHB is a well-established, efficient, and parallel Bayesian Optimization (BO) algorithm designed for hyperparameter optimization and NAS.

BO algorithms can be applied to almost any NAS search space, but typically have limited parallelization capability, and can struggle in high-dimensional spaces, especially if there are too few evaluations available. Our experiments in the main text were conducted in exactly such a scenario: the search spaces were high-dimensional (the `GanHard` space has 72 variables, the RL space has 36), and the number of total evaluations is smaller than

Table 3.6: Results of running BOHB. The best value in each column is in bold.

Algorithm	FID ↓ (CIFAR-10, `GanHard`)	Score ↑ (Quadruped Run)
PBT-NAS	$\mathbf{13.25}_{\pm 1.64}$	$\mathbf{801}_{\pm 70}$
BOHB	$18.95_{\pm 1.48}$	$621_{\pm 28}$

the number of variables (24 evaluations for `GanHard`, 12 for RL). In addition, PBT-NAS and the baselines considered in the main text are fully parallelizable (unlike BO algorithms).

We report the performance of BOHB in two settings (in which all our ablations were done). BOHB is run with the same budget as PBT-NAS both for Reinforcement Learning (the Quadruped Run task, total number of epochs equal to fully training 12 architectures) and GAN training (`GanHard` space, CIFAR-10 dataset, total number of epochs equal to fully training 24 architectures). Same as in the main text, every experiment is repeated three times, we report the mean and standard deviation of the performance of the best architecture.

As can be seen in Table 3.6, BOHB is outperformed by PBT-NAS in both settings. The BOHB results for GAN training are particularly poor, we speculate that BOHB suffers from excessive greediness in this setting (it early-stops two-thirds of solutions at each fidelity). PBT-NAS, while also greedy, has lower selection pressure, so it is affected less.

3.C. Statistical testing

Table 3.7 lists p-values for one-sided Wilcoxon pairwise rank tests (Wilcoxon, 1992) with Bonferroni correction (Dunn, 1961). In the GAN setting, the null hypothesis is that Algorithm 1 has a higher FID than Algorithm 2. In the RL setting the null hypothesis is that Algorithm 1 has a lower score than Algorithm 2.

In the GAN setting, results for CIFAR-10 using `Gan`, `GanHard`, and STL-10 using `GanHard` are tested together to increase sample size (total sample size is 9). Similarly to increase sample size, in the RL setting all the tasks (Quadruped Run, Walker Run, Humanoid Run) are tested together (total sample size is 9). P-values below the significance threshold of 0.0125 are highlighted (target p-value=0.05, 4 tests, corrected p=0.0125).

Table 3.7: P-values of conducted experiments (values below the significance threshold of 0.0125 are highlighted).

Algorithm 1	Algorithm 2	Setting	P-value
PBT-NAS	Random search	GAN	0.001953125
PBT-NAS	SEARL-like mutation	GAN	0.00390625
PBT-NAS	Random search	RL	0.001953125
PBT-NAS	SEARL-like mutation	RL	0.005859375

3.D. Reproducibility issues with `Gan` on STL-10

In (C. Gao et al., 2020), results for STL-10 are reported on a single seed, we ran the experiment 5 times using the official implementation. The resulting FID for STL-10 increases by about 10 points, which is a substantial drop in performance (see Table 3.8). The immediate cause is the divergence of the training procedure before good results could be achieved. However, as the official implementation was used, it is unclear why this divergence appeared consistently for all seeds and why none of the five seeds achieved the performance reported in (C. Gao et al., 2020). The authors did not respond when we notified them of the problem. We are also aware of an independent reproduction attempt running into the same issue.

Since our code relies on the AdversarialNAS codebase, whatever the issue is, it influenced all our experiments with `Gan` on STL-10, bringing their validity into question. Nonetheless, for transparency, our results (for three seeds) are reported in Table 3.8. PBT-NAS achieves the best performance out of the approaches tested by us.

Note that we did not face the reproducibility issue with `Gan` on CIFAR-10, as the reproduced result (FID $12.29_{\pm 0.80}$) was close to the reported one (FID 10.87 (C. Gao et al., 2020)). Also of note is that with `GanHard` on STL-10, results were even better than in (C. Gao et al., 2020) (see Table 3.1): FID of $25.11_{\pm 0.94}$. Since the same training code was used for `Gan` and `GanHard`, this implies that architectures could be the root cause of the problem, with architectures from `Gan` being a poor fit for STL-10, in contrast to architectures from `GanHard`. This is not consistent with the results from the main text for CIFAR-10, where, controlled for the amount of computational effort, better architectures could be found in `Gan` than in `GanHard`.

We would also like to note that `GanHard` was designed before any experiments with STL-10 were run, precluding the possibility that better STL-10 results with `GanHard` (compared to `Gan`) were achieved by deliberate search space adaptation to the dataset. STL-10 was our test dataset, the results for which did not influence any design or hyperparameter choices made in this chapter.

Table 3.8: Additional results for GAN training (mean ± st. dev.).

Algorithm	STL-10	
	`Gan`	
	FID ↓	IS ↑
AdversarialNAS (reported in (C. Gao et al., 2020))	26.98	9.63
AdversarialNAS (reproduced)	$36.87_{\pm 3.62}$	$8.90_{\pm 0.32}$
Random search	$32.54_{\pm 3.20}$	$9.15_{\pm 0.08}$
SEARL-like mutation (Franke et al., 2021)	$33.98_{\pm 4.36}$	$8.98_{\pm 0.19}$
PBT-NAS	$29.51_{\pm 0.91}$	$9.19_{\pm 0.05}$

3.E. Illustrations of GAN search spaces

Figure 3.6 depicts the setup of projections in the generator search space introduced in AdversarialNAS. In Figure 3.7, discriminator cell search spaces for `Gan` and `GanHard` are shown.

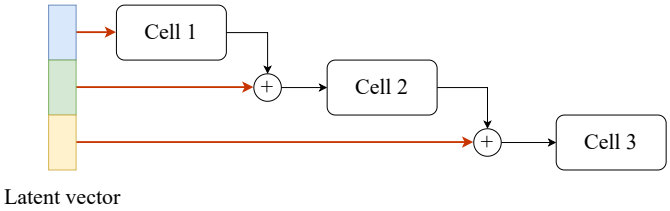
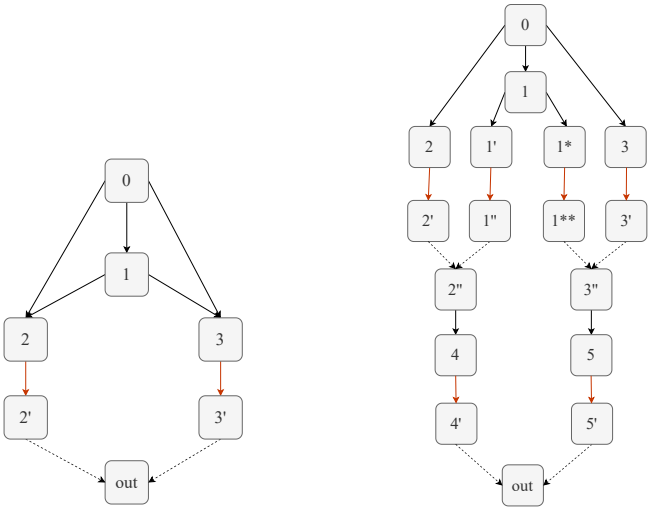


Figure 3.6: Projections from parts of a latent vector in the macro architecture of AdversarialNAS. Each projection (dark orange) is a linear layer, the output of which is reshaped and pointwise summed with the output of the corresponding cell (except for the first projection, the output of which is the input of the first cell).

3.F. Implementation details

The experiments were run on servers equipped with three Nvidia A5000 GPUs each. Each server is equipped with 2 Intel(R) Xeon(R) Bronze 3206R CPUs, and 96 GB of RAM. The used OS is Fedora Linux 36. The Ray (Moritz et al., 2018) framework was used to run the experiments in a distributed fashion. The configuration files specifying versions of all software libraries are included in the source code.

The experiment costs in total GPU hours vary by the setting (CIFAR-10 | `Gan`: 200, CIFAR-10 | `GanHard`: 270, STL-10 | `Gan`: 720, STL-10 | `GanHard`: 1200, Quadruped Run: 210, Walker Run: 330, Humanoid Run: 1300).



(a) `Gan` (b) `GanHard`

Figure 3.7: Discriminator cell search spaces. Black arrows correspond to normal operations (same as in (C. Gao et al., 2020), possible operations for the edges are None, Identity, Convolution 1x1 with Dilation = 1, Convolution {3x3, 5x5} with Dilation = {1, 2}); dark-orange arrows represent downsampling operations (Average Pooling, Max Pooling, Convolution {3x3, 5x5} with Dilation = {1, 2}; only parameter-less operations—Average Pooling, Max Pooling—are allowed at the start of the branch because they will be applied to several inputs); dashed arrows represent identity. All inputs to a node are summed. For `GanHard`, downsampling is either at the start or at the end of a branch (or disabled entirely).

4

To Be Greedy, or Not to Be — That Is the Question for Population Based Training Variants

The contents of this chapter are based on the following publication: Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2025b. To Be Greedy, or Not to Be - That Is the Question for Population Based Training Variants. *Transactions on Machine Learning Research,* ISSN: 2835-8856. https://openreview.net/forum?id=3qmnxysNbi.

Achieving excellent results with neural networks requires careful hyperparameter tuning, which can be automated via hyperparameter optimization algorithms such as Population Based Training (PBT). PBT stands out for its capability to efficiently optimize hyperparameter *schedules* in parallel and within the wall-clock time of training a single network. Several PBT variants have been proposed that improve performance in the experimental settings considered in the associated publications. However, the experimental settings and tasks vary across publications, while the best previous PBT variant is not always included in the comparisons, thus making the relative performance of PBT variants unclear. In this work, we empirically evaluate five single-objective PBT variants on a set of image classification and reinforcement learning tasks with different setups (such as increasingly large search spaces). We find that the Bayesian Optimization (BO) variants of PBT tend to behave greedier than the non-BO ones, which is beneficial when aggressively pursuing short-term gains improves long-term performance and harmful otherwise. This is a previously overlooked caveat to the reported improvements of the BO PBT variants. Examining their theoretical properties, we find that the returns of BO PBT variants are guaranteed to asymptotically approach the returns of the *greedy* hyperparameter schedule (rather than the optimal one, as claimed in prior work). Together with our empirical results, this leads us to conclude that there is currently no single best PBT variant capable of outperforming others both when pursuing short-term gains is helpful in the long term, and when it is harmful.

4.1. Introduction

Neural network training requires setting values of many hyperparameters, a process that is both crucial for good performance and time-consuming. Hyperparameter Optimization (HPO) algorithms aim to automate Hyperparameter (HP) tuning (Bergstra et al., 2011; Feurer & Hutter, 2019).

The Population Based Training (PBT) algorithm (Jaderberg et al., 2017) and its variants (Parker-Holder et al., 2020; Dalibard & Jaderberg, 2021; Parker-Holder et al., 2021; Wan et al., 2022) stand out for several reasons. Firstly, they are designed to efficiently optimize a schedule of HPs (rather than one set of fixed values), which improves performance (Loshchilov & Hutter, 2017; Tan & Le, 2021). Unlike other dynamic HPO algorithms (Baydin et al., 2018; Badia et al., 2020; J. Li et al., 2022), PBT is general, i.e., not restricted to a specific HP such as learning rate, or to a specific setting such as Reinforcement Learning (RL). Secondly, PBT runs within the wall-clock time of a single training of a network, which is desirable in practice. Thirdly, PBT is convenient to scale: adding parallel workers improves results without strongly influencing the wall-clock time of the algorithm. Finally, PBT can directly optimize non-differentiable objectives of interest, such as accuracy of a classifier, or score of an RL agent.

PBT was built upon in Population Based Bandits (PB2, Parker-Holder et al. (2020)) where Bayesian Optimization (BO) was introduced for sampling new HP values, PB2-Mix (Parker-Holder et al., 2021) where the BO setup of PB2 was extended to include discrete HPs, Bayesian Generational PBT (BG-PBT, Wan et al. (2022)) where a more ad-

vanced BO approach was used, and Faster Improvement Rate PBT (FIRE-PBT, Dalibard & Jaderberg (2021)) where the greedy nature of PBT was addressed. In each publication, the proposed variant outperformed previous variants included in the experiments, which unfortunately did not always include the best previous variant. Additionally, each subsequent publication performed evaluations on different tasks and in different setups (e.g., different implementations of RL tasks, different computation budgets).

These issues make it difficult to say if one of the PBT variants is substantially better than all others, and should therefore be preferred in practice. The question of whether there is one superior algorithm is the key question we aim to answer in this work. Towards that goal, we conduct an impartial empirical evaluation of the PBT variants in image classification and RL settings, using the FashionMNIST (Xiao et al., 2017), CIFAR-10/100 (Krizhevsky & Hinton, 2009), and TinyImageNet (Stanford CS231N, 2017) datasets for the former, and a subset of Brax tasks (Freeman et al., 2021) for the latter. Furthermore, we look into the theoretical foundations of the BO variants of PBT to better understand the proved guarantees and how they correspond to the behaviour of the algorithms in practice.

Although there exist extensions of PBT for multi-objective optimization (Dushatskiy et al., 2023) and architecture search (Franke et al., 2021; Wan et al., 2022; Chebykin et al., 2023), the focus of this article is on single-objective optimization of HPs only. Furthermore, we do not aim to compare PBT variants (which we also refer to as PBTs) to other HPO algorithms.

The main contributions of our paper are threefold:

- We investigate whether the more recently introduced PBT variants are always superior to the earlier ones on a set of image classification and RL tasks.
- We study how the performance and runtime of the PBT variants change when the task setup is varied.
- Theoretical and mechanistic causes for the observed differences in performance are explored.

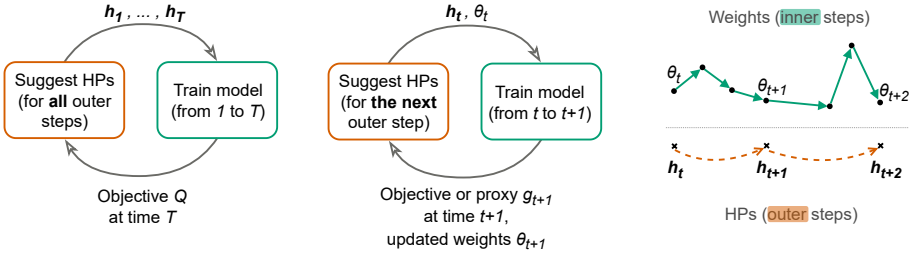
4.2. Dynamic HPO

4.2.1. Problem statement

We formalize the problem setting by building upon the notation of Jaderberg et al. (2017). In the single-objective setting, the goal is to maximize the final performance of a neural network by optimizing a schedule of HPs. Let us denote the HP search space as H, the HPs used during training between times t and $t+1$ as $\boldsymbol{h}_t \in H$, the maximum time as $T+1$, the weight space as W, the weights as $\theta_t \in W$, the objective function as $Q : W \to \mathbb{R}$, and the training procedure as $\texttt{train} : W \times H^T \to W$. The dynamic HPO optimization problem can be written as:

$$\{\boldsymbol{h}_t^*\}_{t=1}^T = \underset{\{\boldsymbol{h}_t\}_{t=1}^T}{\arg\max}\ Q(\texttt{train}(\theta_1 | \{\boldsymbol{h}_t\}_{t=1}^T)). \tag{4.1}$$

Note that while Eq. 4.1 formalizes *dynamic* HPO in that $\boldsymbol{h}$ varies over time, the optimization is written as *static* or non-sequential (arg max is applied to the entire schedule)



(a) Non-sequential optimization (Eq. 4.1)

(b) Sequential optimization (Eq. 4.2)

(c) Inner and outer optimization steps

Figure 4.1: Illustration of concepts relevant to the problem statement (see Section 4.2.1).

4

to correspond to the informal problem statement of maximizing the *final* performance. Drawing the distinction between optimizing a schedule statically (non-sequentially) or dynamically (sequentially) is important for explaining our theoretical results in Section 4.4. In Figure 4.1, panels (a) and (b) illustrate the non-sequential and sequential optimization loops.

For *sequential* optimization, consider $\texttt{step} : W \times H \to W$, a logical (outer) step of a PBT algorithm that corresponds to multiple inner steps (in which the weights are updated via, e.g., gradient descent). The hyperparameters are updated after each outer step. We can expand $\texttt{train}(\theta_1|\{\boldsymbol{h}_t\}_{t=1}^{T}) = \texttt{step}(\texttt{step}(\dots\texttt{step}(\theta_1|\boldsymbol{h}_1)\dots|\boldsymbol{h}_{T-1})|\boldsymbol{h}_T)$. If we denote the objective or its proxy after t steps as $g_t : H^t \to \mathbb{R}$, dynamic sequential HPO can be written as a sequence of problems:

$$\{\tilde{\boldsymbol{h}}_t\}_{t=1}^{T} = \{\arg\max_{\boldsymbol{h}_1} g_1(\boldsymbol{h}_1), \arg\max_{\boldsymbol{h}_2} g_2(\boldsymbol{h}_2|\tilde{\boldsymbol{h}}_1), \dots, \arg\max_{\boldsymbol{h}_T} g_T(\boldsymbol{h}_T|\tilde{\boldsymbol{h}}, \dots \tilde{\boldsymbol{h}}_{T-1})\}. \tag{4.2}$$

The sequential nature of optimization entails that previous choices cannot be altered. Consequently, directly optimizing Q at each step would make the optimization greedy, while optimizing an alternative g_t could lead to better long-term outcomes (as showcased by FIRE-PBT, see Section 4.3.1).

Let us further denote by $\text{step}_{\text{inner}}$ the number of inner optimization steps taken within each outer `step`. Since `step` is applied T times, the total number of inner optimization steps is $\text{total}_{\text{inner}} = T \cdot \text{step}_{\text{inner}}$. Figure 4.1c shows an example where each outer `step` corresponds to $\text{step}_{\text{inner}} = 3$ inner steps. For the visualized two outer `steps`, the total number of inner optimization steps $\text{total}_{\text{inner}}$ is equal to 6.

4.2.2. Dynamic HPO algorithms

In this section, we provide an overview of dynamic HPO algorithms that are not PBT variants. While there are many HPO algorithms that tune static values of HPs (Bergstra et al., 2011; Falkner et al., 2018; Lisha Li et al., 2018; Awad et al., 2021; Tan et al., 2024), few can efficiently tune schedules of HPs (although any HPO algorithm can learn an HP schedule by introducing separate variables, one for each HP at each time step, this is unlikely

to be efficient). One approach to dynamic HPO is to adjust the HPs via hypergradient descent (Baydin et al., 2018; J. Li et al., 2022). While using hypergradients can be efficient, it is also restrictive of HPs that can be tuned this way: e.g., only learning rate (in (Baydin et al., 2018)) or HPs that are both continuous and differentiable (in (Zahavy et al., 2020; J. Li et al., 2022)). Additionally, hypergradient algorithms introduce HPs of their own that require tuning (Paul et al., 2019; Y. Jin et al., 2021).

Another research direction focuses on dynamic HPO specifically for RL, which can be performed via bandit optimization (Badia et al., 2020), evolutionary methods (Tang & Choromanski, 2020), hypergradients (Zahavy et al., 2020), or by estimating HP quality via weighted importance sampling (Paul et al., 2019).

Biedenkapp et al. (2020) use RL for dynamic optimization in the general setting of dynamic algorithm configuration. While promising, the approach relies on RL that requires a large number of roll-outs to learn, thus making it too inefficient for dynamic HPO on time-consuming tasks such as neural network training (e.g., it required 40,000 training runs in the practical setting of tuning CMA-ES (Adriaensen et al., 2022)).

Unlike the approaches discussed above, the PBT variants are general algorithms that were empirically shown to be capable of tuning arbitrary HPs on challenging tasks. We further discuss them in Section 4.3.

4.3. PBT variants

4.3.1. Exploration via perturbation ("Perturbation PBTs")

PBT Jaderberg et al. (2017) introduced the first PBT algorithm for dynamic HPO. In PBT, the training of N models is interleaved with HPO. A solution p_i at (logical) time t is a tuple of model weights and HPs: $(\theta_t^i, \boldsymbol{h}_t^i)$. A population $P = \{p_i\}_{i=1}^N$ encompasses all solutions. At each time $t = 1 \dots T$, all models are trained in parallel[1] for $\text{step}_{\text{inner}}$ inner steps corresponding to one outer step: $\theta_{t+1}^i = \texttt{step}(\theta_t^i | \boldsymbol{h}_t^i)$, and evaluated: $q_{t+1}^i = Q(\theta_{t+1}^i)$.

After evaluation has been performed at time t, the `exploit` procedure is applied. Jaderberg et al. (2017) experimented with several `exploit` procedures and found truncation selection to perform best, which was then used in all the PBT variants we describe. In truncation selection, each of the λ% worst-performing models is replaced with a copy of one of the λ% best-performing ones (chosen randomly).

Finally, the `explore` procedure is applied to the copies created by the `exploit` procedure. In PBT, real-valued HPs are explored via perturbation: each HP is multiplied by a randomly chosen factor from a predefined set (e.g., $\{0.5, 2.0\}$); categorical HPs are randomly resampled. After the `exploit` step, the interleaved training and HPO continues with the updated population (as described above).

FIRE-PBT Dalibard & Jaderberg (2021) observed that PBT can behave too greedily, potentially harming its long-term performance. This is addressed by splitting the population into hierarchical subpopulations $P_{1..J}$ and a subpopulation of "evaluators". All subpopulations $P_{1..J}$ execute the standard PBT algorithm (in parallel) but optimize different objective functions. P_1 directly optimizes the target objective Q. Each subsequent subpopulation P_j optimizes an Improvement Rate (IR) objective defined relative to P_{j-1}.

[1] Note that we describe a synchronous version of PBT.

Omitting details, for $({}^{j}\theta_t^i, {}^{j}\boldsymbol{h}_t^i) \in P_j$, IR measures how fast Q improves when the weights ${}^{j}\theta_t^i$ are trained with the currently best HPs from P_{j-1}: ${}^{j-1}\boldsymbol{h}_t^*$. The "evaluators" enable the computation of IR by performing this training. Additionally, the weights trained by an evaluator can replace the weights associated with ${}^{j-1}\boldsymbol{h}_t^*$, thus allowing the weights from P_j to propagate to P_{j-1}. Using IR instead of Q in the subpopulations $P_{2..J}$ makes optimization less greedy, see Dalibard & Jaderberg (2021) for further details.

4.3.2. Exploration via Bayesian optimization ("Bayesian PBTs")

PB2 Parker-Holder et al. (2020) replace the random perturbation `explore` procedure of PBT with a BO approach based on Time-Varying Gaussian Process Bandits (TV-GPB) (Bogunovic et al., 2016). At each time t, changes in performance relative to the previous step are computed: $y_t^i = q_t^i - q_{t-1}^i$, and collected in a dataset: $D_t = D_{t-1} \cup \{(y_t^i, t, \boldsymbol{h}_t^i)\}_{i=1}^N$. Afterwards, a Gaussian Process (GP) model is fit to the dataset and new HPs are selected by optimizing an acquisition function that extends TV-GPB to a parallel setting. The theoretical guarantees for PB2 (and other Bayesian PBTs) are based on the TV-GPB formalism that is further discussed in Section 4.4.

PB2-Mix The BO model in PB2 is defined only for real-valued hyperparameters, while the categorical HPs are randomly resampled. To include categorical variables into the BO setup of PB2, Parker-Holder et al. (2021) introduce the `TV.EXP3.M` multi-armed bandit algorithm for the time-varying setting. `TV.EXP3.M` is used to sample the categorical variables, after which a time-varying GP model relying on a joint categorical-continuous kernel is learned and used to sample the continuous variables.

BG-PBT BG-PBT (Wan et al., 2022) builds upon the previous Bayesian PBT variants by explicitly considering ordinal variables in its extended version of the CASMOPOLITAN algorithm (Wan et al., 2021). This algorithm was chosen for its suitability to high-dimensional and mixed-input problems thanks to adaptations such as trust regions. BG-PBT places focus on enabling neural architecture search, which is out of scope for our article, we therefore omit the related adaptations from our analysis and experiments (which therefore apply to a variant of BG-PBT without architecture search).

4.4. Analysis

4.4.1. The returns of the Bayesian PBTs asymptotically approach the returns of the greediest schedule

Parker-Holder et al. (2020) formalize the problem of dynamic HPO as optimizing a time-varying objective function $F_t(\boldsymbol{h}_t)$. It is equivalent to $Q(\texttt{train}(\theta_1|\boldsymbol{h}_t, \{\tilde{\boldsymbol{h}}_i\}_{i=1}^{t-1}))$ in our notation: evaluating the weights trained with a schedule where the HPs before time t are fixed, while the HPs at t can be varied. The stated goal is to optimize the final performance $F_T(\boldsymbol{h}_T)$ with respect to a hyperparameter schedule $\{\boldsymbol{h}_t\}_{t=1}^T$. The optimization is sequential: the best choice at each step is defined as $\tilde{\boldsymbol{h}}_t = \arg\max_{\boldsymbol{h}_t} f_t(\boldsymbol{h}_t)$, where f_t is the improvement in F_t after one outer `step` of an algorithm: $f_t(\boldsymbol{h}_t) = F_t(\boldsymbol{h}_t) - F_{t-1}(\boldsymbol{h}_{t-1})$. This corresponds to the dynamic sequential HPO setup in Eq. 4.2 with $g_t = f_t$. The regret is defined as $\tilde{r}_t(\boldsymbol{h}_t) = f_t(\tilde{\boldsymbol{h}}_t) - f_t(\boldsymbol{h}_t)$. Let us reproduce Lemma 1 of Parker-Holder et al. (2020):

Lemma 1 (Parker-Holder et al. (2020))**.** *Maximizing the final performance F_T of a model with respect to a given hyperparameter schedule $\{\boldsymbol{h}_t\}_{t=1}^T$ is equivalent to maximizing the time-varying black-box function $f_t(\boldsymbol{h}_t)$ and minimizing the corresponding cumulative regret $\tilde{r}_t(\boldsymbol{h}_t)$:*

$$\max F_T(\boldsymbol{h}_T) = \max \sum_{t=1}^{T} f_t(\boldsymbol{h}_t) = \min \sum_{t=1}^{T} \tilde{r}_t(\boldsymbol{h}_t).$$

We find two issues with this lemma. The first is a minor technicality unrelated to our main argument: `max` and `min` should be `argmax` and `argmin`. This is due to the proof (Eq. 4.3) using $F_T(\boldsymbol{h}_T) - F_1(\boldsymbol{h}_1)$ in place of $F_T(\boldsymbol{h}_T)$. Subtracting a constant (as $F_1(\boldsymbol{h}_1)$ is assumed to be) affects `max` but not `argmax`.

Secondly, we find that the lemma is proved in the setting of non-sequential optimization (Eq. 4.1) rather than the sequential optimization actually performed by PBTs (Eq. 4.2). We show that $\tilde{r}$, the regret relative to the schedule $\{\tilde{\boldsymbol{h}}_t\}_{t=1}^T$, lacks meaning in the non-sequential setting, rendering any conclusions about it moot. Finally, we show that the lemma does not hold in the sequential setting. Let us start with the original proof:

$$\max [F_T(\boldsymbol{h}_T) - F_1(\boldsymbol{h}_1)] = \max \sum_{t=1}^{T} [F_t(\boldsymbol{h}_t) - F_{t-1}(\boldsymbol{h}_{t-1})] = \max \sum_{t=1}^{T} f_t(\boldsymbol{h}_t) = \min \sum_{t=1}^{T} \tilde{r}_t(\boldsymbol{h}_t). \tag{4.3}$$

Let us replace `max` with `argmax`, specify the arguments, replace F_1 with F_0 to correct the indexing of the telescoping sum, and expand the last transition (in accordance to Parker-Holder et al. (2020)):

$$\begin{aligned}
\arg\max_{\{\boldsymbol{h}_t\}_{t=1}^T} [F_T(\boldsymbol{h}_T) - F_0(\boldsymbol{h}_0)] &= \arg\max_{\{\boldsymbol{h}_t\}_{t=1}^T} \sum_{t=1}^{T} [F_t(\boldsymbol{h}_t) - F_{t-1}(\boldsymbol{h}_{t-1})] \\
&= \arg\max_{\{\boldsymbol{h}_t\}_{t=1}^T} \sum_{t=1}^{T} f_t(\boldsymbol{h}_t) \\
&\overset{(4.4.3)}{=} \arg\max_{\{\boldsymbol{h}_t\}_{t=1}^T} \sum_{t=1}^{T} \left[f_t(\boldsymbol{h}_t) - f_t(\tilde{\boldsymbol{h}}_t) \right] = \arg\min_{\{\boldsymbol{h}_t\}_{t=1}^T} \sum_{t=1}^{T} \tilde{r}_t(\boldsymbol{h}_t).
\end{aligned} \tag{4.4}$$

Observe that due to the dependencies between the summands, the argument of `argmax` must be the entire schedule for the transitions in the original proof (Eq. 4.3) to hold as written. Thus, the lemma is proved in the non-sequential setting. Let us next consider transition (4.4.3) in Eq. 4.4: it holds because for each time t, $f_t(\tilde{\boldsymbol{h}}_t)$ is a constant (so these values do not influence `argmax`). However, this holds not just for $\{\tilde{\boldsymbol{h}}_t\}_{t=1}^T$ but for any schedule $\{\boldsymbol{h}_t^A\}_{t=1}^T \in H^T$: for each time t, $f_t(\boldsymbol{h}_t^A)$ is a constant, and therefore transition (4.3) will hold also with $f_t(\boldsymbol{h}_t^A)$ in place of $f_t(\tilde{\boldsymbol{h}}_t)$ on the right side. Therefore, maximizing f_t is equivalent to minimizing the cumulative regret relative to this arbitrary schedule, $r_t^A(\boldsymbol{h}_t) = f_t(\boldsymbol{h}_t^A) - f_t(\boldsymbol{h}_t)$.

Thus, we can use the proof of the lemma to conclude that the regret relative to all possible schedules is minimized, making the original result about the specific regret $\tilde{r}$ lack meaning. The notion of minimizing regret *relative to a schedule* is problematic if the

final performance is optimized *non-sequentially*: the optimal schedule will maximize F_T regardless of the schedule used to define the regret, as seen in the transition (4.3). The lemma ignores the sequential nature of optimization that is fundamental to the design of PBTs and that is required for a meaningful notion of regret relative to a schedule.

Next, we give a high-level argument that the lemma does not hold in the sequential setting, with details provided in Appendix 4.A. $f_t(\boldsymbol{h}_t)$ is defined as $F_t(\boldsymbol{h}_t) - F_{t-1}(\boldsymbol{h}_{t-1})$, which reduces to $F_t(\boldsymbol{h}_t)$ during sequential optimization because $F_{t-1}(\boldsymbol{h}_{t-1})$ is constant at time t. With $f_t(\boldsymbol{h}_t) = F_t(\boldsymbol{h}_t)$, performance at the current step is maximized rather than the final performance. Due to potential dependencies on the past choices, greedily optimizing F_t leads to worse final results (unless no dependencies are present, which would make the greedy solution optimal). Thus, contrary to the statement of Lemma 1, sequentially maximizing f_t is *not* equivalent to maximizing the final performance F_T.

In the theory underlying all Bayesian PBTs, Lemma 1 connects minimizing regret to achieving optimal final performance. The associated regret bounds still hold, and it is true that the returns of the schedules discovered by Bayesian PBTs asymptotically approach those of $\{\tilde{\boldsymbol{h}}_t\}_{t=1}^T$, as shown by Parker-Holder et al. (2020). Our contribution lies in reinterpreting these results: the returns achieved by the Bayesian PBTs asymptotically approach the returns of the *greedy* solution (which we find $\{\tilde{\boldsymbol{h}}_t\}_{t=1}^T$ to be) that will only be optimal if previous hyperparameter choices do not restrict achievable future results (which cannot be generally expected).

4.4.2. Limitations of the current theoretical approach

The theoretical asymptotic behavior, while informative, does not completely describe an algorithm's behaviour in practice. Since Bayesian PBTs work with a population, a solution that does not lead to the best performance at the current step may still survive until the next step, where it may actually perform the best. Together with the search space exploration performed by BO, this enables Bayesian PBTs to discover schedules different from the greediest one (note that the `exploit` procedure can also influence the (asymptotic) behavior despite not being included in the theory of Bayesian PBTs, see Appendix 4.B).

Nonetheless, Bayesian PBTs are solving an optimization problem that is fundamentally not aligned with the stated goal of optimizing the final performance. The TV-GPB formalization is well-suited for time-varying problems where previous choices do not affect future results, such as adjusting a thermostat (Bogunovic et al., 2016). However, dynamic HPO typically does not fit this criterion, as it is a time-linked problem (Bosman, 2005), meaning that previous choices can strongly influence downstream results: e.g., reducing the learning rate too fast may improve performance in the short term, but in the end leads to premature convergence and a suboptimal final result.

Note that the standard PBT optimizes the same greedy objective function as Bayesian PBTs, but it does so less effectively due to its simple exploration via random perturbation (in contrast to BO). However, since the greedy objective function being optimized does not completely align with the true objective function (the final performance), the less effective optimization of PBT can lead to better final results.

FIRE-PBT stands out because its higher-order subpopulations greedily optimize not the current performance but the improvement rate, which encourages better long-term

results. Changing the optimization problem is one way of aligning an online algorithm with the target objective, see Section 4.6 for further discussion.

4.4.3. Step size as a mechanistic cause for the greediness of the PBT variants

All PBT variants operate on two levels. As discussed in Sections 4.2.1 and 4.3, on the upper level, a PBT performs T outer `steps` of HPO, while on the lower level, within each outer `step`, the underlying models are trained for $\text{step}_{\text{inner}}$ gradient descent (GD) steps, to the total of $\text{total}_{\text{inner}} = T \cdot \text{step}_{\text{inner}}$ steps over the entire run. While the $\text{total}_{\text{inner}}$ budget is fixed, the number of outer `steps` T is a hyperparameter of PBT that can be set freely by appropriately adjusting $\text{step}_{\text{inner}}$: e.g., if the training budget is 100 epochs, they can be split into 100 outer `steps` of 1 epoch, or 10 outer `steps` of 10 epochs, etc.

PBT variants optimize the objective function after one outer `step`, which can correspond to an arbitrary number of inner steps. PBTs are blind to how long the inner optimization is: no matter if the weights are trained for a single GD step or for many epochs, it is still a single outer `step` for a PBT. What happens if we increase the number of outer steps T? Holding $\text{total}_{\text{inner}}$ constant, this requires decreasing $\text{step}_{\text{inner}}$. A PBT will still optimize one outer `step` ahead but fewer inner steps ahead. In terms of inner steps, the optimization becomes greedier, as shorter-term improvements are pursued.

Thus, one mechanistic cause for how greedy a specific PBT variant behaves is the number of outer steps T (or, equivalently, $\text{step}_{\text{inner}}$). It is not clear how T should be set, as none of the PBT publications discuss it, while effectively using different values: 200 in PBT, 20 in PB2, 50 in PB2-Mix, 150 in BG-PBT.

Decreasing T could decrease greediness, as it is equivalent to increasing $\text{step}_{\text{inner}}$, i.e., how many inner steps ahead the optimization is done. However, this would also make the schedule less granular and reduce the amount of exploration an algorithm can do, which could degrade performance in challenging search spaces.

The optimal amount of greediness and thus the optimal value of T (or $\text{step}_{\text{inner}}$) is likely to be task-specific, and may even vary during the optimization: Wan et al. (2022) empirically observe that their chosen $\text{step}_{\text{inner}}$ leads to excessively greedy behavior on some tasks, and add linear annealing of $\text{step}_{\text{inner}}$ for these tasks. We argue that $\text{step}_{\text{inner}}$ is an important hyperparameter which could influence the behavior of PBTs in *any* task. We empirically demonstrate in Section 4.5.4 that varying the number of outer `steps` can change not only the final performance but also which PBT variant performs best.

4.5. Experiments & Results

4.5.1. General setup

We compare the PBT variants on image classification, RL, and toy problems. In our main classification experiments, the PBTs optimize hyperparameters of a ResNet-12 (He et al., 2016) on Fashion-MNIST (Xiao et al., 2017) and CIFAR-10 (Krizhevsky & Hinton, 2009) datasets (see Appendix 4.C for further details).

In main RL experiments, a Proximal Policy Optimization (PPO) (Schulman et al., 2017) agent is trained on the Hopper and Humanoid tasks from the Brax library (Freeman et al., 2021). These tasks were selected because Wan et al. (2022) reported that PBTs tended to

get stuck in local optima if the step size was small and we wanted to see if Perturbation PBTs could avoid local optima better than Bayesian PBTs.

By default, only the Learning Rate (LR) is tuned in the classification experiments, i.e., the *small* search space is used (see Appendix 4.E for the description of search spaces) but we found that for the Humanoid task this leads to minimal performance differences between PBTs (see Section 4.5.5) which motivated using the *large* search space in the RL experiments. For classification, the *large* search space is used in the scaling experiments (Sections 4.5.6 and 4.5.7) so that the scaling of PB2-Mix could be observed. PB2-Mix is equivalent to PB2 in continuous search spaces and so is excluded from the experiments in the *small* space.

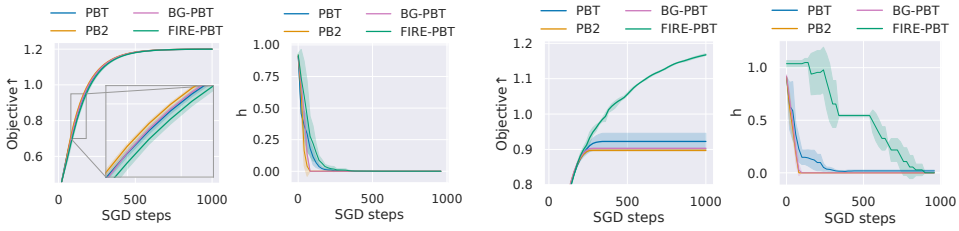
Unless specified otherwise, the PBTs are run with a population of size 22, see Appendix 4.D for the explanation, as well as other implementation details of PBTs. Five random seeds were used for the classification and toy experiments, while seven seeds were used for RL (known to have high variability). We typically plot Interquartile Mean (IQM) and Interquartile Range (IQR) in order to focus on average tendencies rather than outliers, similar to previous work (Parker-Holder et al., 2020). Statistical testing is described in Appendix 4.F.

4.5.2. Toy problems: plain and time-linked

In this section, we describe two toy problems: one where effective greedy optimization leads to better results, and another where the less greedy optimization is preferable.

The problems are created based on the toy problem from Jaderberg et al. (2017), where a quadratic function $Q(\theta) = 1.2-(\theta_0^2+\theta_1^2)$ is maximized without knowing its formula. Instead, a differentiable surrogate function is provided $\hat{Q}(\theta) = 1.2-(h_0\theta_0^2+h_1\theta_1^2)$, with $h = [h_0, h_1]$ as the hyperparameters to be optimized. The gradient is $\nabla_\theta \hat{Q} = -2h\theta$, therefore h should be greedily maximized to reach the optimum faster. We slightly adjust this problem and refer to it as `PlainToy`. Namely, we make θ one-dimensional, and replace h with $2-h$ to prevent h from going to infinity (while allowing it to increase up to 2). The objective is $g_1(\theta) = 1.2-\theta^2$, the surrogate is $\hat{g}_1(\theta) = 1.2-(2-h)\theta^2$, the gradient is $\nabla\hat{g}_1 = -2(2-h)\theta$. The initial values are uniformly randomly sampled from $[0.9, 1.1]$. This problem is designed so that the greedy behavior of reducing h to 0 as fast as possible is also the optimal behavior that leads to the best final result. On this problem, the greedier algorithms are expected to outperform the less-greedy ones.

Our second problem, `TimeLinkedToy`, is designed so that the greediest solution is optimal in the short-term but suboptimal in the long-term. It is an extension of the `PlainToy` with an additional penalty that is incurred by deviating from a linear decay schedule. As such, $2-h$ is replaced with $max(2-h-c\cdot p_{t-1}, 0)$, where $p_{t-1} = \sum_{i=0}^{t-1}|h^i - \frac{T-i}{T}|$ (h^i denotes the historical value of h at time i) and $c = 0.2$ (a constant to control the impact of the penalty). The "max" function is added to prevent the surrogate gradient from degrading the objective when the penalty grows large (making the gradient incorrect). The objective is $g_2(\theta) = 1.2-\theta^2$, the surrogate is $\hat{g}_2(\theta) = 1.2-\max(2-h-c\cdot p_{t-1}, 0)\cdot\theta^2$, the gradient is $\nabla\hat{g}_2 = -2\cdot max(2-h-c\cdot p_{t-1}, 0)\cdot\theta$. With $c < 1$, rapidly reducing h gives the best results in the first steps but the penalty gradually adds up, eventually stopping all progress.



(a) `PlainToy` (b) `TimeLinkedToy`

Figure 4.2: The performance of PBTs on the toy problems. For each problem, on the left, the IQM of the objective is shown (with IQR shaded), and on the right, the average value of the hyperparameter h (with the standard deviation shaded).

4

4.5.3. Bayesian PBTs underperform Perturbation PBTs when greedy choices are harmful in the long term

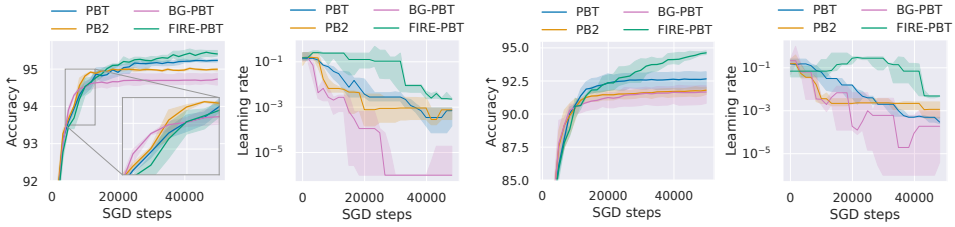
Based on our theoretical analysis in Section 4.4, Bayesian PBTs should perform worse than Perturbation PBTs when short-term improvement is detrimental to long-term success (and they should perform better if not). The goal of the experiments in this section is to empirically validate these claims.

Toy problems We start with the `PlainToy` problem, where the greediest solution is optimal. Figure 4.2a shows that all PBT variants successfully solve `PlainToy` by reducing the hyperparameter h to zero. Bayesian PBTs find the optimum faster than PBT, achieving better intermediate results. The less greedy FIRE-PBT reduces h the slowest, which leads to the slowest improvement (see the inset in Figure 4.2a (left)).

The results are different for the `TimeLinkedToy` problem (Figure 4.2b), where the greediest solution is optimal in the short but not the long term. Bayesian PBTs again quickly reduce h to zero, which leads to large penalties downstream due to deviating from the target linear decay schedule. Cumulative penalties eventually stop all progress of Bayesian PBTs, leading to poor final performance. Standard PBT, which cannot reduce the LR as quickly, performs better than the Bayesian PBTs, while FIRE-PBT achieves the best result by successfully addressing time linkage.

Classification Figure 4.3 shows the results of optimizing the LR of an image classifier. For both datasets, Perturbation PBTs on average outperform Bayesian PBTs in terms of final accuracy. This is due to Bayesian PBTs decaying the LR too aggressively. It is interesting to see in Figure 4.3a that the latest PBT variant, BG-PBT, reduces LR the fastest and on average performs better at the start of the training — only to be later overtaken by a less recent PB2 that reduces the LR somewhat slower. PB2, in its turn, is overtaken by PBT that reduces the LR even more gradually. However, by that point, the eventual winner, FIRE-PBT, already outperforms others despite starting off as the worst. Figure 4.3b shows similar behaviour.

These results are consistent with our analysis in Section 4.4. They clearly demonstrate in a practical setting that *more effective* Bayesian optimization can, in fact, be *greedier*



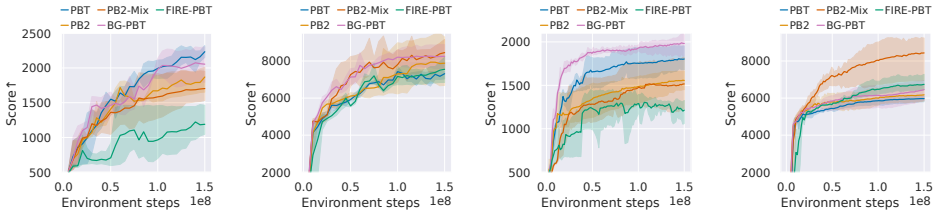
(a) Fashion-MNIST (b) CIFAR-10

Figure 4.3: The performance of PBTs on the classification tasks. For each dataset, on the left, the IQM of the accuracy is shown, and on the right, the median LR of the best solution. The IQRs are shaded.

4

optimization, ultimately leading to worse results. Note, however, that depending on the budget, different algorithms should be considered best: for Fashion-MNIST, BG-PBT performs best after $\approx 5{,}000$ steps, whereas PB2 is the best after $\approx 10{,}000$ steps. This implies that the properties of the task as well as the budget influence which PBT should be used for obtaining the best results.

Reinforcement learning Figure 4.4 shows the scores of the PBTs on RL tasks when either 30 or 100 outer steps are used. In the latter case, the algorithm with the highest final score on both tasks is a Bayesian PBT (BG-PBT for Hopper, PB2-Mix for Humanoid), performing well throughout the training. Although a pattern of rapid improvement followed by stagnation is visible in Figure 4.4b (similar to the classification tasks), Perturbation PBTs generally fail to overtake the Bayesian ones like they did in the classification tasks. When 30 outer steps are used (Figure 4.4a), Bayesian PBTs again tend to be better, although PBT marginally outperforms BG-PBT on Hopper (while continuing to perform poorly on Humanoid). We can conclude that in complex RL tasks and search spaces, the greedier but more effective optimization of Bayesian PBTs can lead to better results than the less greedy and less effective optimization of Perturbation PBTs.



Hopper Humanoid Hopper Humanoid

(a) 30 outer steps (b) 100 outer steps

Figure 4.4: The performance (IQM, with IQR shaded) of PBTs on the RL tasks with 30 or 100 outer `steps`.

4.5.4. The number of outer steps impacts the greediness and the relative performance of PBT variants

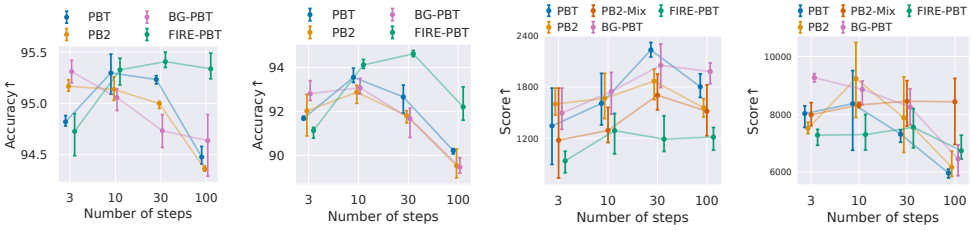
In Section 4.4.3, we reasoned that increasing the number of outer `steps` (while keeping the number of inner steps constant) should make a PBT variant greedier. Let us first examine the results for the classification tasks in Figure 4.5 (a, b). Bayesian PBTs perform well when the number of outer `steps` is very low (3) as they are able to effectively find good LR values despite having few exploration opportunities, unlike the less-effective Perturbation PBTs (which consequently perform worse in this setting). However, as the number of outer `steps` increases, the performance of Bayesian PBTs tends to worsen, since they start to effectively optimize the increasingly shorter-term objective.

On the other hand, Perturbation PBTs strongly improve when the number of outer `steps` goes from 3 to 10, as they have more opportunities to explore the search space but not enough to arrive at the greedy solution that optimizes short-term performance improvement. However, as the number of outer `steps` is further increased, the Perturbation PBTs become capable of effective greedy optimization, and their performance deteriorates (at 30 `steps` for PBT, at 100 `steps` for FIRE-PBT).

The impact of the number of outer `steps` on RL scores (Figure 4.5 (c, d)) does not follow a clear pattern. PBT peaks at different number of outer `steps` in different tasks while FIRE-PBT show consistently poor performance. BG-PBT tends to outperform PB2 and PB2-Mix but as the number of outer `steps` is increased, its performance improves on Hopper and deteriorates on Humanoid. As discussed in Section 4.5.3, greedy optimization does not appear as detrimental for these tasks as for the classification tasks, explaining absence of a similar interpretable pattern. Nonetheless, the number of outer `steps` affects both the absolute and the relative performance of the PBT variants (in most cases), highlighting the importance of this HP also in tasks without strong time linkage.

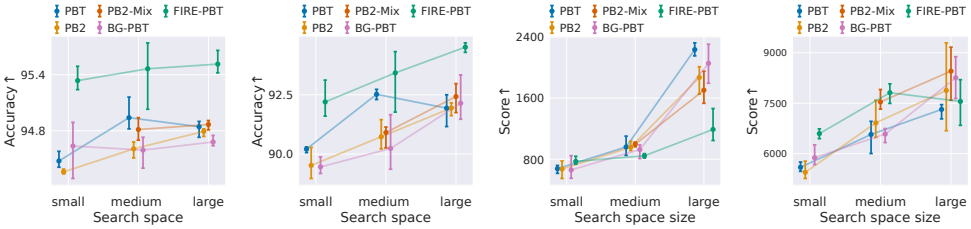
4.5.5. As the search space size is increased, performance of PBTs tends to improve

We investigated how the relative and absolute performance of PBT variants change when the size of the search space is varied. Figure 4.6 shows the performance of PBTs in search



(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

Figure 4.5: The performance (IQM and IQR) of the PBT variants as the number of outer `steps` is varied, while the number of inner steps (Stochastic Gradient Descent (SGD) steps for classification, environment steps for RL) is kept constant.



(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

Figure 4.6: The performance (IQM and IQR) of the PBT variants as the search space size is increased. Some IQR marks are invisible due to the IQR being too narrow.

spaces of different sizes (note that search spaces differ between the classification and RL tasks, see Appendix 4.E).

As can be seen in Figure 4.6 (a, b), for classification, FIRE-PBT tends to perform best in all search spaces, with performance increasing in larger search spaces (which shows that optimizing additional hyperparameters is beneficial). The performance of PBT improves when going from the *small* to the *medium* search space but deteriorates in the *large* space. The latter observation is likely due to inability of PBT to optimize the increased number of variables. The performance of Bayesian PBTs tends to increase with the search space size, although without reaching the level of performance of FIRE-PBT.

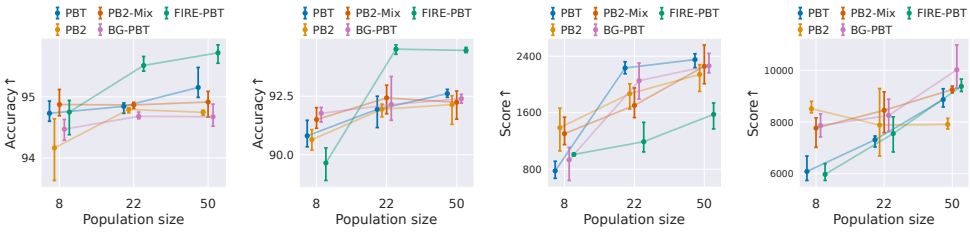
Figure 4.6 (c, d) shows that the performance of all PBT variants improves on the RL tasks as more HPs are optimized in larger search spaces. The relative performance of PBTs varies strongly, with no algorithm significantly outperforming others or even consistently achieving a higher average rank (see Appendix 4.F). Interestingly, the difference in performance of the same PBT variant *across* search spaces can be larger than the differences between PBTs in the *same* search space. The disappointing relative performance of FIRE-PBT in most RL search spaces despite its excellent performance across classification search spaces serves as evidence that FIRE-PBT is not generally superior to the other PBT variants.

4.5.6. As the population size is increased, performance of PBTs tends to improve

We investigated the effect of increasing the population size on the performance of the PBT variants. Bayesian PBTs were designed to work well with a small population (typically 8, although scaling to larger populations was shown to be beneficial) while Perturbation PBTs tend to use a population size of at least 20.

Consequently, even for the time-linked classification tasks, PBT and FIRE-PBT do not outperform the Bayesian PBTs if the population size is 8 (see Figure 4.7 (a, b)). However, as the population size increases, FIRE-PBT strongly outperforms all the other variants on the classification tasks, with PBT tending to be second-best as the population size reaches 50.

Figure 4.7 (c, d) shows that on RL tasks, Perturbation PBTs again perform poorly with a population size of 8. While they tend to improve with an increasing population size, they typically still underperform Bayesian PBTs, except for PBT on Hopper. PBT performs



(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

Figure 4.7: The performance (IQM and IQR) of the PBT variants as the population size is increased.

well not only with a population size of 22 (as previously seen in Section 4.5.3), but with a population size of 50 as well. Which properties of the task lead to these results is an open question. In general, performance of all PBTs tends to improve as the population size is increased, consistently with their nature as population-based algorithms. At the same time, the relative positions of the PBT variants changes across population sizes and tasks, with no algorithm significantly outperforming others, except for FIRE-PBT on classification tasks with population sizes of 22 and 50 (see Appendix 4.F).

4.5.7. PBTs scale differently in terms of wall-clock time

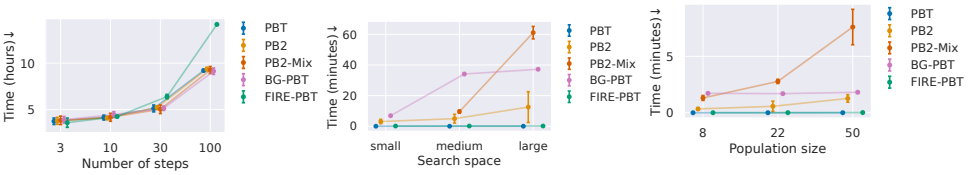
An attractive property of PBT variants is their ability to optimize hyperparameters within the wall-clock time of a single training run, if all N networks in the population are trained in parallel. We would like to add nuance by considering how much extra time the advanced PBT variants spend, and what they spend it on.

The first source of extra time spending is intermediate evaluations. At the end of each outer `step`, all networks need to be evaluated, with more outer steps requiring more evaluations and longer wall-clock time.

All PBT variants except for FIRE-PBT use the same number of evaluations. FIRE-PBT requires an order of magnitude more intermediate evaluations to compute the IR objective used in the higher-order subpopulations. As a result, FIRE-PBT tends to take the longest wall-clock time, which is noticeable for RL tasks where evaluations interrupt efficient Brax-enabled training. Figure 4.8 (a) shows that increasing the number of steps increases the total time spent on training and evaluation by all the algorithms but especially by FIRE-PBT.

The second source of extra time spending comes from selecting new hyperparameters in the `explore` step. The Bayesian PBTs optimize the hyperparameters of the GP and the acquisition function, while the random perturbation of PBT is near-instantaneous. FIRE-PBT fits several GPs when computing the IR objective. The cost of these additional operations could be influenced by the search space and the population size.

As can be seen in Figure 4.8 (b), increasing the search space size strongly affects only BG-PBT and PB2-Mix. BG-PBT slows down considerably when going from a continuous search space (*small*) to a mixed search space (*medium*), but does not meaningfully slow down when the number of variables is further increased in *large*. PB2-Mix, on the other



(a) Training and evaluation time as number of steps is varied

(b) Exploration time as search space is varied

(c) Exploration time (per population member) as population size is varied

Figure 4.8: Wall-clock time (mean ± st. dev.) taken by the PBTs on the Hopper task in various setups.

4

hand, is slower in the *large* search space than in the *medium* search space. This is consistent with BG-PBT relying on a more efficient BO algorithm. Figure 4.8 (c) shows that increasing the population size does not affect per-population-member update times, except for PB2-Mix: a larger population creates more data for the fitting of a GP, which appears to slow down more than linearly (which would correspond to a horizontal line). The exploration of FIRE-PBT incurs negligible costs, as does the exploration of PBT.

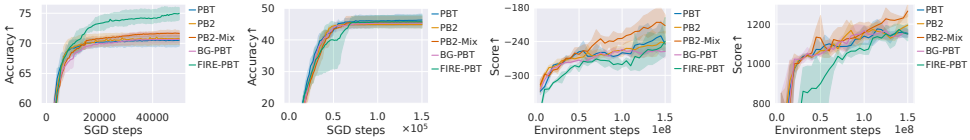
4.5.8. Do our observations generalize to similar settings?

Our computation budget is limited (since each PBT run entails training multiple neural networks, the total computational cost of the main experiments reach ≈ 900 GPU-days) but we would still like to know whether changes in the experimental setup would substantially alter the results. For this purpose, we run additional experiments in selected settings.

Classification CIFAR-100 (Krizhevsky & Hinton, 2009) is a step-up in complexity from CIFAR-10 in number of classes. Tiny ImageNet (Stanford CS231N, 2017) is larger still in number of classes, number of samples, and resolution (which we upsample to 224^2). This motivates training a larger architecture (ConvNeXt-T (Z. Liu et al., 2022)) for 3 times more steps. We use the *large* search space, and 100 outer `steps`, further details are provided in Appendix 4.C.

Figure 4.9a shows that FIRE-PBT performs best on CIFAR-100, as expected. The standard PBT does not outperform the Bayesian PBTs, likely due to the search space being large, which is consistent with our findings presented in Section 4.5.5. On Tiny ImageNet, however, FIRE-PBT manages only to catch up to other PBTs but is not able to overtake them. The combination of increased difficulty of the underlying task and the large search space may strain the simple exploration procedure of FIRE-PBT that nonetheless performs best in the *small* space, as expected (see Appendix 4.G). These results highlight the difficulty in choosing the PBT variant that would perform best for a specific task, search space, and budget, as discussed in Section 4.5.3.

Reinforcement learning We chose to evaluate on Brax Pusher and Walker tasks because they were not previously optimized by any PBT variant. Figure 4.9 shows that the algorithm that achieves the best final result, PB2-Mix, tends to perform well throughout optimization, with no issues caused by greediness (similarly to what we observed on



(a) CIFAR-100 (b) Tiny ImageNet (c) Pusher (d) Walker

Figure 4.9: The performance (IQM, with IQR shaded) of PBTs on additional tasks.

Hopper and Humanoid). PB2-Mix outperforming BG-PBT on both tasks is consistent with our previous results, where PB2-Mix was also found to be better than BG-PBT in some settings. On the other hand, in some previously considered settings, BG-PBT tended to perform better, preventing the conclusion that one of the two algorithms is superior.

4.6. Discussion & Conclusion

In this work, we compared five single-objective PBT variants on image classification and RL tasks, and found no single best algorithm that is capable of outperforming others in all (or even most) considered settings. This is partially explained by different PBT variants exhibiting (and different tasks requiring) different degrees of greediness. If good long-term results are achieved by optimizing short-term performance gains, then effective optimization of Bayesian PBTs is beneficial. If, however, effectively optimizing short-term gains compromises long term results, then even the standard PBT can outperform Bayesian PBTs, with FIRE-PBT performing even better. In our experiments, optimizing RL hyperparameters fell into the first category, while optimizing the LR of an image classifier typically fell into the second one.

From a theoretical perspective (as discussed in Sections 4.4.1 and 4.4.2), the underwhelming results of BO in time-linked settings stem from effectively solving an optimization problem that is not fully aligned with the desired goal of maximizing final performance. Is there an alternative to formalizing dynamic HPO as optimization of a time-varying function? In Eq. 4.1, we defined as optimum the schedule optimized over all steps simultaneously, which is incompatible with the efficiency brought by the sequential optimization of PBTs. Nonetheless, greedy optimization could be improved upon, e.g., by using an estimate of the future performance as the objective (Bosman, 2005), or, as in FIRE-PBT, by letting a subset of the population optimize an alternative objective directed at long-term performance.

The objective of all PBT variants except FIRE-PBT is to optimize performance after one outer `step` of the algorithm. We observe that by changing how much inner optimization (e.g., via gradient descent) is performed within an outer `step`, we can influence how greedily an algorithm behaves on time-linked tasks (although it is not the only effect, as discussed in Section 4.4.3). Varying the step size can substantially affect both the absolute and the relative performance of the PBT variants in all settings, with no single best value across algorithms and settings. Potentially having to try several values reduces the benefits of PBTs: efficiency and running within the wall-clock time of a single training. In order for a more complete picture of the upsides and downsides of different approaches

to dynamic HPO to emerge, alternative dynamic HPO algorithms should be studied and compared both to PBTs and to each other.

PBT variants typically do not add much wall-clock time to the training of the underlying model (if it is large enough), with the exception of FIRE-PBT. Its success in optimizing time-linked problems comes at a cost of an order of magnitude higher number of evaluations, which can be ignored if the evaluations are fast but adds up if they are not (as in our RL experiments). Future work could investigate the possibility of performing fewer intermediate evaluations or formulating a more efficient alternative to the improvement rate objective.

While we separately varied the number of steps of PBTs, their population sizes, and the search-space sizes, we did not tune them jointly, nor have we fully explored all PBT hyperparameters (e.g., we varied λ of the truncation selection only in a few settings, as reported in Appendix 4.I; the perturbation factor of FIRE-PBT was also varied in a limited number of settings, see Appendix 4.J). Arguably, efficient HPO algorithms should perform well without requiring tuning of their HPs, as this only shifts the HPO problem to a higher level. While the untuned hyperparameters is a limitation of our work, we believe that our conclusions on the interplay of the greediness and performance of the PBT variants in time-linked settings are general, given their theoretical underpinnings that align with empirical results. Deeper investigation of the interaction between the hyperparameters, the performance, and the behaviour of PBTs is a potentially interesting subject for future work.

Although our theoretical and empirical results can explain behaviour of PBT variants optimizing the LR of a classifier, they are far from a complete explanation of behaviour of PBTs on arbitrary tasks. Further understanding why a PBT variant performs poorly or well on a specific underlying task is an intriguing avenue of future research.

Appendix

4.A. Detailed reasoning on Lemma 1 in sequential setting

Here we show that Lemma 1 does not hold in the sequential setting: specifically, that sequential maximization of f_t does not generally result in optimal F_T. Parker-Holder et al. (2020) consider as optimal the schedule

$$\{\tilde{\boldsymbol{h}}_t\}_{t=1}^{T} = \{\arg\max_{\boldsymbol{h}_1} f_1(\boldsymbol{h}_1), \arg\max_{\boldsymbol{h}_2} f_2(\boldsymbol{h}_2), \dots, \arg\max_{\boldsymbol{h}_T} f_T(\boldsymbol{h}_T)\}.$$

Recall that $f_t(\boldsymbol{h}_t)$ is defined as $F_t(\boldsymbol{h}_t) - F_{t-1}(\boldsymbol{h}_{t-1})$. For sequential optimization, $F_{t-1}(\boldsymbol{h}_{t-1})$ is constant at time t. Therefore, it does not influence the optimization of $f_t(\boldsymbol{h}_t)$, allowing us to replace $\arg\max_{\boldsymbol{h}_t} f_t(\boldsymbol{h}_t)$ with $\arg\max_{\boldsymbol{h}_t} F_t(\boldsymbol{h}_t)$, revealing that $\{\tilde{\boldsymbol{h}}_t\}_{t=1}^{T}$ is simply the greedy schedule $\{\arg\max_{\boldsymbol{h}_1} F_1(\boldsymbol{h}_1), \arg\max_{\boldsymbol{h}_2} F_2(\boldsymbol{h}_2), \dots, \arg\max_{\boldsymbol{h}_T} F_T(\boldsymbol{h}_T)\}$. Consequently, sequentially maximizing f_t is equivalent to greedily maximizing F_t at each t (in other words, optimizing a proxy f_t (rather than the objective itself, F_t) makes no difference in the outcome).

Such greedy maximization does not, in general, maximize the final performance. Although we and Parker-Holder et al. (2020) write F_t as a function of only $\boldsymbol{h}_t$, this is not correct in the context of dynamic HPO. Previous hyperparameter choices can influence future results via the weights inherited from the previous step (e.g., too high a learning rate can lead to divergence that cannot be recovered from). Thus, performance at step t is not generally independent of the previous steps, which can be made explicit: $F_t(\boldsymbol{h}_t|\boldsymbol{h}_1 \dots \boldsymbol{h}_{t-1}) = Q(\texttt{train}(\theta_1|\{\boldsymbol{h}_i\}_{i=1}^{t}))$.

Due to potential dependencies on the past choices, greedily optimizing F_t leads to worse results (unless no dependencies are present, which would make the greedy solution optimal):

$$\max_{\{\boldsymbol{h}_t\}_{t=1}^{T}} F_T(\boldsymbol{h}_T|\boldsymbol{h}_1 \dots \boldsymbol{h}_{T-1}) \geq F_T(\tilde{\boldsymbol{h}}_T|\tilde{\boldsymbol{h}}_1 \dots \tilde{\boldsymbol{h}}_{T-1})$$

$$\text{where } \{\tilde{\boldsymbol{h}}_t\}_{t=1}^{T} = \{\arg\max_{\boldsymbol{h}_1} F_1(\boldsymbol{h}_1), \arg\max_{\boldsymbol{h}_2} F_2(\boldsymbol{h}_2|\tilde{\boldsymbol{h}}_1), \dots, \arg\max_{\boldsymbol{h}_T} F_T(\boldsymbol{h}_T|\tilde{\boldsymbol{h}}, \dots \tilde{\boldsymbol{h}}_{T-1})\}.$$

To sum up, in the sequential setting, contrary to the statement of Lemma 1, maximizing f_t is *not* equivalent to maximizing the final performance F_T (even though it is equivalent to minimizing regret $\tilde{r}_t$).

4.B. Theoretical consequences of the `exploit` procedure: an example

One can imagine a variant of the truncation selection where only the best solution in the population survives, and all others are replaced, with the weights copied from this solution, and hyperparameters explored via BO. Asymptotically, as population size goes to infinity, the best greedy solution maximizing current performance will be discovered at each step t. Only the weights from this solution will be present in the population of the next step, thus making sure that all schedules discovered in the future have the "greedy"

hyperparameters at the time t. The final schedule will have the "greedy" hyperparameters at each step, i.e., it will be the greediest schedule.

On the other hand, if truncation selection with $\lambda < 50\%$ is used, the population size going to infinity allows for many suboptimal-in-the-short-term weights to be preserved at each step, likely preventing the dominance of the greediest solution.

This is only an illustrative example of the potential influence of the `exploit` procedure on the results that can occur despite the BO component remaining constant. We hope to encourage a deeper theoretical treatment of the `exploit` procedure in future research.

4.C. Task implementation details

Classification The default training hyperparameters are: 50,000 SGD steps with a batch size of 128, a learning rate of 0.1, weight decay set to 0.0005, and the use of RandAugment (Cubuk et al., 2020) with 1 augmentation of magnitude 10.

The CIFAR-10/100 (Krizhevsky & Hinton, 2009) datasets contain 50,000 training and 10,000 testing samples each, we further separate 10,000 images from the training set to use as the validation set. The Fashion-MNIST (Xiao et al., 2017) contains 60,000 training and 10,000 testing samples, we further separate 10,000 images from the training set to use as the validation set. The Tiny ImageNet (Stanford CS231N, 2017) dataset contains 100,000 training, 10,000 validation, and 10,000 test samples. The labels for the test set are not available, therefore we treat the original validation set as the test set and separate 10,000 images from the training set to use as the validation set.

For each dataset, the validation set is used during the search to estimate the quality of a solution. The results in the main text are reported on the validation set so that the algorithms would be compared in terms of the objective they were optimizing. It also allows us to compare the intermediate performance. The final models are also evaluated on the test set. We observe that the results are similar to those on the validation set, see Figures 4.12-4.14 (Appendix 4.H).

RL For the RL tasks, we follow the setup of Wan et al. (2022) where hyperparameters of a PPO (Schulman et al., 2017) agent are tuned for 150 million environment interactions on the tasks from the Brax library (Freeman et al., 2021). To avoid overfitting to a single seed, a different random seed is set in each outer `step` of a PBT. The test performance is evaluated on a previously unseen seed. Both the validation and the test rewards are averages over 256 parallel rollouts. The test results are presented in Figures 4.12-4.14 (Appendix 4.H). We would like to highlight that the validation and test performance on the RL tasks is almost identical. The common RL issue of the train-test performance gap (Eimer et al., 2023) was avoided by using a different random seed in each outer step of a PBT algorithm.

For further implementation details, see our codebase: https://github.com/AwesomeLemon/PBT-Zoo. The codebase includes configuration files for all experiments, as well as code for plotting and statistical testing.

4.D. PBT variants implementation details

Population sizes are set based on the requirements of FIRE-PBT: while the other PBTs have a single population, we use FIRE-PBT with two hierarchical subpopulations and

an evaluator subpopulation. To avoid the evaluators staying idle, their number should be equal to $(100 - \lambda)\%$ of the second subpopulation, where λ is the hyperparameter of the truncation selection (25% by default). We follow Dalibard & Jaderberg (2021) in using population sizes of 22 (two subpopulations of size 8, and 6 evaluators) and 50 (two subpopulations of size 18, and 14 evaluators). We additionally consider the smaller population size of 8, which is typically used by the Bayesian PBTs. In this setting, FIRE-PBT has two subpopulations of size 3, and 2 evaluators. The λ for FIRE-PBT is appropriately increased to 34% from the default 25%.

In Perturbation PBTs, we use $\{0.5, 2.0\}$ as factors for perturbation. For some hyperparameters with narrow ranges, a perturbation can only flip between the maximum and minimum values, preventing reasonable exploration. To address this, in our implementation, we check if the variable range is narrow like this, and if so, normalize the range to $[0.0, 1.0]$, perform random perturbation in the normalized range, and renormalize the resulting value back to the original range.

We implemented FIRE-PBT from scratch, as no implementation was available. PBT was also implemented from scratch. As to PB2, PB2-Mix, and BG-PBT, we built upon the official implementations and adapted implementations of PB2-Mix and BG-PBT to be task-agnostic. Furthermore, we have fixed several issues with the gradients of PB2 and PB2-Mix that were present in these implementations (the issues were confirmed by the original authors in private communication). Specifically:

- PB2: The gradient of variance should not square the length scale, as it is not squared in the forward path of the kernel (https://github.com/jparkerholder/PB2/blob/master/kernel.py, lines 31, 47). Alternatively, the length scale could be squared in both places.

- PB2: The gradient of variance should be multiplied by the value of the time kernel, which is independent of it and is multiplied by it in the forward path (https://github.com/jparkerholder/PB2/blob/master/kernel.py, lines 33, 47).

- PB2: The gradient of the length scale has an erroneous additional factor of "-2", which can be confirmed by taking the gradient of the kernel with respect to the length scale (https://github.com/jparkerholder/PB2/blob/master/kernel.py, line 48).

- PB2-Mix: The gradient of the length scale has an erroneous minus sign, which is absent in the gradient formula in the PB2-Mix publication itself (https://github.com/jparkerholder/procgen_autorl/blob/main/pb2_utils.py, line 248).

Based on our preliminary experiments, these changes did not impact the final results achieved by the algorithms but substantially improved their exploration speed. This happened because the errors led to some gradients having signs opposite to the correct ones, which deteriorated the performance of L-BFGS that relied on these analytical gradients to optimize the hyperparameters of the kernel.

4.E. Search spaces

In classification tasks, the *small* search space corresponds to searching just the learning rate. In the *medium* space, the number of augmentations in RandAugment and the augmentation strength are additionally considered. The *large* space also includes weight decay and momentum. See Table 4.1.

Hyperparameter	Type	Exponent base	Range
learning rate	real	10	[-6, 0]
number of augmentations	integer	✗	[1, 4]
augmentation strength	integer	✗	[1, 30]
weight decay	real	10	[-8, -2]
momentum	real	✗	[0.5, 0.999]

Table 4.1: Hyperparameter tuning ranges for the classification tasks. All variables above the first and the second dashed lines are included in the *small* and *medium* search spaces accordingly, while the full table corresponds to the *large* search space.

In RL tasks, our search space is based on that of Wan et al. (2022), with architecture-related hyperparameters omitted. We also narrow the ranges of the "batch size" and "number updates per epoch" hyperparameters in order to reduce the computational burden of the experiments. We refer to the resulting search space as *large*. The *medium* space is its subset that contains only "learning rate", "entropy coefficient", and "batch size". The *small* space contains "learning rate" only. See Table 4.2.

Hyperparameter	Type	Exponent base	Range
learning rate	real	10	[-4, -3]
entropy coefficient	real	10	[-6, -1]
batch size	integer	2	[8, 10]
discount factor	real	✗	[0.9, 0.9999]
unroll length	integer	✗	[5, 15]
reward scaling	real	✗	[0.05, 20]
number of updates per epoch	integer	✗	[2, 16]
GAE parameter	real	✗	[0.9, 1]
clipping parameter	real	✗	[0.1, 0.4]

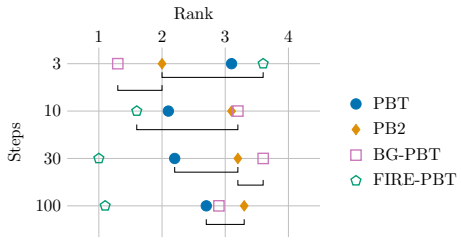
Table 4.2: Hyperparameter tuning ranges for the RL tasks. All variables above the first and the second dashed lines are included in the *small* and *medium* search spaces accordingly, while the full table corresponds to the *large* search space.

4.F. Statistical analysis

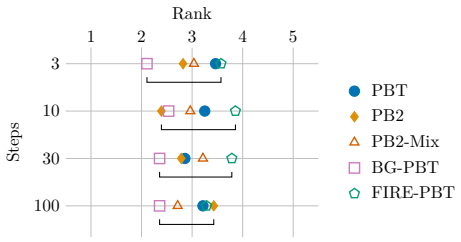
Due to high computational costs and the large number of settings considered in this article, we were able to run each algorithm with only a few random seeds in each setting:

5 seeds for the classification tasks, 7 seeds for the RL tasks. To partially counteract the lack of data, the results of the classification tasks were pulled together (so that each of the algorithm had 10 data points), and the results of the RL tasks were pulled together (so that each algorithm had 14 data points).

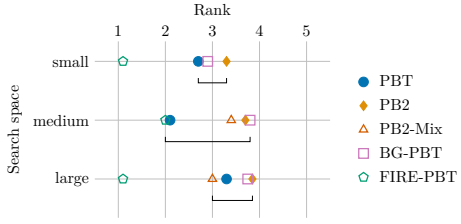
The tests were performed independently in each setting, with the target p-value of 0.05. Firstly, a Friedman test (Friedman, 1937) was performed to determine if any algorithm achieved results statistically different to those of the others. If so, pairwise signed Wilcoxon rank tests (Wilcoxon, 1992) with Holm correction (Holm, 1979) were performed. The results are presented in Figure 4.10 and referenced in the main text. We would like to additionally speculate that repeating experiments more times could lead to more significant differences found in some cases with large differences in average ranks (e.g., between PB2 and FIRE-PBT in Figure 4.10 (a) with 3 steps).



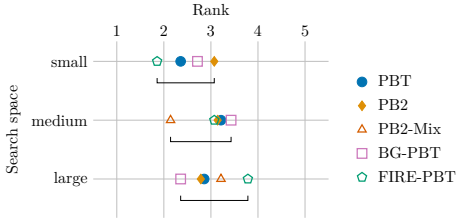
(a) Classification tasks: vary number of steps



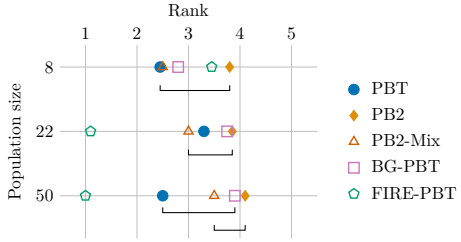
(b) RL tasks: vary number of steps



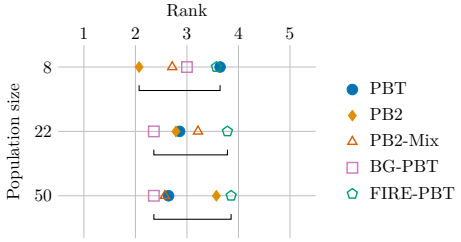
(c) Classification tasks: vary search space



(d) RL tasks: vary search space



(e) Classification tasks: vary population size



(f) RL tasks: vary population size

Figure 4.10: Critical difference diagrams for PBTs. Algorithms within a bracket were not found to be statistically significantly different from each other. Each row in each subfigure corresponds to one experimental setting that was analyzed independently of all others.

4.G. Additional results on Tiny ImageNet

In Section 4.5.8, we unexpectedly found that FIRE-PBT does not outperform other PBTs on Tiny ImageNet in the setting where it did so for all other datasets (Fashion-MNIST, CIFAR-10/100): *large* search space, 100 outer `steps`. We hypothesized that this could be due to the increased difficulty of the underlying task in combination with the *large* search space. To test the impact of the search space, we ran an additional experiment in the *small* search space. Figure 4.11 shows that in this setting FIRE-PBT outperforms other variants, as expected.

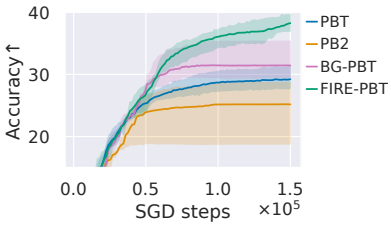
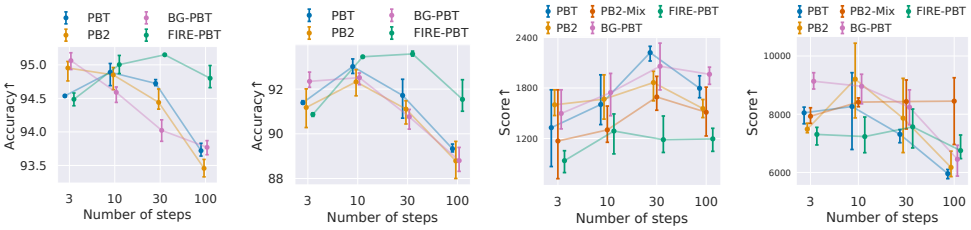


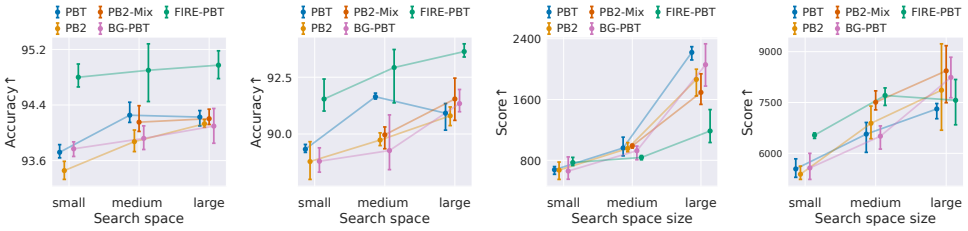
Figure 4.11: The performance (IQM, with IQR shaded) of PBTs on Tiny ImageNet in the *small* search space.

4.H. Results on the test split



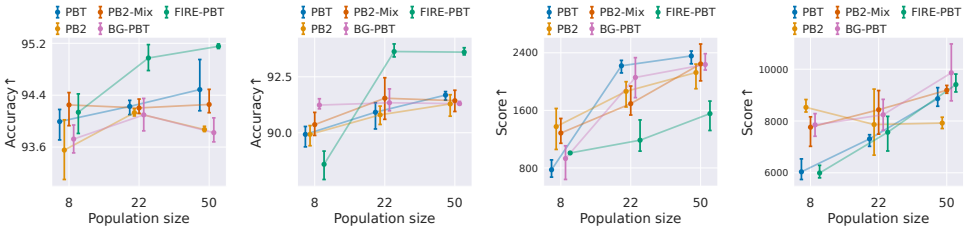
(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

Figure 4.12: Test performance (IQM and IQR) of the PBT variants as the number of outer `steps` is varied, while the number of inner steps (SGD steps for classification, environment steps for RL) is kept constant.



(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

Figure 4.13: Test performance (IQM and IQR) of the PBT variants as the search space size is increased. Some IQR marks are invisible due to IQRs being too narrow.



(a) Fashion-MNIST (b) CIFAR-10 (c) Hopper (d) Humanoid

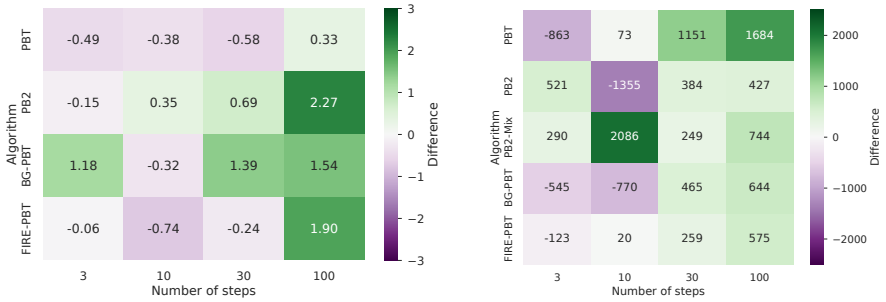
Figure 4.14: Test performance (IQM and IQR) of the PBT variants as the population size is increased.

4.I. Impact of λ on performance of the PBT variants

To evaluate the impact of the truncation selection parameter λ on the performance of PBTs, we repeat a subset of experiments from Figure 4.5 (specifically, those with CIFAR-10 and Humanoid) with the value of λ of truncation selection changed from the default 25% to 12.5%.

Figure 4.15 shows the difference in IQM that arise from such a change of λ. It can be generally observed that decreasing lambda improves results for some combinations of (algorithm, number of steps) but leads to worse results in others. Therefore, we cannot conclude that one of the considered values of lambda should be generally preferred.

Interestingly, all algorithms seem to benefit from the decreased lambda when the number of steps is 100. Since smaller λ corresponds to lower selection pressure and slower evolution, reducing it could be especially helpful in reducing short-term focus of optimization in this setting of many steps. We leave further investigation of the interaction between the hyperparameters and the performance of PBTs for future research.

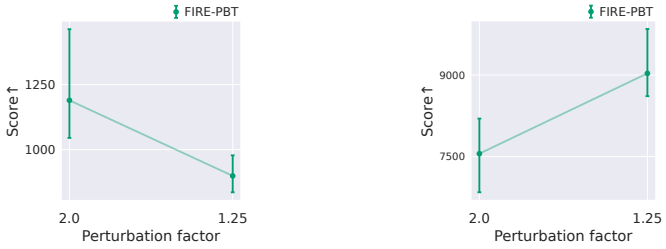


(a) CIFAR-10 (b) Humanoid

Figure 4.15: Differences in IQMs of the performance of the PBT variants when changing the value of λ of truncation selection from 25% to 12.5%.

4.J. Impact of the perturbation factor on performance of FIRE-PBT in challenging search spaces

In order to evaluate the potential impact of the perturbation factor hyperparameter of FIRE-PBT on its performance in larger and more challenging search spaces, we have run FIRE-PBT with the perturbation factor of 1.25 instead of 2.0 in the *large* search space on two RL tasks, Hopper and Humanoid. As can be seen in Figure 4.16, the smaller perturbation factor leads to a decreased score on Hopper and an increased score on Humanoid. This indicates that the perturbation-based search procedure is sensitive to the perturbation factor, which may need to be tuned per task to achieve optimal performance.



(a) Hopper (b) Humanoid

Figure 4.16: The performance (IQM and IQR) of FIRE-PBT when the perturbation factor is reduced from 2.0 to 1.25. The *large* search space is used.

5

Iterated Population Based Training with Task-Agnostic Restarts

Hyperparameter Optimization (HPO) can lift the burden of tuning hyperparameters (HPs) of neural networks. HPO algorithms from the Population Based Training (PBT) family are efficient thanks to dynamically adjusting HPs every few steps of the weight optimization. Recent results indicate that *the number of steps between HP updates* is an important meta-HP of all PBT variants that can substantially affect their performance. Yet, no method or intuition is available for efficiently setting its value. We introduce Iterated Population Based Training (IPBT), a novel PBT variant that automatically adjusts this HP *via restarts* that reuse weight information in a task-agnostic way and leverage time-varying Bayesian optimization to reinitialize HPs. Evaluation on 8 image classification and reinforcement learning tasks shows that, on average, our algorithm matches or outperforms 5 previous PBT variants and other HPO algorithms (random search, ASHA, SMAC3), without requiring a budget increase or any changes to its HPs.

The contents of this chapter are based on the following preprint: Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2025a. Iterated Population Based Training with Task-Agnostic Restarts. *Accepted for publication in the proceedings of the 43rd International Conference on Machine Learning (ICML 2026)*. arXiv: 2511.09190. https://arxiv.org/abs/2511.09190.

5.1. Introduction

Machine learning often achieves excellent results thanks to well-tuned hyperparameters (HPs). Hyperparameter Optimization (HPO) algorithms can reduce the time and effort needed for tuning (Feurer & Hutter, 2019; Tan et al., 2024).

Efficiency is important for HPO in general but it is vital when the underlying task is expensive, as is the case for neural network training, where often only a few training runs can be afforded. One HPO algorithm that is well-suited for this setting is Population Based Training (PBT) (Jaderberg et al., 2017). PBT draws its efficiency from dynamically adjusting HPs during training. Specifically, a population of networks – each with its own set of HPs – undergoes weight optimization via gradient descent for several steps. After this, poorly performing networks are replaced by copies of better-performing ones, with their corresponding HPs copied and perturbed.

Further efficiency gains can be achieved by replacing random perturbations with Bayesian Optimization (BO), an idea explored by several PBT variants (Parker-Holder et al., 2020; Parker-Holder et al., 2021; Wan et al., 2022). Despite the reported improvements, it was recently shown (Chebykin et al., 2025b) that none of the PBT variants consistently outperforms others, with substantial variability due to the value of the "step size" HP (i.e., how many weight update steps are performed before HPs are adjusted). The step size varies across papers with no discussion on how it is set for the tasks at hand or how it should be set for unseen tasks. Having to empirically determine a good value for this HP detracts from the utility and efficiency of PBT variants.

We aim to address this issue by having the step size adjusted automatically without sacrificing efficiency. Towards this goal, we introduce Iterated Population Based Training (IPBT), a novel PBT variant that performs several iterations of a PBT-like HPO procedure with a step size that is increased on each restart. A restart is triggered upon hitting diminishing returns, as determined by our novel data-driven mechanism. Crucially for efficiency, information is transferred across restarts in a task-agnostic manner: the partially-trained weights are shrink-perturbed (Ash & Adams, 2020) (i.e., reduced in magnitude and injected with noise), while HPs are reinitialized via a time-varying meta BO procedure that learns across restarts. Figure 5.1 presents an example run of our algorithm.

IPBT builds upon the idea of PBT with restarts as introduced in Bayesian Generational PBT (BG-PBT) (Wan et al., 2022). Key differences of IPBT relative to BG-PBT are the step size adjustment upon restarts, *task-agnostic* weight reuse after a restart, and reinitialization of HPs across restarts via time-varying BO. See Sections 5.2.2 and 5.3 for further details.

An important advantage of PBT variants is that they run in parallel, taking approximately the same wall-clock time as training a single network. To uphold this advantage, IPBT is designed not to increase the total number of weight update steps, restarting aggressively to most effectively utilize the limited budget. We explicitly target low-budget HPO, with the budget in our main experiments set to 8 full training runs. Additionally, we assume that no prior information about the task is available, motivating a black-box HPO approach.

We evaluate IPBT on image classification tasks using CIFAR-10/100 (Krizhevsky & Hinton, 2009), Fashion-MNIST (Xiao et al., 2017), and TinyImageNet (Stanford CS231N, 2017),

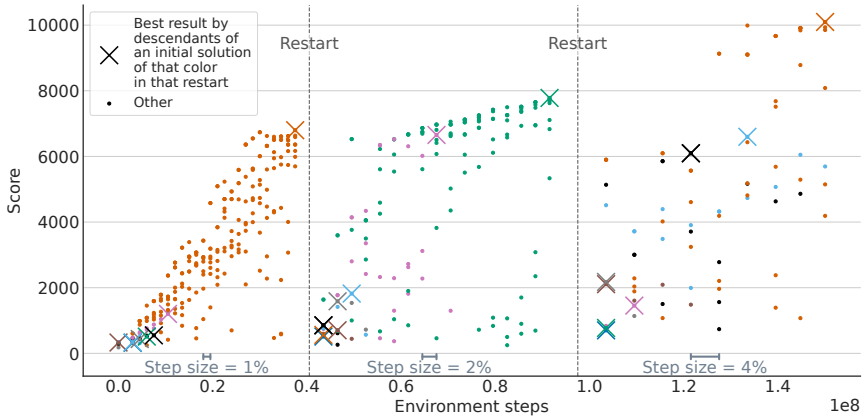


Figure 5.1: An example run of IPBT on the Humanoid task. Three iterations can be seen. Within each iteration, HPs are dynamically optimized during training via a PBT-like procedure (Section 5.2.2). If the performance is determined to be stagnating (Section 5.3.2), a restart is triggered. Upon restart, information from weights and HPs of previous iterations is reused (Section 5.3.3), and the step size of the PBT-like procedure is increased (Section 5.3.4). New HPs are predicted via a time-varying meta BO procedure which is trained on the initial HPs of previous iterations as inputs and their descendants' maximum achieved scores as outputs (in the figure, the descendants of each initial solution share its color). The pseudocode of IPBT is provided in Appendix 5.A.

as well as on reinforcement learning tasks using Brax (Freeman et al., 2021) Humanoid, Hopper, Pusher, Walker. IPBT is compared to 5 previous PBT variants (see Sections 5.2.2 and 5.4.1) with untuned and tuned step sizes, as well as Random Search (RS) (Bergstra et al., 2011), Asynchronous Successive Halving Algorithm (ASHA) (Liam Li et al., 2020), and SMAC3 (Lindauer et al., 2022). We additionally perform a thorough ablation study of the components of IPBT to better understand their influence.

Our main contributions are as follows:

- We introduce IPBT, a novel PBT variant that efficiently adjusts its step size via restarts.
- IPBT is evaluated on 8 image classification and reinforcement learning tasks, where it is compared with 5 previous PBT variants and other HPO algorithms (RS, ASHA, SMAC3) .
- We explore how different methods of reusing information contained in weights and HPs impact performance.

5

5.2. Related work

5.2.1. Efficient algorithms for expensive HPO

BO algorithms are a good option for expensive optimization tasks, including HPO (Snoek et al., 2012; Cowen-Rivers et al., 2022). However, since BO algorithms are typically initialized with as many random samples as the problem dimensionality, they face difficulties in low-budget HPO settings, where the budget in terms of full training runs is often smaller than the dimensionality. Therefore, while efficient algorithms for expensive HPO often have BO components, they pursue additional efficiency in other ways.

PBT variants (BO and non-BO) are efficient thanks to dynamic HPO where HPs are changed during training based on short-term feedback (we discuss these algorithms in Section 5.2.2). Alternatively, the HPs can be fixed during training but the training itself may be shorter, use a smaller model, or rely on some other proxy. These algorithms are called "multi-fidelity", as the proxy task may be parameterized to be more or less expensive (and accordingly provide information with higher or lower fidelity) (Won et al., 2025).

In Successive Halving (SH) (Jamieson & Talwalkar, 2016), N randomly sampled configurations are trained for a number of epochs. After this, all but the best $\frac{1}{\eta}$ configurations are stopped, and the remaining ones continue training for η times as many epochs (default $\eta = 2$). This procedure repeats until the maximum per-configuration budget R is reached. SH avoids wasting time on unpromising configurations but suffers from greediness and does not learn from experience. Hyperband (Lisha Li et al., 2018) improves upon SH by running multiple instances of it with different values of (N, R). Hyperband still relies on random sampling, which was replaced with BO in BO Hyperband (BOHB) (Falkner et al., 2018). Awad et al. (2021) showed that replacing BO with differential evolution further improves results, while Lindauer et al. (2022) demonstrated that using random forest as a predictor in BO performs even better, achieving excellent results as a part of the SMAC3 package. Throughout this paper, we use "SMAC3" to refer to the multi-fidelity setup of this package.

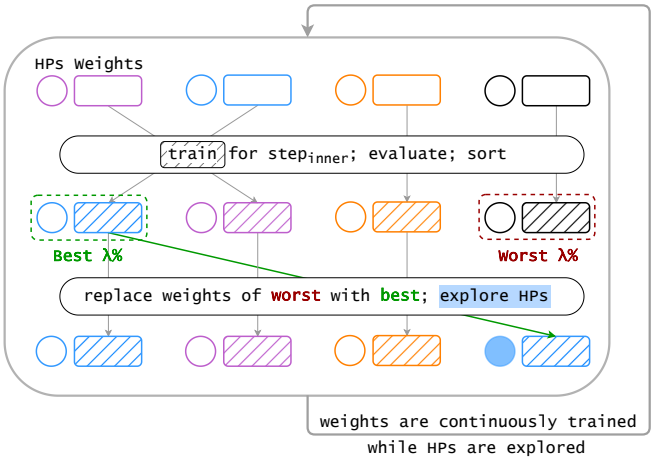


Figure 5.2: General PBT loop: in each outer step, the weights are trained for $\text{step}_{\text{inner}}$ inner steps and the HPs are explored.

In another research line, ASHA (Liam Li et al., 2020) effectively parallelized SH and was shown to outperform SH, Hyperband, and BOHB without any learning, thus making it the most effective approach based purely on random exploration.

5.2.2. PBT variants

As mentioned previously, PBT (Jaderberg et al., 2017) enables efficient HPO by dynamically adapting HPs while continuously training the weights. The optimization is bilevel, with inner steps updating the weights and outer steps updating the HPs. PBT operates with a population of N solutions where each solution at an outer step t is a pair of HPs and weights, $(\boldsymbol{h}_t^i, \theta_t^i)$. The weights of each solution θ_t^i are trained using the corresponding HPs $\boldsymbol{h}_t^i$ for $\text{step}_{\text{inner}}$ steps (e.g., via gradient descent), after which the weights are evaluated on a validation set. Each of the worst-performing λ% solutions is replaced with a copy of one of the best-performing λ% solutions (thus ensuring propagation of well-performing weights), and the HPs of the copies are explored via random perturbation. A graphical overview of the PBT loop is shown in Figure 5.2.

In Population Based Bandits (PB2) (Parker-Holder et al., 2020), BO was suggested as a replacement of random perturbation for exploring HPs. A dataset of changes in performance relative to the previous step (y_t^i) is updated after each outer step $(D_t = D_{t-1} \cup \{(y_t^i, t, \boldsymbol{h}_t^i)\}_{i=0}^{N-1})$ and used to fit a Gaussian Process (GP) model. New HPs are then chosen by optimizing an acquisition function that is derived from time-varying GP bandits (Bogunovic et al., 2016) (that are designed for dynamic problems in which the function being optimized changes over time).

PB2-Mix (Parker-Holder et al., 2021) extended the BO approach of PB2 to include categorical HPs. BG-PBT (Wan et al., 2022) further included ordinal HPs while also building upon a BO algorithm (Wan et al., 2021) that is well-suited to mixed-inputs and high-dimensional problems. In BG-PBT, restarts were introduced, with weight information

transferred via distillation. We discuss relevant components of BG-PBT when introducing our improvements in Section 5.3. BG-PBT was strongly focused on enabling Neural Architecture Search (NAS), which we do not address in our work.

Faster Improvement Rate PBT (FIRE-PBT) (Dalibard & Jaderberg, 2021) is a non-BO PBT variant that reduces the greediness of PBT by employing several subpopulations running the PBT algorithm with different objectives targeting short- or long-term performance.

The recently-proposed Multiple-Frequencies PBT (MF-PBT) (Doulazmi et al., 2025) algorithm addresses the issue of setting the step size by running several subpopulations with different step sizes. This performs well with large populations but seems infeasible for the ones that are too small to be split into multiple subpopulations of reasonable size. Doulazmi et al. (2025) further observed significant difference in performance between using their default population size 32 and the smaller 16. In contrast, IPBT is designed to efficiently adjust step size with a single small population, relying on restarts and information reuse.

It should be noted that since PBT variants modify HPs during training, a *schedule* of HPs is optimized, which is known to outperform fixed HP values (Tan & Le, 2021).

5.2.3. Optimization with restarts

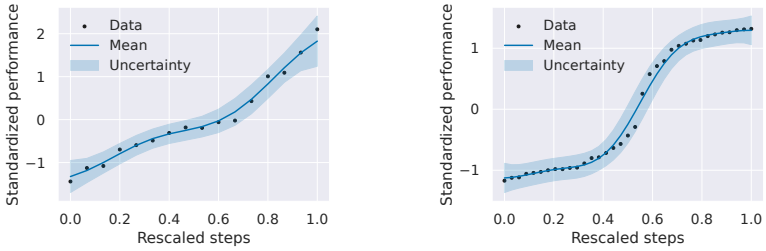
Restarting is a powerful idea in optimization: when the optimization procedure stagnates, it is restarted from a different initial point. In repeated local search, the new starting point is chosen randomly, while in iterated local search (Lourenço et al., 2003), the new point is chosen so that it is not too far from previously-determined good solutions nor so close as to converge to the same local optimum. A conceptually similar example of such a strategy in deep learning is Stochastic Gradient Descent (SGD) with restarts (Loshchilov & Hutter, 2017), where the learning rate is periodically increased (giving SGD an opportunity to escape from a local optimum) while the weights are retained (so the optimization does not deviate too radically from the previous solution).

The weights could also be modified according to the principles of iterated local search. Several algorithms (Sokar et al., 2023; Elsayed et al., 2024) were proposed for modifying previously trained weights to improve downstream optimization. Shrink-perturb (Ash & Adams, 2020) is one such algorithm that was shown to perform well in a variety of contexts (Zaidi et al., 2022; Chebykin et al., 2023): the pretrained weights are modified via shrinking (multiplying by an HP λ_{shrink}) and perturbing (adding a random reinitialization of the weights multiplied by an HP γ_{perturb}). This both preserves some information present in the weights and introduces randomness that may enable optimization to find a better solution.

5.3. Method

5.3.1. Overview

Each iteration of IPBT follows the general PBT procedure in Figure 5.2. The time-varying BO algorithm of BG-PBT is used to suggest new HPs. We also follow BG-PBT by starting optimization with $N_{\text{multiplier}}$ times more networks than the population size N and keeping only the best-performing N after the first step.



(a) (b)

Figure 5.3: Visualizations of the smoothed standardized scores when a trajectory is (a) improving vs. (b) stagnant.

Efficient adjustment of the step size during an IPBT run is achieved by stopping an iteration once the performance stagnates and restarting with a larger step size. This raises the questions of when to start a new iteration (Section 5.3.2), how to reuse information from the previous one (Section 5.3.3), and how to adjust the step size (Section 5.3.4).

5.3.2. Deciding when to restart

In a black-box setting where nothing is known about the underlying task, one way to select a step size is to empirically try several options. However, fully running a PBT variant several times would reduce efficiency. Efficiency can be preserved by allowing a run with a specific step size to continue only while the performance is strongly improving.

While in BG-PBT the step size was not modified on restarts, the design of its two restart-triggering stopping criteria is relevant. The first one checks that the performance has not improved for t_{patience} outer steps in a row. This, unfortunately, does not take into account the margin of improvement: a glacially but consistently improving run will not be stopped. This is addressed by the second criterion of BG-PBT: restarting after 40 million environment steps have elapsed since the previous restart (corresponding to 26.6% of the budget in the experiments of Wan et al. (2022); for some tasks, this value was set to 60 million steps (40% of the budget)). Since meeting one of the two criteria triggers a restart, the second criterion will eventually terminate the slowly improving run. However, it is not a priori clear how to set the size of the budget that must elapse before a restart is triggered. Additionally, restarting after a fixed budget is inflexible, as a run that improves too slowly cannot be stopped earlier while a strongly improving run cannot be continued for longer.

We propose a data-driven approach that is less susceptible to these issues. The essential questions to be answered are “is the run improving?” and “is it improving too slowly?”. To address the first question in the presence of noise, we collect the best performances at each previous outer step, z-score normalize these values, and smooth them via predictive parametric GP regression (Jankowiak et al., 2020) that deals well with heteroscedastic noise. The smoothed value at the current step is compared to that of the previous step, triggering a restart if the current value is not better for t_{patience} consecutive steps.

Even if the scores are improving, the rate of improvement may be too slow. Determining this is difficult in a black-box setting with no information about what a good score or a good improvement rate is (neither of which can be known for novel tasks). Therefore, the judgment needs to be made based on the observed performance within the run. Our second stopping criterion leverages the standardized performance data, triggering a restart if the performance has not improved by a standard deviation over the last t_{interval} outer steps. This is task-agnostic and allows the data to speak for itself: if the scores steadily improve, the difference of the current score with that of t_{interval} steps ago is large (allowing the run to continue while the steady gains persist). If, however, the gains are large at some point but later diminish, the performance t_{interval} steps ago will eventually be within a standard deviation, since standardization is done based on the entire trajectory (which incorporates information about the possibility of fast improvements). Figure 5.3 illustrates both cases.

5.3.3. Reusing information from previous iterations

Weights

Leveraging information contained in partially-trained weights is crucial for efficiency of the PBT paradigm. Introducing restarts requires figuring out how to transfer this information across them. The two simplest approaches are to either copy the weights from the previous iteration (thus treating the restart as any other outer step of a PBT) or to reinitialize them randomly (as the weights could have caused the stagnation of the previous iteration).

BG-PBT instead relies on distillation to transfer useful behavior from the previous iteration into the next one. This approach allows information transfer even across different architectures but comes with a serious downside: the distillation procedure and its HPs needs to be adapted to the task at hand. Clearly, a fixed distillation procedure cannot be applied across deep learning settings as different as reinforcement learning and image classification. For black-box HPO, a more general approach is desired.

Such an approach should work with the weights directly, since they are the only commonality in all deep learning scenarios. It should both preserve useful information present in the weights and enable downstream optimization to escape from their basin of attraction.

As described in Section 5.2.3, shrink-perturb (Ash & Adams, 2020) is a straightforward method that consists of copying the weights (multiplied by a constant) and adding noise to them. It is general, operates on weights directly, and both preserves information and injects noise. We integrate shrink-perturb in our algorithm as a method to modify the weights from the previous iteration after a restart. To improve chances of a successful perturbation, before applying shrink-perturb, weights outside the best λ% of the population are replaced with copies of the weights of the best λ%.

Additionally, we must consider that weights may end up in unfavorable regions of the optimization space which cannot be escaped via shrink-perturb. To avoid performance degradation in such settings, we randomly reinitialize the weights of half of the population upon restart (instead of shrink-perturbing them).

Hyperparameters

After restarting, the HPs of the solutions in the new population need to be set. Sampling them randomly neglects the previous experience. On the other hand, relying on past results could also be harmful, since the weights are trained continuously and HP values that were good initially may not be good for a later stage of the training.

In BG-PBT, an auxiliary global BO procedure was proposed. Firstly, a surrogate S_i is fit on the pairs of (HPs, score) at the latest iteration i. Secondly, the best HPs from previous iterations are found and evaluated using S_i. The dataset of these HPs and evaluations is used to fit an auxiliary surrogate S_A. Finally, $10 \cdot N$ random configurations are sampled, out of which N with the best values of an acquisition function (computed based on S_A) are kept. This procedure does not take all the historical trajectories into account (only one from each restart) and uses performance predicted by one model to fit another (which is likely to bias the results).

We propose an alternative approach, specifically, to perform time-varying BO *at the level of restarts*. That is, the HPs at the start of each iteration are optimized to maximize long-term performance within that iteration. This introduces an additional level of dynamic optimization to the PBT setup: while BO optimizes HPs within an iteration, meta BO optimizes the initial HPs across iterations.

Since HP values that lead to good results after a restart are expected to change as the weights are trained, it is reasonable to reuse the time-varying BO procedure of BG-PBT, fitting it on the long-term performance of initial HPs. To estimate the performance of a set of initial HPs, we track the history of the weights initially trained with them (before potentially switching to other HPs). The best performance achieved by any of these weights is taken as the long-term performance of the initial HPs. Many initial weights are removed from the population relatively quickly, resulting in shorter-term (and worse) performance compared to the best weights. We consider this desirable, as it steers HPO away from the initial HPs that lead to uncompetitive weights.

This procedure aims to learn from past information, which, however, may mislead the algorithm and bias it toward poorly performing local optima. The exploration performed by BO is limited, which is valuable if its model of the problem is correct, but in a black-box setting we must be prepared for any problem, including those that violate the assumptions made by BO. Therefore, similarly to how we randomly reinitialize the weights of half the population, we also sample random HPs for half the population (with sampling independent for weights and HPs).

5.3.4. Adjusting step size

BG-PBT introduced a decreasing step size schedule for certain tasks, which were selected manually because fixed-step-size BG-PBT performed poorly on them. It is not clear how such tasks could be identified in advance. Additionally, a deterministic schedule gives the algorithm no opportunity to retain a good step size for longer.

Whereas BG-PBT starts with a large step size, we propose starting with a small step size. An intuitive choice is the minimum reasonable amount of weight updates (e.g., an epoch), or 1% of the budget. Starting with a small step size leads to better use of budget in terms of inner steps: as HPs are updated more frequently, fast gains can be made. At

the same time, restarts with larger step sizes enable the algorithm to eventually target longer-term performance.

In IPBT, the step size is doubled on each restart. This exponential growth allows larger step sizes to be reached even with limited budget. Alternatively, linear growth can be considered, with the step size increased from $\text{step}_{\text{inner}}$ to $2 \cdot \text{step}_{\text{inner}}$ to $3 \cdot \text{step}_{\text{inner}}$, etc.

5.4. Experiments and Results

5.4.1. Setup

The performance of IPBT is evaluated on the 8 image classification and reinforcement learning tasks that were used by Chebykin et al. (2025b) to benchmark PBT variants. The setup of the tasks is followed exactly and the large search spaces are used. The classification tasks entail maximizing accuracy of a ResNet-12 (He et al., 2016) on CIFAR-10/100 (Krizhevsky & Hinton, 2009) and Fashion-MNIST (Xiao et al., 2017) datasets and of a ConvNeXt-T (Z. Liu et al., 2022) on Tiny ImageNet (Stanford CS231N, 2017). In the reinforcement learning tasks, the score of a proximal policy optimization (Schulman et al., 2017) agent is maximized on the Humanoid, Hopper, Pusher, and Walker tasks from Brax (Freeman et al., 2021). See Appendix 5.B for further details.

Across tasks, the attainable performance scores span different ranges. To enable easy comparison of general algorithm performance, we largely follow the methodology of Agarwal et al. (2021). Specifically, we repeat each experiment 8 times, normalize performance per task across all algorithms, and use stratified bootstrapping to estimate the Interquartile Mean (IQM) and the 95% Confidence Interval (CI). For statistical significance testing, a paired stratified bootstrap test (Efron & Tibshirani, 1993) with Holm correction (Holm, 1979) is used with $\alpha = 0.05$, as described in Appendix 5.C.

We compare IBPT to the 5 PBT variants available in PBT-Zoo (Chebykin et al., 2025b): PBT, PB2, PB2-Mix, BG-PBT, and FIRE-PBT (see section 5.2.2 and the original publications for details).

Since we are interested in efficient low-budget HPO, the population size of IPBT and PBT variants is set to 8. Although IPBT introduces multiple components with additional HPs, these are designed to work well out of the box and therefore remain fixed across all tasks (see Appendix 5.D for more details on the HPs). The default values and the components of IPBT are evaluated in our ablation studies (with performance evaluated on 4 tasks: CIFAR-10/100, Humanoid, and Hopper). To ensure unbiased performance estimates, the main experiments were run with seeds different from those used in the ablation studies.

IPBT is run with an initial step size of 1% of the budget (that is automatically adjusted). Other PBT variants are run with the step sizes from (Chebykin et al., 2025b): 1%, 3.3%, 10%, 33.3%. For each PBT baseline and for each task, the results across all steps are pooled to estimate untuned performance, while the results corresponding to the step size yielding the maximum performance are considered the tuned performance.

We additionally run non-PBT HPO algorithms: RS as the standard baseline, ASHA (Liam Li et al., 2020) as an efficient multi-fidelity extension of RS, and SMAC3 (Lindauer et al., 2022) as a BO multi-fidelity algorithm. Unlike PBT variants, these algorithms cannot directly search for hyperparameter schedules, placing them at a disadvan-

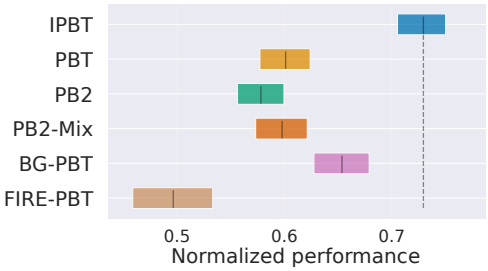


Figure 5.4: Normalized performance across 8 tasks (IQM and CI) of IPBT and *untuned* PBT variants.

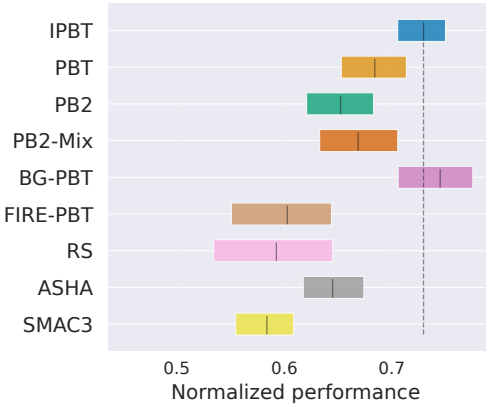


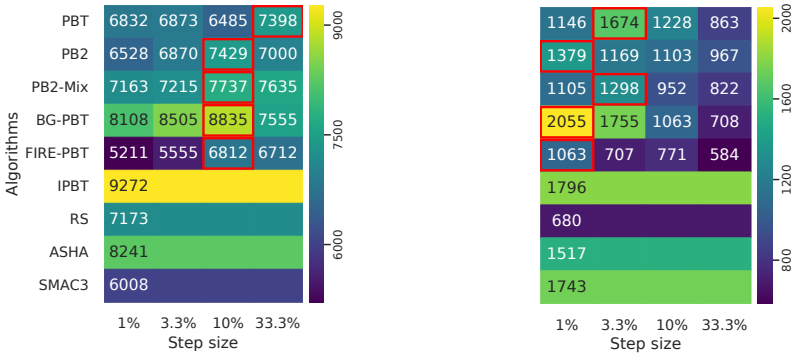
Figure 5.5: Normalized performance across 8 tasks (IQM and CI) of IPBT, *tuned* PBT variants, and baselines.

tage. To address this, we extend their search spaces to include parameters of a cosine learning rate schedule with restarts (Loshchilov & Hutter, 2017), thus allowing them to potentially find learning rate schedules similar to that of IPBT. The baseline algorithms are run with default HPs and the same budget as IPBT. See Appendix 5.E for further implementation details.

5.4.2. Overall performance

We start by comparing IPBT to previous PBT variants in the one-shot setting, where the step size of each of the PBT variants is not tuned. Figure 5.4 shows that IPBT strongly outperforms previous PBT variants, clearly demonstrating its utility for dynamic HPO without per-task tuning of the step size. The performance of PBT variants relative to each other is reasonable: the most recent Bayesian variant, BG-PBT, achieves the highest IQM, while FIRE-PBT, which uses several subpopulations, performs worst, likely due to the subpopulations being too small.

Selecting the best step size for each task for each PBT variant improves their absolute performance while keeping the relative performance unaffected, as can be seen in Fig-



(a) Humanoid (b) Hopper

Figure 5.6: IQM of scores of PBT variants with different step sizes (the best score of each variant is framed in red), IPBT, and baselines on the Humanoid and Hopper tasks.

5

ure 5.5. Furthermore, selecting the best-performing step size leads to BG-PBT achieving a higher IQM than IPBT, although the difference is not statistically significant. At the same time, IPBT performs statistically significantly better than all other PBT variants, despite using four times less compute (since it was run once, whereas each PBT variant was run four times with different step sizes). We conclude that IPBT should be preferred over other PBT variants, except for BG-PBT, which can achieve better results if its step size is tuned.

The need for tuning step sizes of PBT baselines can be clearly seen in Figure 5.6, which shows that PBT variants achieve the best performance with larger step sizes (10%, 33.3%) on Humanoid, and smaller (1%, 3.3%) on Hopper.

We further compare IPBT with non-PBT HPO algorithms that can be run out-of-the-box with the same small computation budget. Figure 5.5 shows that IPBT achieves a much better IQM (statistically significant) than RS, ASHA, and SMAC3. The unexpectedly poor performance of SMAC3 (on par with RS) led us to investigate, revealing that it is likely due to the algorithm allocating a large fraction of the budget to cheap evaluations and running few expensive ones. This strategy works well for some tasks (e.g, Hopper in Figure 5.6) but not others (e.g., Humanoid in Figure 5.6). While HPs of SMAC3 (and ASHA) could potentially be adjusted per-task to achieve better results, this would increase the computational budget and serve as a disadvantage compared to IPBT that performs well without per-task HP adjustment.

5.4.3. Ablation studies

In this section, we perform ablation studies by varying components of IPBT. The results are shown in Figure 5.7 and will be discussed in the order presented there.

Firstly, replacing our data-driven mechanism of stagnation detection with the procedure used in BG-PBT reduces IQM, demonstrating the benefit of a more flexible mechanism.

We then investigate components related to the reuse of weight information. Since shrink-perturb is a middle ground between copying weights exactly (which corresponds

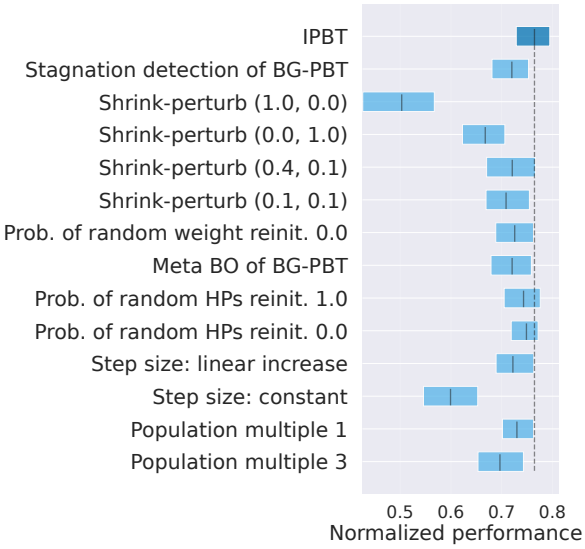


Figure 5.7: Ablation studies of IPBT: normalized performance (IQM and CI) across 4 tasks.

to parameters $(1.0, 0.0)$) and reinitializing them randomly (which corresponds to $(0.0, 1.0)$), we try both of these options and find that they worsen performance substantially. Next, we vary λ_{shrink} from its default of 0.2 to 0.4 and 0.1, observing a reduction in the IQM that highlights the importance of this HP. Another component related to weight reuse is random reinitialization of weights of half the models on a restart. Disabling it moderately reduces performance, highlighting the importance of injecting randomness to escape local minima. In Appendix 5.F, we additionally replace shrink-perturb with the distillation procedure of BG-PBT on reinforcement learning tasks, and find performance to be similar (which means that performance is not sacrificed when the task-agnostic shrink-perturb is used).

Investigating the HP information reuse, we find that replacing our HP reinitialization strategy based on time-varying BO with either BG-PBT's global BO strategy or random sampling lowers the IQM. However, if no random sampling is done at all, performance decreases, albeit marginally. We conclude that some noise injection improves overall performance.

In IPBT, step size is increased exponentially (doubled) on each restart. Increasing it linearly is worse, while keeping it constant is the worst by a substantial margin. This shows that adjusting the step size is essential to the success of IPBT.

Finally, we investigated whether restarting with twice the population size (and retaining the best half after the first step) is beneficial compared to not doing so ("Population multiple 1") or starting with thrice the size. Our results indicate that the default value of 2 strikes a good balance between discarding models with poor initial performance and avoiding excessive bias toward models with good performance after a single outer step.

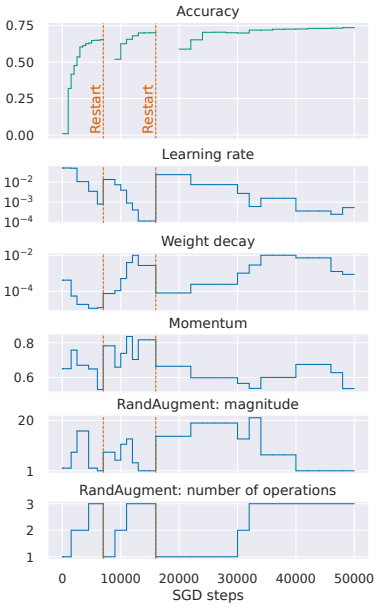


Figure 5.8: Accuracy and HP schedule of the best solution from an IPBT run on CIFAR-100.

5.4.4. Qualitatively inspecting a discovered schedule

IPBT is designed to naturally discover schedules with restarts. It can find such schedules not only for the learning rate (where restarts are well-established) but also other hyperparameters, as can be seen in Figure 5.8. For example, the number of Rand-Augment[1] operations is reset to 1 on each restart (and then increases within the iteration), while the initial magnitude of the augmentations increases across restarts (and appears to first increase and then decrease during the iteration). This serves as a clear example of how within- and across-iteration optimization both contribute to dynamically optimizing HPs.

5.5. Discussion and Conclusion

In this paper, we have introduced IPBT, a powerful HPO algorithm that demonstrates superior performance across 8 image classification and reinforcement learning tasks. By automatically adjusting its step size, IPBT substantially outperforms previous PBT variants with untuned step sizes and matches or exceeds their performance when their step sizes are tuned (while avoiding the cost of tuning). Moreover, IPBT significantly surpasses other popular HPO algorithms, such as RS, ASHA, and SMAC3.

IPBT achieves good results thanks to its iterative nature. Restarts allow escape from local optima in both weight and HP space, while efficiency is retained through information reuse across restarts: weights are adapted via shrink-perturb, and the initial HPs of each

[1] RandAugment (Cubuk et al., 2020) is an image augmentation procedure that applies several random augmentations of a specific magnitude (e.g., for rotation, different magnitudes correspond to different maximum angles by which an image can be rotated).

restart are sampled using a time-varying BO procedure. The decision when (and whether) to trigger a restart is made by our data-driven mechanism, allowing for flexibility based on performance dynamics.

Our ablation studies empirically demonstrated the benefit of the components of IPBT. We have generally observed that injecting some noise both into the weights and into HPs is beneficial in comparison to injecting no noise or relying on random exploration entirely. The component that seems most interesting for further investigation in future work is shrink-perturb (the mechanism for weight reuse), both in terms of how exactly its parameters influence HPO, and whether it can be productively replaced by a different method for partially resetting the weights.

Other promising directions for future work (and current limitations) include adaptation to multi-objective optimization (Dushatskiy et al., 2023) and NAS (Franke et al., 2021).

We designed IPBT to work well for black-box problems and small computation budgets (equivalent to 8 training runs). Algorithms that incorporate prior information about the problem are likely to outperform IPBT if such information is available (Mallik et al., 2023). Similarly, larger computational budgets are likely to enable state-of-the-art BO algorithms (Cowen-Rivers et al., 2022; R. T.-Y. Yu et al., 2025) to achieve better results.

Nonetheless, enabling efficient search for HP schedules within the wall-clock time of a single training run remains a strong advantage of PBT variants in general and IPBT in particular. Strong out-of-the-box performance of IPBT further marks it as a PBT variant of choice for efficiently optimizing HPs of deep learning tasks in low-budget settings.

Appendix

5.A. Pseudocode of IPBT

Algorithm 5.1 IPBT

Input: population size N, maximum number of inner steps step_{max}, step size $\text{step}_{\text{inner}}$, selection parameter λ

pop $\leftarrow$ {a pair of random HPs $\boldsymbol{h}^i$ and weights $\theta^i\}_{i=0}^{N-1}$
D $\leftarrow$ {} # Dataset of performance differences
$\text{D}_{\text{it}} \leftarrow$ {} # Dataset of performance differences within the current iteration (i.e., before a restart)
step $\leftarrow 0$
while step $<$ step_{max} **do**
  step $\leftarrow$ step + $\text{step}_{\text{inner}}$
  for $i \leftarrow 0$ **to** $N-1$ **do** # in parallel
    Train θ^i for $\text{step}_{\text{inner}}$ steps using $\boldsymbol{h}^i$
    $\text{perf}_{\text{i}}^{\text{step}} \leftarrow \texttt{evaluate}(\theta^i)$
    $\text{info}_{\text{i}} \leftarrow \left(\text{perf}_{\text{i}}^{\text{step}} - \text{perf}_{\text{i}}^{\text{step}-\text{step}_{\text{inner}}}, \text{step}, \boldsymbol{h}^i\right)$ # store difference to the previous performance and the HPs
    $\text{D}_{\text{it}} \leftarrow \text{D}_{\text{it}} \cup \text{info}_{\text{i}}$
    $\text{D} \leftarrow \text{D} \cup \text{info}_{\text{i}}$
  Check if a restart is triggered # Section 3.2
  if no restart **then**
    Sort pop by perf
    Replace the worst λ% with the copies of the best λ% (selected randomly)
    Fit a surrogate on D_{it}, use BO to set HPs of the copies
  else
    Reinitialize weights # Section 3.3
    Reinitialize HPs using BO on D # Section 3.3
    $\text{step}_{\text{inner}} \leftarrow 2 \cdot \text{step}_{\text{inner}}$ # Section 3.4
    $\text{D}_{\text{it}} \leftarrow$ {}

The pseudocode of IPBT is listed in Algorithm 5.1, see the main text and the source code for further details.

5.B. Tasks, search spaces, and evaluation

Our experiments are run on the tasks previously used for evaluating PBT variants (Chebykin et al., 2025b). In image classification tasks, hyperparameters of a ResNet-12 (He et al., 2016) are optimized on CIFAR-10/100 (Krizhevsky & Hinton, 2009), and Fashion-MNIST (Xiao et al., 2017) datasets for 50,000 Stochastic Gradient Descent (SGD) steps; and hyperparameters of a ConvNeXt-T (Z. Liu et al., 2022) are optimized on Tiny ImageNet (Stanford CS231N, 2017) for 150,000 steps. In reinforcement learning tasks, hyperparameters of a proximal policy optimization (Schulman et al., 2017) agent

are optimized on tasks implemented in Brax (Freeman et al., 2021) (Humanoid, Hopper, Pusher, Walker) during 150,000,000 environment interactions.

For PBT variants, the *large* search spaces of (Chebykin et al., 2025b) are used for both classification (Table 5.1) and reinforcement learning (Table 5.2) tasks. For the non-PBT baselines, the search spaces are extended with hyperparameters of a cosine learning rate schedule with restarts (Table 5.3).

Hyperparameter	Type	Exponent base	Range
Learning rate	real	10	[-6, 0]
Number of augmentations	integer	✗	[1, 4]
Augmentation strength	integer	✗	[1, 30]
Weight decay	real	10	[-8, -2]
Momentum	real	✗	[0.5, 0.999]

Table 5.1: Search space for the classification tasks (the *large* search space from (Chebykin et al., 2025b)).

Hyperparameter	Type	Exponent base	Range
Learning rate	real	10	[-4, -3]
Entropy coefficient	real	10	[-6, -1]
Batch size	integer	2	[8, 10]
Discount factor	real	✗	[0.9, 0.9999]
Unroll length	integer	✗	[5, 15]
Reward scaling	real	✗	[0.05, 20]
Number of updates per epoch	integer	✗	[2, 16]
GAE parameter	real	✗	[0.9, 1]
Clipping parameter	real	✗	[0.1, 0.4]

Table 5.2: Search space for the reinforcement learning tasks (the *large* search space from (Chebykin et al., 2025b)).

We did not change evaluation metrics, using accuracy for image classification, and task score for reinforcement learning tasks.

All performance claims in the paper are based on the test performances of the best models of each run of an algorithm. For algorithms with restarts (IPBT, BG-PBT, non-PBT baselines), the best model is chosen based on validation performance achieved either before a restart or at the end of training, while for the PBT variants without restarts, the best model is chosen based on the validation performance at the end of training.

5.C. Statistical testing

In the main text, in order to compare algorithms while taking performance of all tasks into account, the IQM of the normalized performance across tasks and seeds is estimated

Hyperparameter	Type	Exponent base	Range
Number of iterations before the first restart	integer	✗	[5, 50]
Multiplier for the number of iterations between restarts	integer	✗	[1, 2]
Minimum learning rate	real	10	[-8, -6]

Table 5.3: Additional hyperparameters of a cosine learning rate schedule with restarts used by non-PBT baselines. Note that the non-PBT baselines are run with an iteration equal to 1% of the budget, thus the number of iterations before the first restart is between 5% and 50% of the budget.

Table 5.4: P-values (rounded to 5 decimal places) of pair-wise statistical tests of IPBT against baselines (sorted by p-value).

Algorithm	Rejected H_0	p	p (Holm)
FIRE-PBT	Yes	0.00002	0.00016
SMAC3	Yes	0.00002	0.00016
Random search	Yes	0.00004	0.00024
ASHA	Yes	0.00012	0.00060
PB2	Yes	0.00022	0.00088
PB2-Mix	Yes	0.00810	0.02430
PBT	Yes	0.02070	0.04140
BG-PBT	No	0.49701	0.49701

via stratified bootstrapping (Agarwal et al., 2021). To assess statistical significance of differences between the IQMs of two algorithms, a paired stratified bootstrap test (Efron & Tibshirani, 1993) is executed with 50,000 replicates.

In every replicate, we sample seeds with replacement, use those seeds as indices for both algorithms across all tasks (thus getting a paired sample), and compute the difference between the IQMs. The replicate differences are then centered by the observed IQM, and a two-sided p-value is obtained from the (corrected) proportion of centered replicates whose magnitude exceeds the observed difference (Davison & Hinkley, 1997). The code for this procedure is included in our source code release.

IPBT is pairwise compared with 5 tuned PBT variants and 3 baselines (RS, ASHA, SMAC3) — 8 tests in total. The target significance level is 0.05, and the Holm correction (Holm, 1979) is used to control the family-wise error rate. The results in Table 5.4 show that performance of IPBT is significantly different from that of all algorithms except for BG-PBT.

5.D. Hyperparameters

The hyperparameters of all the algorithms are listed in the configuration files included in the source code release. In this section, we discuss the setting of hyperparameters of novel components of IPBT.

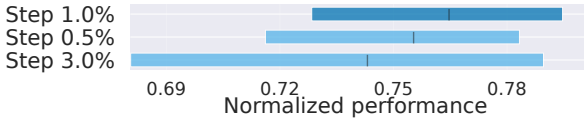


Figure 5.9: Extra ablation study on the initial step size of IPBT: normalized performance (IQM and CI) across 4 tasks (same setup as the ablation study in the main text).

Shrink-perturb The parameter values used in the main experiments are $(0.2, 0.1)$. The results when using $(0.4, 0.1)$ and $(0.1, 0.1)$ are reported in the ablation study in the main text. Additionally, the setting $(0.5, 0.5)$ was tried during early development (on the CIFAR-10 task) and found to perform worse.

Stagnation detection The parameters for restart detection used in the experiments were set to $t_{\text{patience}} = 3$ and $t_{\text{interval}} = 15$. These values were chosen based on visual inspection of performance traces during preliminary experiments. While t_{interval} was not varied, values of 2, 3, and 4 were tested for t_{patience} using an early version of the algorithm on CIFAR-10/100 and Brax Humanoid tasks. The setting $t_{\text{patience}} = 3$ yielded the highest IQM.

Initial step size In the experiments, the initial step size is set to 1% of the budget. We additionally experiment with settings of 0.5% and 3%, results are shown in an ablation study in Figure 5.9 (not included in the main text due to space constraints). It can be observed that reducing the step size to 0.5% leads to only a slight decrease in IQM, as IPBT can increase the step size when needed. Starting with a larger step size of 3% reduces performance more, as IPBT is not designed to decrease the step size and larger steps lead to faster budget exhaustion.

Strategy of step size increase The step size is doubled on each restart. We additionally tried increasing it linearly, with the results reported in the ablation study in the main text.

All preliminary experiments were run with seeds distinct from those used for the ablation studies and the main experiments.

5.E. Implementation details

Hardware and software The configuration of the servers used to run experiments: 3 Nvidia A5000 GPUs, 2 Intel(R) Xeon(R) Bronze 3206R CPUs, and 96 GB of RAM. The OS is Fedora Linux 36. The random seeds and the versions of all used frameworks and libraries are specified in the configuration files included in the source code release.

SMAC3 Directly using the latest version of SMAC3 (2.3.1) and setting the total budget in line with the documentation caused the algorithm to stop after only $\approx 75\%$ of the budget was used. We believe this issue stems from a discrepancy between the theoretically calculated budget usage, and the practical asynchronous implementation. To address this, we forked the code and introduced an alternative stopping criterion that monitors

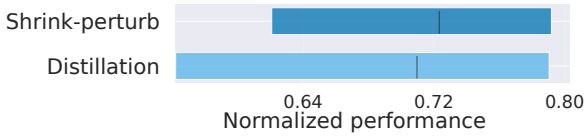


Figure 5.10: Extra ablation study on the weight reuse mechanism of IPBT: normalized performance (IQM and CI) across the Humanoid and Hopper tasks.

the actual budget consumed. This resulted in nearly full budget usage and improved performance on CIFAR-10 in preliminary experiments.

BG-PBT As mentioned in Section 3.3, Wan et al. (2022) use either 26.6% or 40% of the budget in their second stopping criterion (depending on the task). For uniformity, we use 30% for all tasks. We enable this criterion only with the step size of 1% (that is the closest to the original setting of BG-PBT) because otherwise too few steps are done before a forced restart is triggered. We also enable distillation only in this setting. Additionally, when the step size is 33.3%, we change $N_{\text{multiplier}}$ from the default 3 to 1 because otherwise the entire budget is spent on random initialization.

5.F. Replacing shrink-perturb with distillation

In BG-PBT, a distillation procedure is used to transfer information stored in weights across restarts. We replace shrink-perturb in IPBT with this procedure, and evaluate it on the Humanoid and Hopper tasks (as the distillation is specific to reinforcement learning tasks). Figure 5.10 shows that the performance is approximately the same, meaning that task-agnostic shrink-perturb matches task-specific distillation.

6

Hyperparameter-Free Medical Image Synthesis for Sharing Data and Improving Site-Specific Segmentation

Sharing synthetic medical images is a promising alternative to sharing real images that can improve patient privacy and data security. To get good results, existing methods for medical image synthesis must be manually adjusted when they are applied to unseen data. To remove this manual burden, we introduce a **Hy**perparameter-**Free** distributed learning method for automatic medical image **S**ynthesis, **S**haring, and **S**egmentation called **HyFree-S3**. For three diverse segmentation settings (pelvic MRIs, lung X-rays, polyp photos), the use of HyFree-S3 results in improved performance over training only with site-specific data (in the majority of cases). The hyperparameter-free nature of the method should make data synthesis and sharing easier, potentially leading to an increase in the quantity of available data and consequently the quality of the models trained that may ultimately be applied in the clinic.

The contents of this chapter are based on the following publication: Alexander Chebykin, Peter A. N. Bosman & Tanja Alderliesten. 2024. Hyperparameter-Free Medical Image Synthesis for Sharing Data and Improving Site-Specific Segmentation. In *Medical Imaging with Deep Learning, 3-5 July 2024, Paris, France,* 250: 199–219. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v250/chebykin24a.html.

6.1. Introduction

Deep learning methods can be beneficial for medical applications, but often suffer from limited data availability (Bowles et al., 2018). Generating and sharing synthetic datasets was suggested as a viable solution (Dube & Gallagher, 2014).

Many approaches can synthesize medical images (Bowles et al., 2018; Yi et al., 2019), fewer jointly produce segmentation maps (Guibas et al., 2018; Han et al., 2023) (which would be required for training downstream segmentation models) and, to the best of our knowledge, not a single method exists that could be applied to a new task without manually adjusting hyperparameters of training or data preprocessing.

The benefits of a hyperparameter-free adaptive method are self-evident. Such a method was realized for segmentation tasks by nnU-Net (Isensee et al., 2021), a method of automatically adjusting architecture and hyperparameters of a U-Net (Ronneberger et al., 2015) that showed excellent and robust performance in tens of challenges (Isensee et al., 2021).

Automatic adaptation of a high-quality underlying model and training pipeline is a general idea that could be extended to other tasks, such as medical image synthesis. A Generative Adversarial Network (GAN) called StyleGAN2 (Karras et al., 2020) demonstrated good results in medical image synthesis (Woodland et al., 2022). However, it currently does not automatically adapt to the image dimensions or the dataset size.

In this chapter, we introduce an automatically adjustable StyleGAN2 setup and integrate it with nnU-Net to create a **Hy**perparameter-**Free** medical image **S**ynthesis, **S**haring, and **S**egmentation method called **HyFree-S3**. We construct it as a distributed learning method where each site (e.g., a hospital) can automatically and asynchronously create a synthetic dataset and share it. A segmentation model can be automatically trained on the merged synthetic data and distributed back to the sites to be further automatically fine-tuned for improved performance on local data (see Figure 6.1).

This approach to distributed learning has practical advantages of requiring minimal coordination between sites and of reduced privacy risk thanks to not sharing real data, models trained with it, or their gradients (which is a potential source of data leakage in federated learning (Zhu et al., 2019)). An important concern is whether synthetic data includes memorized real data, which we address with a quantitative and qualitative investigation, as well as a technique for ensuring that synthetic data is not too similar to the real data.

We evaluate our method in three segmentation settings (pelvic MRIs, lung X-rays, polyp photos) to test its generality, the impact of synthetic data sharing, and the difference in performance compared to the realistic baseline of using only local data and the strong baseline of having central access to all the real data.

In this chapter, we only consider 2D models: compared to 3D models, they have lower computational requirements and need less data to be trained (which is important for the data sharing setting where some sites could have small datasets). In the future, our approach could be extended to 3D models for improved segmentation performance in settings where there is enough data and computational resources.

The contributions of this work are as follows:

- We propose HyFree-S3, a hyperparameter-free distributed learning method integrating image synthesis, data sharing, and segmentation.

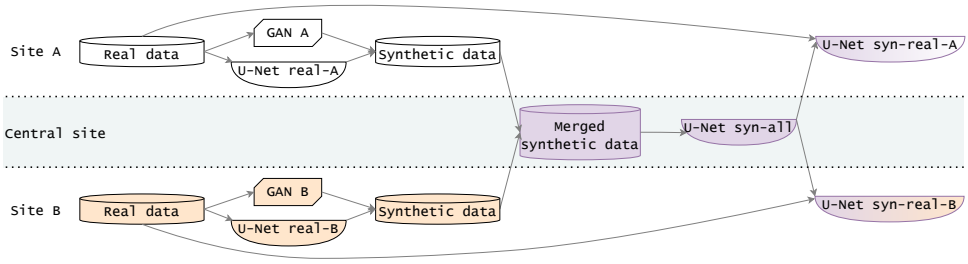


Figure 6.1: Overview of HyFree-S3 for two sites. Synthetic datasets are generated at each site independently, merged at a central site, and used in training a general segmentation model. That model is copied to all sites and independently fine-tuned on the local data. All models automatically adapt to the properties of the data.

- Towards that goal, we introduce a hyperparameter-free StyleGAN2 setup that can adapt to various image dimensions and dataset sizes.
- The segmentation quality of HyFree-S3 is evaluated in three settings. Furthermore, its scaling behavior and ability to avoid data memorization is investigated.

6.2. Related work

6.2.1. Sharing synthetic medical image data

Synthesizing medical data for sharing purposes so as to avoid privacy issues is an established idea in the literature (Goncalves et al., 2020). Medical *image* data can be synthesized by GANs (Bowles et al., 2018; Yi et al., 2019; L. Sun et al., 2022), but for supervised learning to be possible, an annotation needs to be associated with a synthetic image. While conditioning GANs on an image class is straightforward when generating data for classification tasks (X. Hu et al., 2018), conditioning on segmentation maps (Chang et al., 2023) to generate data for segmentation tasks requires the additional complexity of sharing segmentations to condition on (e.g., by training a separate GAN for them (Guibas et al., 2018; Han et al., 2023)). Thambawita et al. (2022) propose a GAN for joint unconditional generation of polyp images and segmentations that improves performance but requires a separate GAN to be trained for each input image, likely impeding scaling to larger datasets.

6.2.2. Distributed learning with medical image data

Federated learning (Konečný et al., 2016) can be applied to medical imaging data (Rieke et al., 2020; Adnan et al., 2022) where it allows the global model to be trained on diverse data from multiple hospitals (Ng et al., 2021) without sharing local data.

Federated learning algorithms for training a GAN (Rasouli et al., 2020) can be applied to medical data (Chang et al., 2023) but they incur privacy costs because real data could be reconstructed from the gradients passed between sites (Zhu et al., 2019). A common solution (Chang et al., 2020; Chang et al., 2023; J. Wang et al., 2023) is to train only the generator globally while the discriminators are trained per-site using the local data and the synthetic data from the global generator.

Our method differs from the existing distributed learning techniques in three ways. Firstly, it is asynchronous and does not require simultaneous online access to sites. Secondly, the generated data is filtered to not contain memorized data (in contrast to sharing a black-box generative neural network with potentially undiscovered vulnerabilities). Finally, HyFree-S3 is adaptable and hyperparameter-free, thus making it potentially easier to use.

6.3. Method

6.3.1. Overview of the method

We assume that N sites (such as medical centers) have a goal of solving a segmentation task. Each site has a local dataset that cannot be straightforwardly shared due to privacy or security concerns. The datasets may differ in sizes and image characteristics per-site.

Figure 6.1 shows the flow of data and models in HyFree-S3. Firstly, each site runs hyperparameter-free methods to create a generative model and a segmentation model. A generative model (Section 6.3.3) is used to create data without segmentations, which are then segmented by the segmentation model (Section 6.3.2) to create a complete synthetic dataset for sharing. The reasoning behind using two separate models is given in Section 6.3.4.

Next, the synthetic datasets from all sites are merged in a central location, and a general segmentation model is trained using all the synthetic data. This general segmentation model is transferred back to the sites, and is further automatically fine-tuned at each one. The resulting models benefit from the general pretraining but are specialized for each site.

6.3.2. Hyperparameter-free medical image segmentation

nnU-Net is a robust medical image segmentation method that was shown to perform excellently in a wide variety of competitions and benchmarks (Isensee et al., 2021). nnU-Net can adapt to diverse datasets without hyperparameter tuning thanks to heuristics for adjusting the underlying U-Net architecture (Ronneberger et al., 2015) and the training procedure. The spatial dimensions of the data influence the input size of the model, depths of the encoder (decoder), downsampling (upsampling) strides, and convolution kernel sizes.

To adjust the hyperparameters to the fine-tuning setting not considered by Isensee et al. (2021), we add linear learning rate warm-up for the first 10% of training epochs (Mosbach et al., 2021) and do not otherwise change the default hyperparameters.

6.3.3. Hyperparameter-free data synthesis

StyleGAN2 (Karras et al., 2020) is a powerful generative model that by default generates square images of a resolution of a power of two. However, medical images come in a variety of resolutions. Ideally, synthetic images of the appropriate resolution should be generated.

For segmentation tasks, nnU-Net adapts the structure of the U-Net to the resolution. We noticed a similarity between the structures of the generator/discriminator of a GAN and those of the decoder/encoder of a U-Net. Both the GAN generator and the U-Net decoder gradually upscale a low-resolution many-channel latent representation of an

image towards a high-resolution few-channel output. The architectures need to strike the correct balance between the speeds of increasing the resolution and decreasing the number of channels. As such, a network that strikes the correct balance in one setting, seems likely to do so in the other. Analogously, the discriminator and the encoder gradually downscale a high-resolution few-channel input towards a low-resolution many-channel representation.

Therefore, it appears natural to reuse the hyperparameters of the encoder and the decoder automatically determined by nnU-Net (depths of the networks, convolution strides, and kernel sizes) for the discriminator and the generator of a GAN. This will allow it to create non-square non-power-of-two-sized images of the exact size determined by nnU-Net.

Next, the number of training steps, n_{steps}, needs to be set automatically. In StyleGAN2, n_{steps} is defined in thousands of real images processed during training. As the dataset size increases, so should n_{steps} (to allow the GAN to learn from a larger amount of data). Setting n_{steps} to the number of images in the dataset ensures good image quality across different dataset scales, keeps training times short, and prevents memorization (see Section 6.4.3).

The number of images to be generated, n_{gen}, also needs to be determined. While generating images with GANs requires little compute and time, generating increasingly large numbers of samples leads to diminishing returns (Ravuri & Vinyals, 2019). We also need to consider the proportions of synthetic data coming from different sites used in training the general segmentation model: generative models trained on larger datasets should contribute more than those trained on smaller datasets, as they likely have higher quality. For these reasons, n_{gen} is set to ten times the dataset size for each dataset.

We use the augmentations setup of Zhao et al. (2020) that was shown to improve performance and help avoid overfitting with datasets as small as 100 images.

6.3.4. Why not synthesize images and segmentations jointly?

Segmenting with a separate model was a deliberate design choice and constraint. While it is possible to train a generative model to output segmentations as well as images, this would lead to segmentations influencing the images. This is undesirable for medical data sharing, as the references in many segmentation scenarios vary due to the protocol and observer variation. Letting these variations influence the generated *images* should be avoided: then the images themselves can still contribute to the improvement of the models (e.g., during unsupervised pretraining) even if segmentations need to be discarded or redone. Additionally, if no annotations are available, HyFree-S3 can still be utilized to share images (unlike methods that condition on segmentations).

6.3.5. Measuring memorization and preventing real data leakage

Synthetic data should be similar enough to the real data for the models trained on one to transfer to the other, and dissimilar enough for the privacy concerns to be alleviated. Generative models are capable of memorizing their training data and outputting it as "synthetic" samples (Feng et al., 2021). Memorization is difficult to determine because the reproduction typically includes some variation or noise. It is not obvious where the threshold between the presence and the absence of memorization should be.

We propose automatically determining this threshold based on the real data itself as follows: firstly, the patients are randomly split into two subsets. For each image in the first subset, its dissimilarity with each image in the second subset is computed using the L2 distance between their OpenCLIP embeddings (Ilharco et al., 2021). The minimal dissimilarity (i.e., the distance to the nearest neighbour) is stored for each image. The threshold is then defined as the p-th percentile of these dissimilarities.

After the synthetic dataset is generated, the dissimilarity of each synthetic image to all real images is similarly computed. If the distance of a synthetic image to its nearest real-image neighbor is below the determined threshold, the image is declared to be memorized and is discarded. For $p = 0$ (the minimum of dissimilarities), this procedure ensures that any synthetic image is only as similar to a real image as one unrelated real image to another; we set $p = 5$ to guard against outliers. The procedure relies on the quality of the embedding model that can only be demonstrated empirically (Cherti et al., 2023).

6.4. Experiments

6.4.1. Experiment setup

In our experiments, we emulate a distributed setting with N sites. As per Section 6.3.1, for each site S_i, GAN-S_i and U-Net-*real*-S_i are trained and used to generate a synthetic dataset. U-Net-*syn-all* is trained on the merged datasets and fine-tuned at each site to get U-Net-*syn-real*-S_i. U-Net-*real-all* is trained on merged real data as a baseline. All the experiments are performed via five-fold cross-validation (3 folds for training, 1 fold for validation and test each). The mean and the standard deviation of the Dice Score (DS) and the 95th percentile of the Hausdorff Distance (HD95) across the folds are reported. All the evaluations were performed on real data. The results of statistical testing are given in Appendix 6.A. Appendix 6.D contains comparisons with federated learning baselines. The ablations of pretraining with single-site synthetic data and of using the standard StyleGAN2 architecture are reported in Appendices 6.F, 6.G.

6.4.2. Datasets

Cervix: a private dataset from the Leiden Univercity Medical Center consisting of T2-weighted MRI scans of 185 cervical cancer patients who underwent brachytherapy, with 4 organs-at-risk (bladder, bowel, rectum, sigmoid) delineated. The dataset was split into two sites based on the scanner to emulate a non-i.i.d. data distribution: site A (Philips Ingenia 1.5T (128 patients)), and site B (Philips Intera (36 patients), Ingenia 3T (13), Achieva (8)). The median resolution is $37 \times 432 \times 432$ voxels, the median spacing is $4 \times 0.53 \times 0.53$ mm.
Lung: QaTa-COV19 (Degerli et al., 2022), a dataset of COVID-19 chest X-ray images and binary segmentation masks of pneumonia. We use 6,307 images for which an anonymized patient ID is provided (2,130 patients) and randomly split patients into 2 or 8 sites. The median resolution is 224×224 pixels.
Polyp: polyp photos with binary segmentation masks of polyps, site A contains data from HyperKvasir (Borgli et al., 2020) (1000 images, median resolution 530×621 pixels), site B contains data from CVC-ClinicDB (Bernal et al., 2015) (612 images, 384×288 pixels).

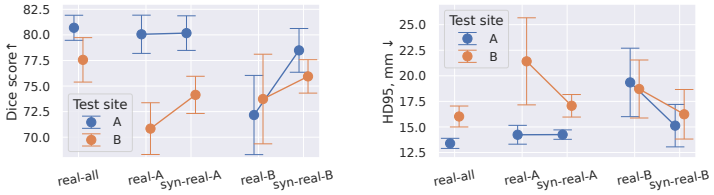


Figure 6.2: Results for the **Cervix** data: U-Nets trained with the settings specified in Section 6.4.1 and evaluated on sites A and B (5 folds).

6.4.3. Results

Synthetic data sharing leads to improved segmentation quality

Figure 6.2 shows that in the experiments with **Cervix**, DS and HD95 metrics improve in most cases. The performance on site A is approximately the same across the *real-A*, *syn-real-A*, *real-all* settings, showing that adding data from site B is not very helpful even if it is real data. This is likely due to the large number and uniformity of patients in A itself. Nonetheless, the performance in the *syn-real-A* setting on site B is improved compared to *real-A* (by 3.3 DS and 4.3 HD95 on average), showing that pretraining on merged synthetic data improves the robustness of the model to data shifts. For site B, the pretrained and fine-tuned model (*syn-real-B*) outperforms its counterpart trained exclusively on the local data (*real-B*) when tested on both A (by 6.3 DS and 4.3 HD95 on average) and B (by 2.2 DS and 2.5 HD95 on average). The full results for all datasets are given in Appendix 6.B.

It can be seen in Figure 6.3 that switching from the model trained only on local data (*real*) to the one pretrained on the merged synthetic data and fine-tuned on local data (*syn-real*) leads to improvements in the **Lung** setting of 0.8 DS on average. For **Polyp**, the improvement for the target site is minor (0.5 DS on average), but the improvement in robustness to data shifts, as measured by the performance on the other site, is large (on average, 2.7 DS for A and 13.4 DS for B).

While training on real data centrally (*real-all*) gives the best results overall, our method comes close (the largest difference is 2.0 DS in **Polyp**-B) without requiring real data sharing.

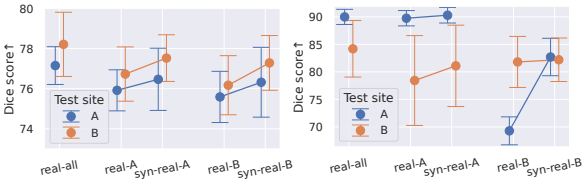


Figure 6.3: DS for the **Lung** (left) and **Polyp** (right) data: U-Nets trained in the settings specified in Section 6.4.1 and evaluated on sites A and B (5 folds).

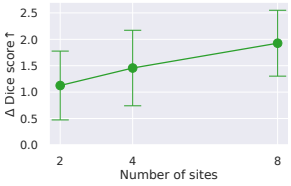


Figure 6.4: DS improvement of *syn-real* over *real* as more sites are added (**Lung**).

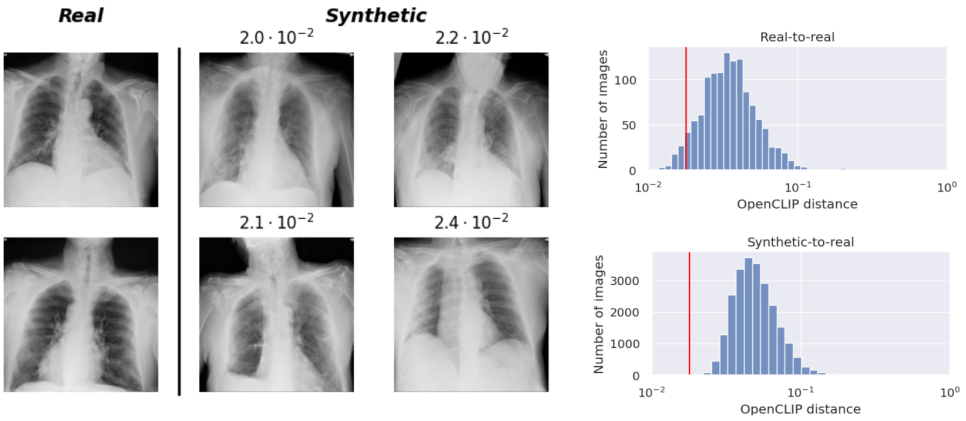


Figure 6.5: **Left:** real images that are the closest to any synthetic image and the two closest synthetic images (including the distance to the real image). **Right:** a distribution of distances to the nearest neighbor for *(top)* two subsets of real images or *(bottom)* synthetic and real images, and the 5^{th} percentile of the real-to-real distances.

6

Benefits of synthetic data sharing increase with more sites

The scaling behavior of HyFree-S3 is investigated in the **Lung** setting, with the data split into 8 sites. The general segmentation model is pretrained with the synthetic data from 2, 4, or 8 sites, and then fine-tuned at each site. The average difference in DS between *real* (training with local data only) and *syn-real*, shown in Figure 6.4, becomes larger as the number of site grows, showing that the method is scalable and enables larger improvements when more sites join.

Memorization is not observed

Per Section 6.3.5, we use OpenCLIP embeddings to compare either subsets of real images, or real images and synthetic images. As a sanity check, we established that an image present in two sets will be its own nearest neighbor, and that a mirrored image will most often be the nearest neighbor of the original image (in 97.9% of cases for **Lung**).

Figure 6.5 (top right) shows a histogram of distances to the nearest neighbor when comparing two subsets of real images (from a **Lung** experiment), as well as their 5^{th} percentile that will be used as a threshold to filter synthetic images. The histogram of distances between real and synthetic images in Figure 6.5 (bottom right) has a similar shape but shifted to the right. As a result, all synthetic images have a distance above the threshold, meaning that while synthetic images are generally similar to real images, they are not too similar.

We visualize adversarially chosen real and synthetic images in Figure 6.5 (left). We select two synthetic images (column 2) with the smallest nearest neighbor distances to some real images (column 1), and the synthetic images that have the second-smallest distances to these real images (column 3). The images are broadly similar but clearly distinct and do not demonstrate memorization (it should be noted that our memorization

analysis relies on OpenCLIP embeddings, see Appendix 6.E for further discussion of memorization).

For **Lung** and **Polyp**, only 6 synthetic images (out of 3×10^5) are discarded as too similar to real images. For **Cervix**, 1,180 images (out of 2.6×10^5) are discarded, some of which look very similar to their real nearest neighbours but none are exact duplicates, which is consistent with the nature of 3D data where nearby slices are similar. See Appendix 6.C for visual comparisons between real and synthetic images. Only 0.2% of synthetic images being discarded shows that our GANs typically do not memorize images but the proposed filtering scheme could nonetheless help to avoid sharing potentially memorized data.

6.5. Discussion and Conclusion

We have introduced HyFree-S3, a method for synthesizing medical images for sharing and segmentation that does not require adjusting hyperparameters, as it relies on our hyperparameter-free setup of StyleGAN2 for image generation and similarly hyperparameter-free nnU-Net (Isensee et al., 2021) for segmentation.

The method is asynchronous and does not expect different sites to share infrastructure or to be online simultaneously. HyFree-S3 was shown to come close to the performance of training with centralized real data and to improve upon training only with local data, while not requiring sharing real patient data. The synthetic dataset produced by HyFree-S3 could be reused for training future models without going back to the sites where the real data is stored. We additionally proposed a memorization evaluation technique for ensuring that synthetic data is sufficiently dissimilar from real data.

However, our method also has a number of limitations. Firstly, as models for data sharing are trained on each site independently, large enough datasets have to be present there (especially for training a GAN). Secondly, we generate images and annotations separately for reasons described in Section 6.3.4, which nonetheless means that the annotations produced by an independent model will not perfectly correspond to the images, introducing noise to synthetic data. Finally, our method as presented cannot be applied to all possible datasets, as it relies on nnU-Net for preprocessing and segmentation: nnU-Net can handle a variety of inputs but not, e.g., high-resolution histopathology images. However, extending nnU-Net with an appropriate data preprocessing procedure would make HyFree-S3 applicable too.

Despite some limitations, adaptable methods that do not require task-specific tuning are still valuable for the translation of deep learning models into the real world, where there may not be time or expertise for careful manual adaptation to each task.

Appendix

6.A. Statistical testing

Table 6.1 lists p-values for one-sided Wilcoxon pairwise rank tests (Wilcoxon, 1992) with Bonferroni correction (Dunn, 1961). For each dataset, we compare metrics of *syn-real* (pretraining with merged synthetic data followed by site-specific fine-tuning on real data) against that of *real* (training with local real data only) with the null hypothesis that *syn-real* performs worse. The performances of *syn-real-A* (evaluated on A, B) and *syn-real-B* (evaluated on A, B) across 5 folds are pulled together for increased sample size (and similarly for *real-A* and *real-B*). The total sample size is 20 for each test. P-values below the significance threshold of 0.0025 are highlighted (target p-value= 0.01, 4 tests, corrected $p = 0.0025$). In all cases the null hypothesis is rejected, thus we can conclude that *syn-real* performs statistically significantly better than *real*.

Table 6.1: P-values of conducted experiments (rounded to four decimal places). Values below the significance threshold of 0.0025 are highlighted.

Dataset	Metric	P-value
Cervix	DS	0.0006
Cervix	HD95	0.0007
Lung	DS	0.0002
Polyp	DS	0.0002

6.B. Full results

Tables 6.2, 6.3, 6.4 give full results for our **Cervix**, **Lung**, **Polyp** experiments. For **Cervix** and **Lung**, U-Net-syn-S_i are additionally trained on local synthetic datasets to compare with U-Net-syn-all. As expected, they perform worse.

Table 6.5 gives per-organ performance for the **Cervix** dataset. Figure 6.6 contains slice predictions for qualitative comparison of a *real* and a *syn-real* models.

Table 6.2: Results for the **Cervix** data. Each column corresponds to a U-Net trained in the specified setting (see Section 6.4.1) and evaluated on sites A and B. Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Training setting							
		real-all	syn-all	real A	syn A	syn-real A	real B	syn B	syn-real B
DS	A	80.7 ± 1.2	79.4 ± 1.3	80.1 ± 1.9	79.0 ± 1.5	80.2 ± 1.7	72.2 ± 3.9	72.4 ± 3.8	78.5 ± 2.1
	B	77.6 ± 2.2	75.4 ± 2.6	70.8 ± 2.5	69.9 ± 1.5	74.1 ± 1.8	73.7 ± 4.4	72.8 ± 3.9	75.9 ± 1.6
HD95	A	13.4 ± 0.5	14.8 ± 1.1	14.2 ± 0.9	15.3 ± 1.2	14.3 ± 0.5	19.4 ± 3.3	19.9 ± 3.1	15.1 ± 2.1
	B	16.0 ± 1.0	16.1 ± 2.7	21.4 ± 4.3	21.0 ± 3.9	17.1 ± 1.1	18.7 ± 2.8	19.0 ± 3.0	16.2 ± 2.4

Table 6.3: Results for the **Lung** data. Each column corresponds to a U-Net trained in the specified setting (see Section 6.4.1) and evaluated on sites A and B. Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Training setting							
		real-all	syn-all	real A	syn A	syn-real A	real B	syn B	syn-real B
DS	A	77.2 ± 0.9	75.8 ± 1.1	75.9 ± 1.0	74.7 ± 1.3	76.5 ± 1.6	75.6 ± 1.3	74.6 ± 1.5	76.3 ± 1.7
	B	78.2 ± 1.6	76.7 ± 1.3	76.7 ± 1.4	75.3 ± 1.7	77.5 ± 1.2	76.2 ± 1.5	75.0 ± 1.6	77.3 ± 1.4

Table 6.4: Results for the **Polyp** data. Each column corresponds to a U-Net trained in the specified setting (see Section 6.4.1) and evaluated on sites A and B. Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Training setting					
		real-all	syn-all	real A	syn-real A	real B	syn-real B
DS	A	90.0 ± 1.4	87.7 ± 1.4	89.7 ± 1.4	90.3 ± 1.4	69.3 ± 2.5	82.7 ± 3.4
	B	84.2 ± 5.1	80.9 ± 5.7	78.4 ± 8.2	81.1 ± 7.4	81.8 ± 4.6	82.2 ± 3.9

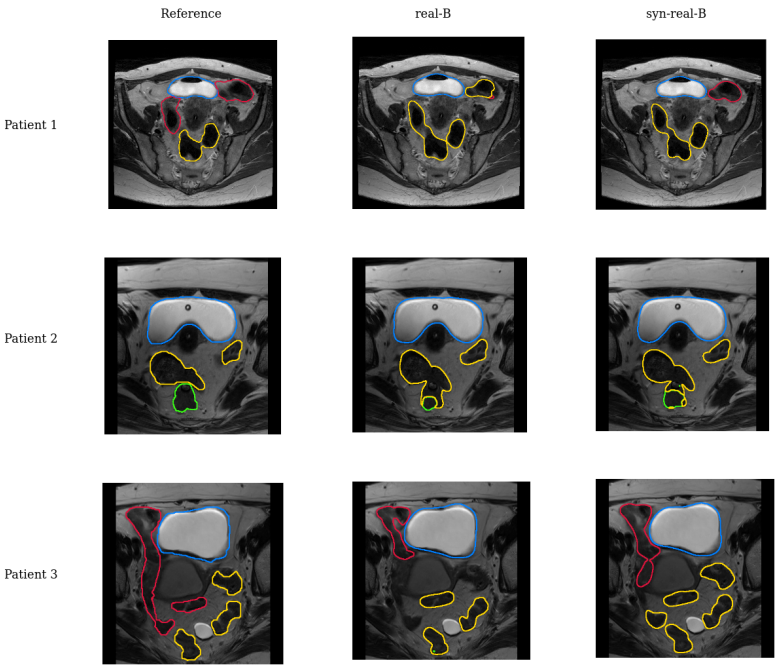


Figure 6.6: Comparison of *real-B* to *syn-real-B* using slices from 3 random patients. Depicted are bladder (blue), bowel (red), rectum (green), sigmoid (yellow).

Table 6.5: Per-organ metrics for the **Cervix** data. Each row corresponds to a U-Net trained in the specified setting (see Section 6.4.1) and evaluated on sites A and B. Mean ± st. dev. is reported for 5 folds.

Model	Avg.		Bladder		Bowel		Rectum		Sigmoid	
	A	B	A	B	A	B	A	B	A	B
Dice Score										
real-A	80.1 ±1.9	70.8 ±2.5	96.0 ±0.4	91.9 ±2.5	68.8 ±3.5	56.0 ±6.4	81.7 ±2.9	76.8 ±3.3	73.7 ±2.5	58.7 ±3.4
syn-A	79.0 ±1.5	69.9 ±1.5	96.0 ±0.4	91.7 ±2.5	66.6 ±2.8	53.2 ±4.9	81.7 ±2.8	76.5 ±2.8	71.7 ±2.0	58.2 ±2.0
syn-real-A	80.2 ±1.7	74.1 ±1.8	96.0 ±0.4	93.3 ±1.3	69.6 ±2.8	61.4 ±6.3	81.5 ±2.8	79.3 ±3.6	73.5 ±2.0	62.6 ±3.1
real-B	72.2 ±3.9	73.7 ±4.4	95.3 ±0.6	94.1 ±0.9	53.2 ±8.5	60.3 ±9.9	77.4 ±3.1	79.1 ±3.2	62.8 ±6.4	61.5 ±7.0
syn-B	72.4 ±3.8	72.8 ±3.9	95.4 ±0.5	93.9 ±1.2	53.4 ±8.9	57.7 ±11.0	77.5 ±2.8	78.8 ±3.5	63.3 ±6.5	60.8 ±3.8
syn-real-B	78.5 ±2.1	75.9 ±1.6	95.8 ±0.4	93.9 ±1.2	65.6 ±3.9	63.4 ±5.6	81.5 ±2.7	80.5 ±3.3	71.0 ±3.7	66.0 ±3.7
real-all	80.7 ±1.2	77.6 ±2.2	96.2 ±0.4	94.4 ±1.0	70.1 ±2.1	65.5 ±6.5	81.8 ±2.6	82.4 ±2.5	74.7 ±1.6	68.0 ±3.0
syn-all	79.4 ±1.3	75.4 ±2.6	96.1 ±0.3	94.0 ±1.2	67.5 ±2.2	61.9 ±7.9	82.0 ±2.8	81.0 ±3.2	72.1 ±2.0	64.9 ±3.8
Hausdorff Distance (95th percentile)										
real-A	14.2 ±0.9	21.4 ±4.3	3.6 ±1.0	11.7 ±8.4	20.3 ±1.8	25.5 ±2.7	13.5 ±2.5	16.4 ±4.9	19.6 ±3.3	32.1 ±7.8
syn-A	15.3 ±1.2	21.0 ±3.9	3.8 ±1.0	13.9 ±9.9	22.0 ±3.4	26.6 ±4.8	13.6 ±2.1	14.1 ±3.2	21.9 ±3.8	29.2 ±6.7
syn-real-A	14.3 ±0.5	17.1 ±1.1	3.7 ±1.3	7.9 ±4.3	18.7 ±3.6	20.6 ±2.4	13.8 ±2.3	12.7 ±2.3	20.9 ±2.2	27.1 ±5.0
real-B	19.4 ±3.3	18.7 ±2.8	3.9 ±0.5	5.8 ±2.2	28.7 ±6.6	24.6 ±6.1	17.0 ±3.3	15.2 ±2.5	27.8 ±5.6	29.3 ±4.7
syn-B	19.9 ±3.1	19.0 ±3.0	4.4 ±1.1	5.9 ±2.5	28.5 ±7.6	26.7 ±8.1	17.4 ±3.8	14.4 ±2.5	29.2 ±5.1	29.2 ±4.2
syn-real-B	15.1 ±2.1	16.2 ±2.4	4.1 ±0.8	6.1 ±2.8	22.8 ±5.8	21.2 ±3.3	13.5 ±1.7	13.3 ±2.4	20.2 ±5.0	24.4 ±6.8
real-all	13.4 ±0.5	16.0 ±1.0	2.9 ±0.4	5.4 ±2.1	18.8 ±3.5	20.9 ±3.2	13.4 ±2.0	12.3 ±2.6	18.5 ±3.0	25.5 ±3.1
syn-all	14.8 ±1.1	16.1 ±2.7	3.9 ±1.1	5.5 ±2.4	20.9 ±3.7	20.8 ±3.4	13.7 ±2.0	12.4 ±2.2	20.6 ±4.3	25.5 ±7.6

6.C. Visually checking memorization of synthetic images

In Figure 6.7, we provide examples of synthetic images that were too similar to real images and therefore discarded. Visually, the synthetic images do not appear to be exact copies of real images.

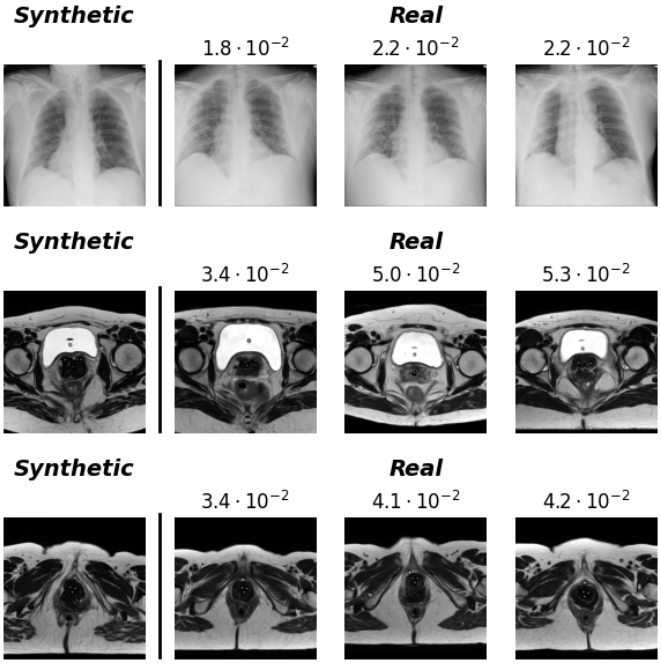


Figure 6.7: Examples of discarded synthetic images and three nearest-neighbor real images for each (annotated with distances to the synthetic image).

6.D. Federated learning baselines

To compare our approach to federated learning, we run two federated learning baselines: Federated Averaging (FedAvg) (McMahan et al., 2017) and Distributed Synthetic Learning (DSL) (Chang et al., 2023).

FedAvg trains a segmentation model on the real data at each location and periodically averages the weights to get a global model. We implement FedAvg atop nnU-Net for fair comparison to our approach. DSL is a state-of-the-art approach to training a GAN in a federated manner to be used for generating synthetic data and training a U-Net. Our DSL experiments are based on the official implementation of DSL, see our fork at https://github.com/AwesomeLemon/DSL_All_Code.

Since the key contribution of our method is its automatic and hyperparameter-free nature, we run the baselines in a similar setting of no hyperparameter tuning. For FedAvg, we used 1000 communication rounds with 1 epoch of local training in between since it was the best setting in (McMahan et al., 2017). For DSL, we followed the paper (Chang

et al., 2023) and the official code base. For hyperparameters that differed across the three datasets in DSL, we used the median values: λ_{L1}: 300, 150, 100 $\rightarrow$ 150; batch size: 6, 6, 3 $\rightarrow$ 6.

The results of the experiments are reported in Tables 6.6, 6.7, 6.8. FedAvg on average has 5.3 worse Dice Score (DS) than HyFree-S3 in the i.i.d. setting of **Lung**, with the gap widening further in the non-i.i.d settings: 10.7 DS for **Cervix**, 29.2 DS for **Polyp**. DSL performs substantially worse in all settings (on average: **Lung**: 32.8 DS worse, **Cervix**: 35.1 DS worse, **Polyp**: 36.9 DS worse). The poor performance of DSL was unexpected, given the excellent results in (Chang et al., 2023). We attribute this to untuned hyperparameters. We have checked that it learns to generate relatively realistic images (see Figure 6.8), on which the U-Net that it trains performs well, however it generalizes to real test images poorly.

We additionally did minimal manual hyperparameter tuning of DSL in the Polyp setting and were able to improve its performance by 9.2 DS on average ("DSL (lightly tuned)" in Table 6.8; we only changed the hyperparameters of the U-Net: we switched the optimizer to AdamW with default hyperparameters, switched step-wise learning rate schedule to cosine annealing, added random color jitter and random rotation augmentations).

While the performance of FedAvg and DSL could potentially be improved further via hyperparameter tuning, their default performance is subpar, highlighting the benefit of automatic methods such as HyFree-S3, the hyperparameter-free nature of which is its key benefit.

Table 6.6: Comparison to the federated learning baselines for the **Cervix** data. The results of HyFree-S3 for sites A/B correspond to syn-real A/B (see Section 6.4.1). Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Method			
		syn-real A	syn-real B	FedAvg	DSL
DS	A	80.2 ± 1.7	78.5 ± 2.1	65.0 ± 1.9	42.8 ± 10.4
	B	74.1 ± 1.8	75.9 ± 1.6	67.7 ± 2.3	41.2 ± 0.6

Table 6.7: Comparison to the federated learning baselines for the **Lung** data. The results of HyFree-S3 for sites A/B correspond to syn-real A/B (see Section 6.4.1). Mean ± st. dev. is reported for 5 folds.

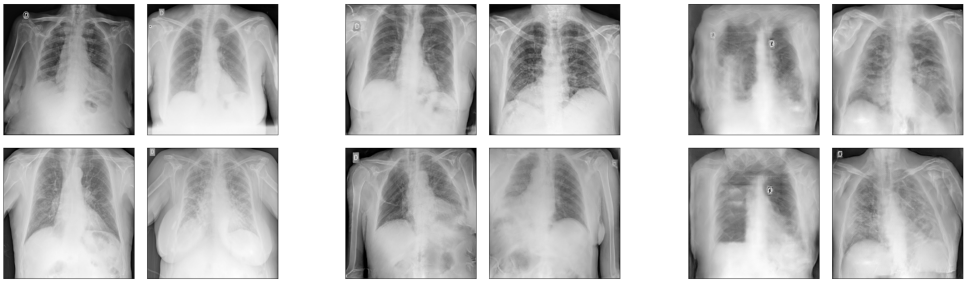
Metric	Test site	Method			
		syn-real A	syn-real B	FedAvg	DSL
DS	A	76.5 ± 1.6	76.3 ± 1.7	71.2 ± 2.5	43.8 ± 13.4
	B	77.5 ± 1.2	77.3 ± 1.4	71.9 ± 1.2	44.3 ± 15.4

6.E. Further discussion of memorization

We have been communicating with a data protection officer about the perspectives of synthetic data since this project started. Based on preliminary discussions, our method could greatly simplify data sharing between institutes and decrease privacy risks. However,

Table 6.8: Comparison to the federated learning baselines for the **Polyp** data. The results of HyFree-S3 for sites A/B correspond to syn-real A/B (see Section 6.4.1). Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Method				
		syn-real A	syn-real B	FedAvg	DSL	DSL (lightly tuned)
DS	A	90.3 ± 1.4	82.7 ± 3.4	54.3 ± 3.8	51.8 ± 1.1	60.7 ± 2.7
	B	81.1 ± 7.4	82.2 ± 3.9	55.2 ± 10.1	42.3 ± 12.1	51.9 ± 7.4



(a) Real (b) HyFree-S3 (c) DSL

Figure 6.8: Random sample of **Lung** images (Site A, fold 0).

currently, there are no official, internationally accepted, guidelines as to what constitutes memorization (of imaging data).

Therefore, it is difficult to say if our approach is "clinically acceptable", given lack of prior work on what that entails. While we are confident in our analysis, it could be made more robust by using several dissimilar embedding models, or by increasing the threshold. Still, our method relies on empirical performance of neural networks (as noted in Section 6.3.5), and therefore no mathematical guarantees can be given that no memorization occurs. Nonetheless, we believe that empirical evaluation similar to ours could be enough for clinical acceptance, once the guidelines are determined.

To further investigate the memorization phenomenon, we use an alternative method for finding memorized images from a concurrent work (Dar et al., 2024), where it was successfully used to find medical images memorized by a diffusion model. The method is similar to the approach we used in that it relies on embeddings of images via a neural net, with the two differences from our work being the model (the authors trained their own embedding model on the target dataset) and the similarity measure (the authors used correlation).

We apply this approach in the **Lung** setting. Whereas our method flagged close to zero samples as memorized, this approach flags $\approx 12\%$. However, visual inspection of the synthetic samples closest to some real ones (Figure 6.9) indicates that the synthetic samples are not duplicates of the real ones (based on our judgement). While similar, they differ in their details, and do not satisfy the properties on which the definition of memorization from Dar et al. (2024) is based: they are not variants of the original image derived via rotation, flipping, or contrast adjustment.

Nonetheless, memorization is difficult to define, and perhaps the eventual guidelines would enforce stricter definitions of memorization that would include these samples. If so, such synthetic outputs could be removed by using an appropriate embedding model within our method. HyFree-S3 is agnostic to the duplicates removal technique used within, better techniques can be substituted when they are developed.

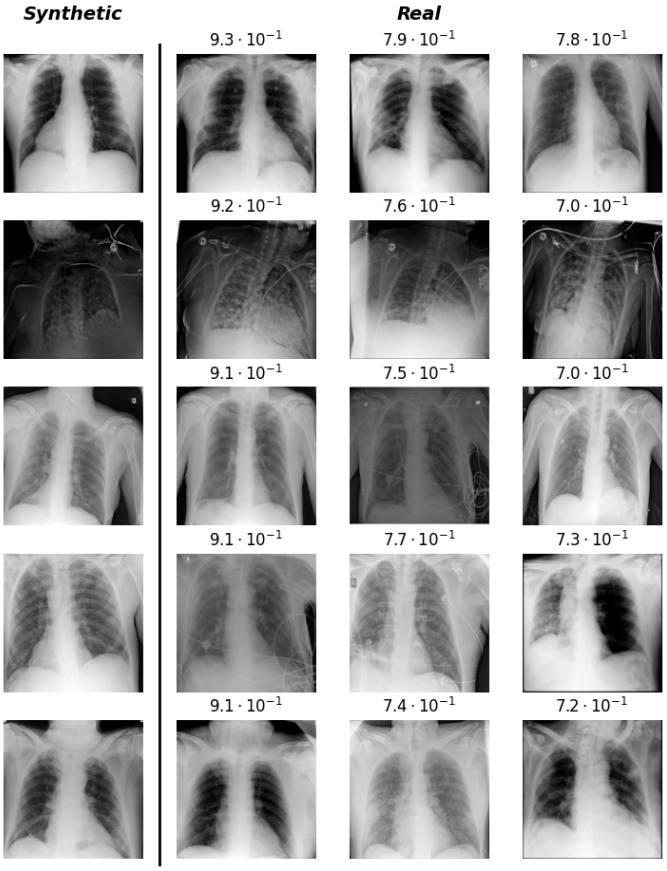


Figure 6.9: Alternative memorization detection: examples of synthetic images flagged as memorized and three nearest-neighbor real images for each (annotated with similarity to the synthetic image). The synthetic images with the highest similarity to real images are visualized.

6.F. Pretraining using single-site synthetic data

Does pretraining on multi-site synthetic data have additional benefit over pretraining on single-site synthetic data? Here we compare our default setting (generating $10 \times n_{\text{real}}$ images at each site and pulling them together) to the setting of generating $20 \times n_{\text{real}}$ images at one site. In both settings, a U-Net is pretrained with the synthetic images and

fine-tuned with the real local data. The experiments are performed for two sites with the **Cervix** data. Table 6.9 demonstrates that the networks pretrained with the local synthetic data (*syn-local-real*) achieve on average 2.7 DS worse performance than the networks pretrained on pooled synthetic data (*syn-real*). This result empirically demonstrates the benefit of bringing data from multiple sites together.

Table 6.9: Comparison of no pretraining (*real*) to pretraining on local synthetic data only (*syn-local-real*) or pooled synthetic data (*syn-real*) in the **Cervix** setting. Mean ± st. dev. is reported for 5 folds.

Metric	Test site	Training setting					
		real A	real B	syn-local-real A	syn-local-real B	syn-real A	syn-real B
DS	A	80.1 ± 1.9	72.2 ± 3.9	80.0 ± 1.8	72.9 ± 3.2	80.2 ± 1.7	78.5 ± 2.1
	B	70.8 ± 2.5	73.7 ± 4.4	71.6 ± 1.7	73.3 ± 3.8	74.1 ± 1.8	75.9 ± 1.6

6.G. Comparison to the standard StyleGAN2 architecture

To experimentally confirm that transferring the nnU-Net architectural parameters to StyleGAN2 is reasonable, we compare our GANs to the standard StyleGAN2 trained with square images of a resolution of a power of two. The experiment is performed with the Polyp-B data, for which the nnU-Net determined resolution (388×320) is the farthest from a square where the dimensions of the sides are a power of two (note that for such images our StyleGAN2 architecture would be exactly the same as the standard one). The images are resized to the closest power of two (256×256), and the standard StyleGAN2 is trained. Then synthetic images are generated and resized to 388×320. Afterwards, the Frechet Inception Distance (FID) (Heusel et al., 2017) is calculated between them and the real images. FID is an established metric of GAN quality, lower values are better.

We compare the FID of the StyleGAN2-256×256 to our StyleGAN2-388×320, and find the latter to achieve better results (102.4 ± 4.0 FID vs 88.0 ± 3.2 FID). This confirms that the architecture is reasonable, as it was able to make use of the increased resolution of its inputs to achieve higher generation quality.

7

Discussion

7.1. Answers to the research questions

Research Question: Chapter 2

Can multiple supernetworks be leveraged to improve performance-efficiency trade-off fronts of neural networks on image classification tasks?

In Chapter 2, we developed a method to create cascades of neural network architectures from different supernetworks. Using a multi-objective EA, we optimized the models to include into a cascade and the thresholds for deciding by how many models in the cascade any given sample should be processed. We have established that utilizing models from more supernetworks in this way is beneficial, increasing the hypervolume of the performance-efficiency trade-off front.

We have further developed a version of our method that jointly trains the supernetworks and decides on their order in the cascade, as well as on the thresholds for their use. This did not strongly improve results, potentially due to the increased complexity of the joint optimization under limited computational budget.

While we demonstrated using multiple supernetworks to improve trade-off fronts, the improvements are on the order of percentage points, which may be meaningful in some but not all settings. Furthermore, supernetworks inherently limit the architectures that can be found, allowing only for modest improvements. We can conclude that considering multiple supernetworks is well-suited for improving trade-off fronts but not for achieving performance breakthroughs.

Research Question: Chapter 3

Can PBT be extended from the optimization of training hyperparameters to general-purpose NAS in challenging search spaces?

In Chapter 3, we extended PBT to NAS by mixing architectures in the population during training, and adapting the weights via shrink-perturb. Our algorithm, PBT-NAS, outperformed baselines in challenging settings of reinforcement learning and image generation, demonstrating its effectiveness and generality. PBT-NAS was also successfully applied to the extended version of the search space for the image generation task that included options that were set manually in prior work.

Unlike previous efficient algorithms relying on continuous relaxation of the architecture search problem (in essence optimizing the weighted sum of the operations), PBT-NAS does not require outputs of all potential operations at the same position to have the same shape. However, it does require them to be interoperable in the context of the network (e.g., if the next operation is convolution, the input resolution may vary but the number of channels must stay fixed). Further research would be required to remove this limitation.

Being able to efficiently search in challenging search spaces seems vital to achieving architectural breakthroughs. The lack of such breakthroughs produced by NAS to date indicates that further thought has to be put into defining and combing through expressive search spaces, or, alternatively, that the task is infeasible with current computational resources, which would therefore be better applied elsewhere. See further discussion in Section 7.2.

Research Question: Chapter 4

How do the PBT variants perform relative to each other on image classification and reinforcement learning tasks, and what are the limitations of these algorithms?

In Chapter 4, we benchmarked 5 PBT variants on image classification and reinforcement learning tasks. No PBT variant was found to consistently outperform others across considered settings. Interestingly, we observed that Bayesian PBT variants can achieve worse performance than the standard PBT due to greedily optimizing short-term performance gains. Examining theoretical properties of Bayesian PBTs, we found that their returns asymptotically approach the returns of the greedy schedule, rather than the optimal one, as previously believed.

When the step size of PBT variants is changed, different variants perform best, with PBT variants leveraging Bayesian optimization performing better on image classification tasks when the step size is large (as they can better optimize the objective given only a few total evaluations) and worse when the step size is small (as they optimize short-term performance too aggressively). Non-Bayesian PBTs tend to perform worse with larger step sizes (since they optimize less effectively) and better with smaller step sizes (for the same reason, they are less effective at being greedy). The results on reinforcement learning tasks do not show clear patterns; interestingly, FIRE-PBT, a non-Bayesian PBT

variant designed for optimizing long-term performance, tends to perform poorly on reinforcement learning tasks despite excellent image classification performance.

Per-task tuning of all hyperparameters of each PBT variant can be expected to improve the performance of each PBT variant but seems implausible in a realistic setting due to high computational costs of running an entire hyperparameter optimization procedure several times. Nonetheless, several step size values should be tried when benchmarking against PBT variants to avoid reporting overly negative results. It would be interesting to consider whether a more extensive evaluation of PBT variants could give rise to a predictor capable of recommending a good step size based on the properties of the target task. Furthermore, benchmarking the algorithms on more than the two problem classes considered in this chapter (image classification and reinforcement learning) could help in deriving more general conclusions and establishing broader applicability (or lack thereof) of PBT variants. The popularity of PBT variants in the reinforcement learning literature, but not beyond it, hints that their dynamic nature may be an especially good fit for dynamic settings.

Research Question: Chapter 5

Can a PBT variant be developed where the step size is automatically adjusted within a single run in an efficient manner?

In Chapter 5, we introduced IPBT, a novel PBT variant that adjusts its step size within a single run via aggressive partial restarts with task-agnostic information transfer. A restart is triggered based on data-driven performance stagnation criteria. After a restart, the step size is doubled, the weights are adapted via shrink-perturb, and the hyperparameters are reset via a meta Bayesian optimization procedure that learns across restarts to predict initial hyperparameter values leading to good long-term performance.

In experiments across 8 image classification and reinforcement learning tasks (used in Chapter 4 to benchmark PBT variants), IPBT on average outperforms or matches 5 previous PBT variants with tuned step sizes. Furthermore, our ablation studies showed the benefit of the components of IPBT. Interestingly, introducing some randomness on restarts is beneficial compared to introducing none or relying on random restarts entirely.

In the future, it could be interesting to combine IPBT with PBT-NAS from Chapter 3 to enable simultaneous architecture and hyperparameter optimization in an effective and efficient manner. Both algorithms partially reset weights via shrink-perturb, which has thus demonstrated its general applicability within the PBT paradigm.

Nonetheless, it is not entirely clear why shrink-perturb works and how it could be improved upon. Better understanding and quantification of the state of weights (e.g., percentage of original network capacity that can still be directed towards solving a problem (rather than being stuck in an undesirable state)) would allow for further progress in deep-learning-specific hyperparameter optimization that incorporates efficient weight reuse. Research in this direction would also be well-positioned to help in any other subarea of deep learning where understanding weights is useful, therefore we consider it as potentially very impactful.

Research Question: Chapter 6

Can a distributed and asynchronous learning system based on synthetic data be constructed that would be applicable to varying types and quantities of data without manual hyperparameter adjustment?

In Chapter 6, we introduced HyFree-S3, a hyperparameter-free distributed learning approach for synthesizing data, and we used it to improve performance on medical segmentation tasks. For segmentation, the nnU-Net framework was used. Its model scaling heuristics were transferred to StyleGAN2 for image generation, allowing it to automatically address inputs of different resolutions. We introduced additional simple heuristics that adjust the training time and the number of generated images based on the size of the training dataset, thus avoiding overfitting and data memorization.

To measure data memorization, we proposed leveraging a pretrained embedding model to create representations of both real and synthetic images. A threshold is computed based on similarity between unrelated real images, and then used to remove synthetic images that are too close to real images.

Synthetic data can be generated asynchronously, and then gathered at a central location to pretrain a segmentation model. The pretrained model can then be fine-tuned on real data at each site. We show that this procedure generally improves robustness of the final model to distribution shifts. Furthermore, this procedure was successfully applied to different types (brachytherapy planning MRIs of the cervix, lung X-rays, polyp photos) and quantities of data without any manual hyperparameter adjustments, outperforming untuned federated learning baselines[1] and demonstrating the effectiveness of the proposed heuristics.

Despite encouraging results, the approach has some limitations, such as reliance on nnU-Net for data preprocessing, requiring sufficient amounts of data at each location to train the models, and not utilizing volume information in 3D imaging data (due to operating on 2D slices). While training 3D models requires more data and computational resources, it could be valuable in practice. However, to the best of our knowledge, there is no 3D generative model that works as robustly out of the box as StyleGAN2 with differentiable augmentations does for 2D data.

It could be interesting to consider if, instead of training local models from scratch, foundation models could be fine-tuned or be conditioned on local data to create synthetic images for sharing.

Detecting memorization is a topic as vital to the prospect of synthetic data as it is vague. Clearly, a synthetic image should be sufficiently distinct from all the real images, but drawing a line can be difficult. Being too cautious has a cost of slower progress and a risk of worse patient outcomes compared to what could be achieved with better data-driven approaches. Therefore, reasonably realistic privacy attacks should be considered (in line with The Article 29 Working Party on Data Protection (2014)), with the synthetic data being evaluated on how well it protects against such risks. While general implementation

[1] It could be interesting to see if more robust federated learning algorithms and implementations (than those considered by us) could be found or how much effort would have to be put into hyperparameter tuning in order to bring the performance of the considered baselines to the level of HyFree-S3.

of such attacks is available for tabular data (Giomi et al., 2023), no such approaches exist for image data (to the best of our knowledge). Developing better privacy evaluation for medical images appears to be a valuable and impactful direction of future work.

7.2. On neural architecture search

More than a thousand papers have been published on the topic of NAS (White et al., 2023). Nonetheless, the currently dominant architecture, Transformer (Vaswani et al., 2017), has not been superseded nor substantially improved upon via architecture optimization: while many works have reported discovering improvements via NAS (So et al., 2019; M. Chen et al., 2021b), none of them have been adopted in practice.

The effectiveness of the transformer calls into question one of the underlying assumptions of NAS: that a task should strongly benefit from a specialized architecture. Since their introduction in 2017 for neural machine translation, transformers have been widely used not only in natural language processing but also computer vision (Dosovitskiy et al., 2021), tabular data processing (Hollmann et al., 2025), and other areas (Baevski et al., 2020; L. Chen et al., 2021). It appears that scaling a general architecture such as transformer (and increasing the quantity of data) enables strong performance improvement while requiring minimal architecture adaptations. Thus, when solving a task with deep learning in the present day, it may be more worthwhile to invest into collecting data to train a larger model instead of optimizing a task-specific architecture.

While NAS has been applied to adapting or improving transformer variants (So et al., 2019; J. Gao et al., 2022; C. Gong et al., 2022), in practice, NAS methods are conspicuously absent from papers describing architectural improvements used in the frontier models (An Yang et al., 2024; DeepSeek-AI et al., 2025). Possibly, computational costs of architecture optimization of large-scale models would eclipse salaries of AI researchers, motivating the more efficient and intelligent but manual researcher-guided exploration of architectures. Additionally, optimization may overfit, prevention of which requires careful objective design and sufficient validation data.

On the other hand, the quality and quantity of the available data generally limits what can be done with any model. When trained from scratch on a small dataset, convolutional networks outperform vision transformers (Y. Liu et al., 2021), showing that the benefits of well-chosen architectural bias are still relevant under certain conditions.

To the best of our knowledge, NAS never discovered substantially useful architectural bias that was not previously known. We believe this largely to come down to the used search spaces, which most often consist of hyperparameters of known operations (as described in Section 1.3.2). Naturally, no breakthroughs can be expected from tinkering with well-established building blocks. However, reliance on them makes optimization feasible, allowing at least some gains to be made. For novel discoveries, less constrained spaces are likely to be required, the exploration of which may unfortunately need currently infeasible amounts of compute.

This can be clearly seen in the results of Real et al. (2020), where 10,000 CPU cores were utilized for 5 days to evolve machine learning algorithms in a less-restricted space, ending up rediscovering known techniques but not new reusable improvements. Prohibitive

computational costs prevent building up on these results or scaling them up — for the time being.

Nonetheless, this direction of research remains promising in principle and will become more attractive as the cost of computational resources decreases over time. The recent history of the transformer adoption has shown that if a scalable and effective operation is discovered, it can achieve massive impact even if large amounts of compute are required. Therefore, if investing significant computational resources into search could plausibly yield an operation that scales better than transformers, such an investment would plausibly be made, with the resulting operation replacing transformer as a basis of large models and thus impacting many areas.

While NAS is currently not particularly effective for achieving state-of-the-art results with frontier models, it nonetheless offers clear benefits at the other end of the spectrum, where smaller good-enough models are required for use in compute-light environments. Multi-objective NAS can currently be utilized for edge deployment, for which multiple models need to be found on a multi-objective trade-off front between performance and latency, model size, or another objective (Benmeziane et al., 2023).

7.3. On hyperparameter optimization

During poster sessions of AI conferences, the author of this thesis would ask researchers how they optimized hyperparameters of their methods. The responses — while not constituting a rigorous study — are nonetheless informative. The vast majority of people adapted hyperparameters in an ad hoc manner by manually trying different ones for different amounts of time and developing an intuition about the problem. At best, grid search would be utilized in the final stage of the project.

Given that substantial effort has been directed at hyperparameter optimization in this thesis and in the subfield in general, this anecdotal evidence is disappointing. Why would modern hyperparameter optimization algorithms not be widely used in deep learning research?

Firstly, some status quo bias is present. Researchers rely on grid search because it is clear and widely accepted. Random search may appear more arbitrary and less "scientific", even though it was shown to perform better than grid search for deep learning tasks (Bergstra & Bengio, 2012; Lorraine et al., 2020). To the best of our knowledge, this result has not been disputed, and yet grid search persists, despite random search being equally easy to understand and implement. Clearly, changing habits of researchers and conventions of scientific fields is difficult and requires better communication from the hyperparameter optimization community.

Secondly, to use hyperparameter optimization algorithms more complicated than grid or random search, the user's code likely needs to be restructured to encapsulate the training procedure in a function that accepts hyperparameter values and returns performance, rather than a script with inputs provided via CLI and outputs logged as structured text. In our experience, the latter is a more true-to-life description of how code is typically written during deep learning research. This forms a barrier to trying out hyperparameter optimization. However, libraries not focused on hyperparameter optimization such as PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019), Hydra (Yadan, 2019), or Weights & Biases (Biewald, 2020) have started including some

hyperparameter optimization functionality, making it more accessible for their users. Unfortunately, some extra effort may still be required to use more involved algorithms implemented in libraries such as Ray Tune (Liaw et al., 2018), not to speak of bespoke implementations of state-of-the-art algorithms that are often available only as stand-alone git repositories.

Thirdly, a hyperparameter optimization algorithm needs to plausibly produce returns sufficient to justify the extra implementation and computational effort, i.e., the relevant metrics should strongly improve. The performance of the algorithms is often reported on relatively simple tabular and surrogate benchmarks (Dong & Yang, 2020; Shala et al., 2024), where the absolute performance differences to random search are frequently small, and the improvements are claimed based on algorithm ranks or normalized regret relative to the best found solution (Awad et al., 2021). Yet, a user of a hyperparameter optimization algorithm is unlikely to care for either, as it is not clear what concrete improvement in absolute terms can be achieved on a more involved and compute-intensive practical task that the user cares about.

Fourthly, users can be expected to avoid any hyperparameter optimization algorithm that requires (manual) tuning of its own hyperparameters, as this largely defeats the purpose. Thus, the algorithm should deliver reasonably good performance out of the box. Furthermore, the algorithm has to perform robustly across a range of tasks in order for a researcher to consider it a reliable tool.

Finally, and particularly for deep learning (where evaluations are expensive), researchers cannot afford to run general-purpose hyperparameter optimization algorithms, as a more efficient and deep-learning-specific algorithm is likely needed to deliver good results with few evaluations, potentially by leveraging the user's intuition about the task at hand.

Our understanding of these issues motivated the design of our latest algorithm, IPBT, as efficient and working out of the box. However, prior knowledge of the user is not taken into account, nor does the dynamic approach to optimization enable easy integration with arbitrary tasks and codebases. While uniform priors for each hyperparameter could be replaced with expert-defined priors (e.g., similarly to (Mallik et al., 2023)), it is not clear if designing dynamic hyperparameter optimization libraries to be as usable as those for static optimization is feasible. This is due to the gray-box nature of dynamic optimization that requires more interaction with the user code, thus increasing complexity and coupling. Consider that intermediate results need to be passed between the library and the user code, introducing extra logic and control flow changes that are absent in black-box optimization where the user code can train the model end-to-end and only return the final result.

We believe that addressing all the concerns discussed in this section is vital in enabling hyperparameter optimization algorithms to go fully mainstream and to realize their potential for improving deep learning research and applications.

7.4. The future of hyperparameter optimization for deep learning

Recent progress in AI has been enabled by scaling up models, data, and computational effort. The effectiveness of this paradigm is likely to remain relevant for the foreseeable future[2]. However, gathering high-quality data may prove difficult for many domains, as laws and public perception limit data sharing or selling. Synthetic data appears to be well-suited to address these concerns, with automatic creation of it on location (similarly to what was done in this thesis with HyFree-S3) as a promising practical direction.

Once data — real, synthetic, or both — is gathered, hyperparameters of the models trained on it will still have to be determined. For the frontier large models, even training as few as 8 instances (like in IPBT) is too expensive. The only seemingly feasible way forward lies in optimizing hyperparameters of smaller models and figuring out how to transfer them (or insights gained from them) to the larger ones. G. Yang et al. (2021) showed that careful parameterization of neural network weights enables zero-shot transfer of some hyperparameters across model sizes. Alternatively, computational resources could be invested into constructing a dataset of hyperparameters, model sizes, and performances over time, afterwards leveraging this data to create hyperparameter performance predictors. This approach would involve high upfront costs but appears inherently feasible, as it does not require research breakthroughs. Thus, it could be valuable if more intelligent (and less compute-intensive) methods for setting hyperparameters would stall.

While the best performance will likely continue to be achieved by large models, smaller models will stay relevant as long as latency, energy usage, and other real-world constraints remain. Consequently, methods for setting hyperparameters of such models will be required, be it for training from scratch or for distillation from larger models. As hardware improves and available computational power grows, it will become possible to evaluate more configurations for a model of a given size, potentially further increasing the practical usefulness and popularity of hyperparameter optimization algorithms.

For setting hyperparameters of smaller models, the algorithms built on the same principles as those in this thesis are likely to remain relevant. Bayesian optimization can produce great results with few evaluations. Evolutionary algorithms enable usage of modern parallel hardware and infrastructure while also being excellently-suited for multi-objective problems (which are ubiquitous). Dynamic hyperparameter optimization both leads to better performance by discovering a schedule, and naturally avoids wasting resources on unpromising hyperparameter settings while exploring the promising ones.

Continuing to integrate these techniques and construct robust, efficient, and easy-to-use hyperparameter optimization algorithms could lead to further simplification and expansion of deep learning applications, thus enabling more intelligent resource usage, insight generation, and problem-solving across our modern information-driven civilization.

[2] At the moment of writing (autumn 2025), the rate of improvement from increasing the size of large language models has decreased. In our opinion, regarding this as the end of scaling is premature, since other components of the training pipeline could be scaled up instead, as demonstrated for test-time computations and as could be imagined for e.g., volume of high-quality data.

Acknowledgements

The cover of this book lists but a single author, yet a PhD is no solitary endeavor. I am deeply grateful to everyone who helped, directly or indirectly, see this project through.

My thanks go first and foremost to my brilliant supervisors. Peter and Tanja, you have taught me much about conducting high-quality research, writing clearly and precisely, and persevering towards perfection. I appreciate our many candid discussions on topics practical and abstract, fun and serious. I appreciate the moral support during times that were difficult for me personally and for the world at large. I appreciate the care you've shown.

My wonderful paranymphs have also played a huge role in my PhD. Arkadiy, it was immensely fun to work in the same office as you, more fun to work together, and even more fun to hang out and explore the Netherlands in our free time. Marco, it was no less enjoyable to spend time with you! I also appreciate your guidance that helped me figure out (among other things) how to pivot my work to get to the first publication (which was a huge deal for me) and how to successfully transition from academia to industry. I'm delighted to call both of you my friends.

Friends have added color and enjoyment to my time not devoted to research (and motivated me to have more such time!). Kirill & Masha, Vera, Dafni, Ilya S, Shane — thank you for our conversations, our walks, our museum visits, our time spent together doing all kinds of things (many of which I wouldn't have tried on my own). Ilya V, thank you for our calls that often stretched into the night. And thank you all for the unwavering support.

My family has been another source of boundless solace and encouragement (as well as fun, advice, and relaxation). Mom, dad, Tyoma — lots of love! Words are not enough — I'll hug you instead.

Although I will not hug my colleagues, I would still like to thank them. Leen, Stef, Timo — for sage advice at the start of this path. Monika — for being positive and for bringing interesting technical problems to discuss. Joe — for showing by example how easy and fun doing atypical things can be. Vanessa — for deep discussions of careers in industry and academia. Arthur, Vangelis — for your help and efforts on our shared project. Everyone already mentioned, as well as Anton, Cedric, Damy, Evi, Georgios, Koen, Leah, Maaike, Mafalda, Pedro, Renzo, Thalea — for engaging conversations during lunch, coffee, and other breaks.

I'm a more fleshed-out human being thanks to the events organized by the CWI PhD Activity Committee — I appreciate everyone who put in their time and effort to make them happen. The same goes for the Taalcafé Stek Oost, which also greatly helped me to (partially) integrate into Dutch society and feel at home here (special thanks go to Rosanne for being the driving force all these years).

I thank CWI for the support I received throughout my PhD. My research would not have been possible without help with both documents and our GPU servers. Of course, there would be no GPUs if not for the funding from NWO, Elekta, and Ortec (occasionally,

SURF and TU Delft contributed extra computational resources, which were also greatly appreciated). LUMC gets my thanks for the GPUs, the data, and the overall support of the project.

Begrudgingly, I thank TU Delft Graduate School for requiring PhD candidates to take numerous courses, since quite a few of those greatly contributed to my personal and professional development.

My final thanks go to all the scientists & artists, thinkers & doers who inspired me to pursue this path.

Bibliography

Abdelfattah, Mohamed S., Abhinav Mehrotra, Lukasz Dudziak & Nicholas Donald Lane. 2021. Zero-Cost Proxies for Lightweight NAS. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=0cmMMy8J5q. (Cited on pages 38, 40).

Abràmoff, Michael D., Philip T. Lavin, Michele Birch, Nilay Shah & James C. Folk. 2018. Pivotal Trial of an Autonomous AI-based Diagnostic System for Detection of Diabetic Retinopathy in Primary Care Offices. *npj Digital Medicine* 1, no. 1 (August 28, 2018): 39. ISSN: 2398-6352. https://doi.org/10.1038/s41746-018-0040-6. (Cited on page 1).

Adnan, Mohammed, Shivam Kalra, Jesse C. Cresswell, Graham W. Taylor & Hamid R. Tizhoosh. 2022. Federated Learning and Differential Privacy for Medical Image Analysis. *Number: 1 Publisher: Nature Publishing Group, Scientific Reports* 12, no. 1 (February): 1953. ISSN: 2045-2322. https://doi.org/10.1038/s41598-022-05539-7. (Cited on pages 10, 107).

Adriaensen, Steven, André Biedenkapp, Gresa Shala, Noor Awad, Theresa Eimer, Marius Lindauer & Frank Hutter. 2022. Automated Dynamic Algorithm Configuration. *Journal of Artificial Intelligence Research* 75 (December): 1633–1699. ISSN: 1076-9757. https://doi.org/10.1613/jair.1.13922. (Cited on page 63).

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville & Marc G. Bellemare. 2021. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 29304–29320. https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html. (Cited on pages 94, 102).

Ainsworth, Samuel K., Jonathan Hayase & Siddhartha S. Srinivasa. 2023. Git Re-Basin: Merging Models modulo Permutation Symmetries. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=CQsmMYmlP5T. (Cited on page 41).

Angelova, Anelia, Alex Krizhevsky, Vincent Vanhoucke, Abhijit S. Ogale & Dave Ferguson. 2015. Real-Time Pedestrian Detection with Deep Network Cascades. In *Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015,* 32.1–32.12. BMVA Press. https://doi.org/10.5244/C.29.32. (Cited on page 17).

Ash, Jordan T., & Ryan P. Adams. 2020. On Warm-Starting Neural Network Training. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*. https://proceedings.neurips.cc/paper/2020/hash/288cd2567953f06e460a33951f55daaf-Abstract.html. (Cited on pages 38, 39, 41, 43, 86, 90, 92).

Awad, Noor H., Neeratyoy Mallik & Frank Hutter. 2021. DEHB: Evolutionary Hyberband for Scalable, Robust and Efficient Hyperparameter Optimization. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, 2147–2153. https://doi.org/10.24963/IJCAI.2021/296. (Cited on pages 9, 62, 88, 129).

Ba, Lei Jimmy, Jamie Ryan Kiros & Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450. arXiv: 1607.06450. http://arxiv.org/abs/1607.06450. (Cited on page 5).

Badia, Adrià Puigdomènech, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo & Charles Blundell. 2020. Agent57: Outperforming the Atari Human Benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, 119: 507–517. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v119/badia20a.html. (Cited on pages 60, 63).

Baevski, Alexei, Yuhao Zhou, Abdelrahman Mohamed & Michael Auli. 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual Event*. https://proceedings.neurips.cc/paper/2020/hash/92d1e1eb1cd6f9fba3227870bb6d7f07-Abstract.html. (Cited on page 127).

Bahdanau, Dzmitry, Kyunghyun Cho & Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1409.0473. (Cited on page 14).

Baker, Bowen, Otkrist Gupta, Ramesh Raskar & Nikhil Naik. 2018. Accelerating Neural Architecture Search using Performance Prediction. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=HJqk3N1vG. (Cited on pages 7, 14).

Baydin, Atilim Gunes, Robert Cornish, David Martínez-Rubio, Mark Schmidt & Frank Wood. 2018. Online Learning Rate Adaptation with Hypergradient Descent. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BkrsAzWAb. (Cited on pages 60, 63).

Benmeziane, Hadjer, Hamza Ouarnoughi, Kaoutar El Maghraoui & Smail Niar. 2023. Multi-objective Hardware-aware Neural Architecture Search with Pareto Rank-preserving Surrogate Models. *ACM Transactions on Architecture and Code Optimization* 20 (2): 29:1–29:21. https://doi.org/10.1145/3579853. (Cited on page 128).

Bergstra, James, Rémi Bardenet, Yoshua Bengio & Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain,* 2546–2554. https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html. (Cited on pages 60, 62, 88).

Bergstra, James, & Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research* 13 (10): 281–305. ISSN: 1533-7928. http://jmlr.org/papers/v13/bergstra12a.html. (Cited on pages 8, 40, 128).

Bernal, Jorge, F. Javier Sánchez, Gloria Fernández-Esparrach, Debora Gil, Cristina Rodríguez & Fernando Vilariño. 2015. WM-DOVA Maps for Accurate Polyp Highlighting in Colonoscopy: Validation vs. Saliency Maps From Physicians. *Computerized Medical Imaging and Graphics* 43 (July): 99–111. ISSN: 0895-6111. https://doi.org/10.1016/j.compmedimag.2015.02.007. (Cited on page 110).

Biedenkapp, André, H. Furkan Bozkurt, Theresa Eimer, Frank Hutter & Marius Lindauer. 2020. Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020),* 325: 427–434. Frontiers in Artificial Intelligence and Applications. IOS Press. https://doi.org/10.3233/FAIA200122. (Cited on page 63).

Biewald, Lukas. 2020. Experiment Tracking with Weights and Biases. https://www.wandb.com/. (Cited on page 128).

Bjorck, Johan, Carla P. Gomes & Kilian Q. Weinberger. 2021. Towards Deeper Deep Reinforcement Learning with Spectral Normalization. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 8242–8255. https://proceedings.neurips.cc/paper/2021/hash/4588e674d3f0faf985047d4c3f13ed0d-Abstract.html. (Cited on page 47).

Blank, J., K. Deb, Y. Dhebar, S. Bandaru & H. Seada. 2020. Generating Well-Spaced Points on a Unit Simplex for Evolutionary Many-Objective Optimization. *IEEE Transactions on Evolutionary Computation* 25 (1): 48–60. (Cited on page 30).

Bogunovic, Ilija, Jonathan Scarlett & Volkan Cevher. 2016. Time-Varying Gaussian Process Bandit Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016,* 51: 314–323. JMLR Workshop and Conference Proceedings. JMLR.org. http://proceedings.mlr.press/v51/bogunovic16.html. (Cited on pages 64, 66, 89).

Borgli, Hanna, Vajira Thambawita, Pia H Smedsrud, Steven Hicks, Debesh Jha, Sigrun L Eskeland, Kristin Ranheim Randel, et al. 2020. HyperKvasir, a Comprehensive Multi-Class Image and Video Dataset for Gastrointestinal Endoscopy. *Scientific Data* 7 (1): 283. ISSN: 2052-4463. https://doi.org/10.1038/s41597-020-00622-y. (Cited on page 110).

Bosman, Peter A. N. 2005. Learning, Anticipation and Time-Deception in Evolutionary Online Dynamic Optimization. In *BNAIC 2005 - Proceedings of the Seventeenth Belgium-Netherlands Conference on Artificial Intelligence, Brussels, Belgium, October 17-18, 2005,* 321–322. Koninklijke Vlaamse Academie van Belie voor Wetenschappen en Kunsten. (Cited on pages 66, 75).

Bouter, Anton, Tanja Alderliesten, Bradley R. Pieters, Arjan Bel, Yury Niatsetski & Peter A. N. Bosman. 2019. GPU-accelerated Bi-Objective Treatment Planning for Prostate High-Dose-Rate Brachytherapy. *Medical Physics* 46 (9): 3776–3787. (Cited on page 18).

Bouter, Anton, Ngoc Hoang Luong, Cees Witteveen, Tanja Alderliesten & Peter A. N. Bosman. 2017. The Multi-Objective Real-Valued Gene-Pool Optimal Mixing Evolutionary Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017,* 537–544. ACM. https://doi.org/10.1145/3071178.3071274. (Cited on page 3).

Bowles, Christopher, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw & Daniel Rueckert. 2018. GAN Augmentation: Augmenting Training Data using Generative Adversarial Networks. *arXiv:1810.10863 [cs]*, October. https://doi.org/10.48550/arXiv.1810.10863. (Cited on pages 106, 107).

Branke, Jürgen, & Hartmut Schmeck. 2003. Designing Evolutionary Algorithms for Dynamic Optimization Problems. In *Advances in Evolutionary Computing,* 239–262. Springer. (Cited on pages 3, 15).

Broomhead, David S, & David Lowe. 1988. Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks. Technical report. Royal Signals and Radar Establishment Malvern (United Kingdom). (Cited on pages 20, 30).

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language Models Are Few-Shot Learners. *Advances in Neural Information Processing Systems* 33: 1877–1901. (Cited on page 14).

Broyden, C. G. 1970. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics* 6, no. 1 (March 1, 1970): 76–90. ISSN: 0272-4960. https://doi.org/10.1093/imamat/6.1.76. (Cited on page 3).

Cai, Han, Chuang Gan, Tianzhe Wang, Zhekai Zhang & Song Han. 2020. Once-for-All: Train One Network and Specialize it for Efficient Deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HylxE1HKwS. (Cited on pages 7, 8, 14, 16, 21, 23, 38, 40).

Cai, Han, Ligeng Zhu & Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=HylVB3AqYm. (Cited on page 21).

Cai, Zhaowei, Mohammad J. Saberian & Nuno Vasconcelos. 2015. Learning Complexity-Aware Cascades for Deep Pedestrian Detection. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 3361–3369. IEEE Computer Society. https://doi.org/10.1109/ICCV.2015.384. (Cited on page 17).

Campbell, Murray, A. Joseph Hoane & Feng-hsiung Hsu. 2002. Deep Blue. *Artif. Intell.* 134, nos. 1–2 (January 24, 2002): 57–83. ISSN: 0004-3702. https://doi.org/10.1016/S0004-3702(01)00129-1. (Cited on page 1).

Chang, Qi, Hui Qu, Yikai Zhang, Mert Sabuncu, Chao Chen, Tong Zhang & Dimitris N. Metaxas. 2020. Synthetic Learning: Learn From Distributed Asynchronized Discriminator GAN Without Sharing Medical Image Data, 13856–13866. https://openaccess.thecvf.com/content_CVPR_2020/html/Chang_Synthetic_Learning_Learn_From_Distributed_Asynchronized_Discriminator_GAN_Without_Sharing_CVPR_2020_paper.html. (Cited on page 107).

Chang, Qi, Zhennan Yan, Mu Zhou, Hui Qu, Xiaoxiao He, Han Zhang, Lohendran Baskaran, et al. 2023. Mining Multi-Center Heterogeneous Medical Data With Distributed Synthetic Learning. *Number: 1 Publisher: Nature Publishing Group, Nature Communications* 14, no. 1 (September): 5510. ISSN: 2041-1723. https://doi.org/10.1038/s41467-023-40687-y. (Cited on pages 107, 117, 118).

Chebykin, Alexander, Tanja Alderliesten & Peter A. N. Bosman. 2025b. To Be Greedy, or Not to Be - That Is the Question for Population Based Training Variants. *Transactions on Machine Learning Research,* ISSN: 2835-8856. https://openreview.net/forum?id=3qmnxysNbi. (Cited on pages 59, 86, 94, 100, 101, 163).

Chebykin, Alexander, Arkadiy Dushatskiy, Tanja Alderliesten & Peter A. N. Bosman. 2023. Shrink-Perturb Improves Architecture Mixing During Population Based Training for Neural Architecture Search. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland*, 372: 381–388. Frontiers in Artificial Intelligence and Applications. IOS Press. https://doi.org/10.3233/FAIA230294. (Cited on pages 37, 61, 90, 163).

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas & Igor Mordatch. 2021. Decision Transformer: Reinforcement Learning via Sequence Modeling. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 15084–15097. https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html. (Cited on page 127).

Chen, Minghao, Jianlong Fu & Haibin Ling. 2021a. One-Shot Neural Ensemble Architecture Search by Diversity-Guided Search Space Shrinking. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, Virtual Event, June 19-25, 2021,* 16530–16539. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR46437.2021.01626. (Cited on pages 14, 16, 23, 31).

Chen, Minghao, Houwen Peng, Jianlong Fu & Haibin Ling. 2021b. AutoFormer: Searching Transformers for Visual Recognition. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021,* 12250–12260. IEEE. https://doi.org/10.1109/ICCV48922.2021.01205. (Cited on page 127).

Chen, Tianqi, Ian J. Goodfellow & Jonathon Shlens. 2016. Net2Net: Accelerating Learning via Knowledge Transfer. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings.* http://arxiv.org/abs/1511.05641. (Cited on page 40).

Chen, Xin, Lingxi Xie, Jun Wu & Qi Tian. 2021. Progressive DARTS: Bridging the Optimization Gap for NAS in the Wild. *International Journal of Computer Vision* 129: 638–655. (Cited on page 40).

Cherti, Mehdi, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt & Jenia Jitsev. 2023. Reproducible Scaling Laws for Contrastive Language-Image Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023,* 2818–2829. IEEE. https://doi.org/10.1109/CVPR52729.2023.00276. (Cited on page 110).

Chiong, Raymond, Thomas Weise & Zbigniew Michalewicz. 2012. Variants of Evolutionary Algorithms for Real-World Applications. Springer. (Cited on pages 15, 18).

Coates, Adam, Andrew Y. Ng & Honglak Lee. 2011. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011,* 15: 215–223. JMLR Proceedings. JMLR.org. http://proceedings.mlr.press/v15/coates11a/coates11a.pdf. (Cited on page 46).

Cowen-Rivers, Alexander I., Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, et al. 2022. HEBO: Pushing The Limits of Sample-Efficient Hyper-parameter Optimisation. *Journal of Artificial Intelligence Research* 74 (July): 1269–1349. ISSN: 1076-9757. https://doi.org/10.1613/jair.1.13643. (Cited on pages 9, 88, 99).

Cubuk, Ekin Dogus, Barret Zoph, Jon Shlens & Quoc Le. 2020. RandAugment: Practical Automated Data Augmentation with a Reduced Search Space. In *Advances in Neural Information Processing Systems,* 33: 18613–18624. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/d85b63ef0ccb114d0a3bb7b7d808028f-Paper.pdf. (Cited on pages 6, 30, 78, 98).

Dalibard, Valentin, & Max Jaderberg. 2021. Faster Improvement Rate Population Based Training. *arXiv:2109.13800 [cs, stat],* September. https://doi.org/10.48550/arXiv.2109.13800. (Cited on pages 10, 38, 60, 61, 63, 64, 79, 90).

Dar, Salman Ul Hassan, Marvin Seyfarth, Jannik Kahmann, Isabelle Ayx, Theano Papavassiliu, Stefan O. Schoenberg & Sandy Engelhardt. 2024. Unconditional Latent Diffusion Models Memorize Patient Imaging Data. *arXiv:2402.01054 [cs, eess],* February. https://doi.org/10.48550/arXiv.2402.01054. (Cited on page 119).

Das, Indraneel, & John E Dennis. 1998. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization* 8 (3): 631–657. (Cited on page 30).

Davison, Anthony Christopher, & David Victor Hinkley. 1997. Bootstrap Methods and Their Application. Cambridge University press. (Cited on page 102).

Deb, Kalyanmoy, & Himanshu Jain. 2013. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation* 18 (4): 577–601. (Cited on pages 16, 18).

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, et al. 2025. DeepSeek-V3 Technical Report, February 18, 2025. https://doi.org/10.48550/arXiv.2412.19437. arXiv: 2412.19437 [cs]. (Cited on page 127).

Degerli, Aysen, Serkan Kiranyaz, Muhammad Enamul Hoque Chowdhury & Moncef Gabbouj. 2022. Osegnet: Operational Segmentation Network for Covid-19 Detection Using Chest X-Ray Images. In *2022 IEEE International Conference on Image Processing, ICIP 2022, Bordeaux, France, 16-19 October 2022,* 2306–2310. IEEE. https://doi.org/10.1109/ICIP46576.2022.9897412. (Cited on page 110).

Devries, Terrance, & Graham W. Taylor. 2017. Improved Regularization of Convolutional Neural Networks with Cutout. *CoRR* abs/1708.04552. arXiv: 1708.04552. http://arxiv.org/abs/1708.04552. (Cited on page 30).

Donahue, Jeff, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng & Trevor Darrell. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014,* 32: 647–655. JMLR Workshop and Conference Proceedings. JMLR.org. http://proceedings.mlr.press/v32/donahue14.html. (Cited on page 17).

Dong, Xuanyi, & Yi Yang. 2019a. One-Shot Neural Architecture Search via Self-Evaluated Template Network. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019,* 3680–3689. IEEE. https://doi.org/10.1109/ICCV.2019.00378. (Cited on pages 14, 24, 25).

Dong, Xuanyi, & Yi Yang. 2019b. Searching for a Robust Neural Architecture in Four GPU Hours. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019,* 1761–1770. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR.2019.00186. (Cited on pages 24, 25).

Dong, Xuanyi, & Yi Yang. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HJxyZkBKDr. (Cited on page 129).

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, et al. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=YicbFdNTTy. (Cited on page 127).

Doulazmi, Waël, Auguste Lehuger, Marin Toromanoff, Valentin Charraut, Thibault Buhet & Fabien Moutarde. 2025. Multiple-Frequencies Population-Based Training, June. https://doi.org/10.48550/arXiv.2506.03225. (Cited on page 90).

Dube, Kudakwashe, & Thomas Gallagher. 2014. Approach and Method for Generating Realistic Synthetic Electronic Healthcare Records for Secondary Use. In *Foundations of Health Information Engineering and Systems,* 69–86. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer. ISBN: 978-3-642-53956-5. https://doi.org/10.1007/978-3-642-53956-5_6. (Cited on page 106).

Duchi, John, Elad Hazan & Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (61): 2121–2159. ISSN: 1533-7928. http://jmlr.org/papers/v12/duchi11a.html. (Cited on page 3).

Dunn, Olive Jean. 1961. Multiple Comparisons Among Means. *J. Amer. Statist. Assoc.* 56 (293): 52–64. (Cited on pages 21, 45, 55, 114).

Duque, Miguel González, Richard Michael, Simon Bartels, Yevgen Zainchkovskyy, Søren Hauberg & Wouter Boomsma. 2024. A Survey and Benchmark of High-Dimensional Bayesian Optimization Of Discrete Sequences. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*. http://papers.nips.cc/paper_files/paper/2024/hash/fe0007fcfd707673660ec0f9014bc48e-Abstract-Datasets_and_Benchmarks_Track.html. (Cited on page 3).

Dushatskiy, Arkadiy, Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2023. Multi-Objective Population Based Training. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, 202: 8969–8989. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v202/dushatskiy23a.html. (Cited on pages 61, 99, 163).

Efron, Bradley, & Robert Tibshirani. 1993. An Introduction to the Bootstrap. Springer. ISBN: 978-1-4899-4541-9. https://doi.org/10.1007/978-1-4899-4541-9. (Cited on pages 94, 102).

Eimer, Theresa, Marius Lindauer & Roberta Raileanu. 2023. Hyperparameters in Reinforcement Learning and How To Tune Them. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, 202: 9104–9149. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v202/eimer23a.html. (Cited on page 78).

Elsayed, Mohamed, Qingfeng Lan, Clare Lyle & A. Rupam Mahmood. 2024. Weight Clipping for Deep Continual and Reinforcement Learning. *RLJ* 5: 2198–2217. https://rlj.cs.umass.edu/2024/papers/Paper307.html. (Cited on page 90).

Elsken, Thomas, Jan Hendrik Metzen & Frank Hutter. 2019. Efficient Multi-Objective Neural Architecture Search via Lamarckian Evolution. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=ByME42AqK7. (Cited on page 40).

Esposito, Roberto, & Lorenza Saitta. 2003. Monte Carlo Theory as an Explanation of Bagging and Boosting. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, 499–504. Morgan Kaufmann. http://ijcai.org/Proceedings/03/Papers/074.pdf. (Cited on page 14).

Falcon, William, & The PyTorch Lightning team. 2019. PyTorch Lightning. V. 1.4, March. https://doi.org/10.5281/zenodo.3828935. (Cited on page 128).

Falkner, Stefan, Aaron Klein & Frank Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 80: 1436–1445. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v80/falkner18a.html. (Cited on pages 9, 40, 54, 62, 88).

Feng, Qianli, Chenqi Guo, Fabian Benitez-Quiroz & Aleix M. Martínez. 2021. When do GANs replicate? On the choice of dataset size. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, 6681–6690. IEEE. https://doi.org/10.1109/ICCV48922.2021.00663. (Cited on page 109).

Feurer, Matthias, & Frank Hutter. 2019. Hyperparameter Optimization. In *Automated Machine Learning - Methods, Systems, Challenges*, 3–33. The Springer Series on Challenges in Machine Learning. Springer. https://doi.org/10.1007/978-3-030-05318-5_1. (Cited on pages 60, 86).

Franke, Jörg K. H., Gregor Köhler, André Biedenkapp & Frank Hutter. 2021. Sample-Efficient Automated Deep Reinforcement Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=hSjxQ3B7GWq. (Cited on pages 9, 38, 41, 47, 56, 61, 99).

Freeman, C. Daniel, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch & Olivier Bachem. 2021. Brax - A Differentiable Physics Engine for Large Scale Rigid Body Simulation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, Virtual Event*. https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/d1f491a404d6854880943e5c3cd9ca25-Abstract-round1.html. (Cited on pages 61, 67, 78, 88, 94, 101).

Friedman, Milton. 1937. The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Publisher: [American Statistical Association, Taylor & Francis, Ltd.] J. Amer. Statist. Assoc.* 32 (200): 675–701. ISSN: 0162-1459. https://doi.org/10.2307/2279372. (Cited on page 81).

Gao, Chen, Yunpeng Chen, Si Liu, Zhenxiong Tan & Shuicheng Yan. 2020. AdversarialNAS: Adversarial Neural Architecture Search for GANs. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020,* 5679–5688. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR42600.2020.00572. (Cited on pages 38, 40, 45, 48, 56, 58).

Gao, Jiahui, Hang Xu, Han Shi, Xiaozhe Ren, Philip L. H. Yu, Xiaodan Liang, Xin Jiang & Zhenguo Li. 2022. AutoBERT-Zero: Evolving BERT Backbone from Scratch. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022,* 10663–10671. AAAI Press. https://doi.org/10.1609/AAAI.V36I10.21311. (Cited on page 127).

Giomi, Matteo, Franziska Boenisch, Christoph Wehmeyer & Borbála Tasnádi. 2023. A Unified Framework for Quantifying Privacy Risk in Synthetic Data. *Proc. Priv. Enhancing Technol.* 2023 (2): 312–328. https://doi.org/10.56553/POPETS-2023-0055. (Cited on page 127).

Goncalves, Andre, Priyadip Ray, Braden Soper, Jennifer Stevens, Linda Coyle & Ana Paula Sales. 2020. Generation and Evaluation of Synthetic Patient Data. *BMC Medical Research Methodology* 20, no. 1 (May): 108. ISSN: 1471-2288. https://doi.org/10.1186/s12874-020-00977-1. (Cited on pages 10, 107).

Gong, Chengyue, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Qiang Liu & Vikas Chandra. 2022. NASViT: Neural Architecture Search for Efficient Vision Transformers with Gradient Conflict Aware Supernet Training. *International Conference on Learning Representations,* https://openreview.net/forum?id=Qaw16njk6L. (Cited on page 127).

Gong, Xinyu, Shiyu Chang, Yifan Jiang & Zhangyang Wang. 2019. AutoGAN: Neural Architecture Search for Generative Adversarial Networks. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019,* 3223–3233. IEEE. https://doi.org/10.1109/ICCV.2019.00332. (Cited on pages 40, 45, 46).

Good, I. J. 1952. Rational Decisions. *Journal of the Royal Statistical Society. Series B (Methodological)* 14 (1): 107–114. ISSN: 0035-9246. JSTOR: 2984087. https://www.jstor.org/stable/2984087. (Cited on page 6).

Goodfellow, Ian, Yoshua Bengio & Aaron Courville. 2016. Deep Learning. MIT Press. (Cited on page 6).

Gui, Jie, Zhenan Sun, Yonggang Wen, Dacheng Tao & Jieping Ye. 2023. A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. *IEEE Trans. Knowl. Data Eng.* 35 (4): 3313–3332. https://doi.org/10.1109/TKDE.2021.3130191. (Cited on page 44).

Guibas, John T., Tejpal S. Virdi & Peter S. Li. 2018. Synthetic Medical Images from Dual Generative Adversarial Networks. *arXiv:1709.01872 [cs],* January. https://doi.org/10.48550/arXiv.1709.01872. (Cited on pages 106, 107).

Han, Kun, Yifeng Xiong, Chenyu You, Pooya Khosravi, Shanlin Sun, Xiangyi Yan, James S. Duncan & Xiaohui Xie. 2023. MedGen3D: A Deep Generative Framework for Paired 3D Image and Mask Generation. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023,* 14220: 759–769. *Series Title: Lecture Notes in Computer Science*. Cham: Springer Nature Switzerland. ISBN: 978-3-031-43906-3 978-3-031-43907-0. https://doi.org/10.1007/978-3-031-43907-0_72. (Cited on pages 106, 107).

Hasselt, Hado. 2010. Double Q-learning. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada,* 2613–2621. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2010/hash/091d584fced301b442654dd8c23b3fc9-Abstract.html. (Cited on page 47).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren & Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016,* 770–778. IEEE Computer Society. https://doi.org/10.1109/CVPR.2016.90. (Cited on pages 5, 47, 67, 94, 100).

Heusel, Martin, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler & Sepp Hochreiter. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA,* 6626–6637. https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html. (Cited on pages 46, 121).

Hollmann, Noah, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister & Frank Hutter. 2025. Accurate Predictions on Small Data with a Tabular Foundation Model. *Nature* 637, no. 8045 (January): 319–326. ISSN: 1476-4687. https://doi.org/10.1038/s41586-024-08328-6. (Cited on page 127).

Holm, Sture. 1979. A Simple Sequentially Rejective Multiple Test Procedure. *Publisher: [Board of the Foundation of the Scandinavian Journal of Statistics, Wiley], Scandinavian Journal of Statistics* 6 (2): 65–70. ISSN: 0303-6898. https://www.jstor.org/stable/4615733. (Cited on pages 81, 94, 102).

Howard, Andrew, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. 2019. Searching for MobileNetv3. *Proceedings of the IEEE/CVF International Conference on Computer Vision,* 1314–1324. (Cited on page 23).

Hu, Shoukang, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu & Dahua Lin. 2020. DSNAS: Direct Neural Architecture Search Without Parameter Retraining. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020,* 12081–12089. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR42600.2020.01210. (Cited on page 38).

Hu, Xiaodan, Audrey G. Chung, Paul Fieguth, Farzad Khalvati, Masoom A. Haider & Alexander Wong. 2018. ProstateGAN: Mitigating Data Bias via Prostate Diffusion Imaging Synthesis with Generative Adversarial Networks. *arXiv:1811.05817 [cs],* November. https://doi.org/10.48550/arXiv.1811.05817. (Cited on page 107).

Hutter, Frank, Holger H. Hoos & Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers,* 6683: 507–523. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-642-25566-3_40. (Cited on page 40).

Hutter, Frank, Lars Kotthoff & Joaquin Vanschoren, eds. 2019. Automated Machine Learning - Methods, Systems, Challenges. The Springer Series on Challenges in Machine Learning. Springer. ISBN: 978-3-030-05317-8. https://doi.org/10.1007/978-3-030-05318-5. (Cited on page 8).

Ilharco, Gabriel, Mitchell Wortsman, Ross Wightman, Cade Gordon, Nicholas Carlini, Rohan Taori, Achal Dave, et al. 2021. OpenCLIP, July. https://doi.org/10.5281/zenodo.5143773. (Cited on page 110).

Isensee, Fabian, Paul F. Jäger, Simon A. A. Kohl, Jens Petersen & Klaus H. Maier-Hein. 2021. Automated Design of Deep Learning Methods for Biomedical Image Segmentation. *arXiv:1904.08128 [cs], Nature Methods* 18, no. 2 (February): 203–211. ISSN: 1548-7091, 1548-7105. https://doi.org/10.1038/s41592-020-01008-z. (Cited on pages 14, 106, 108, 113).

Izmailov, Pavel, Dmitrii Podoprikhin, Timur Garipov, Dmitry P. Vetrov & Andrew Gordon Wilson. 2018. Averaging Weights Leads to Wider Optima and Better Generalization. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018,* 876–885. AUAI Press. http://auai.org/uai2018/proceedings/papers/313.pdf. (Cited on page 30).

Jaderberg, Max, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, et al. 2017. Population Based Training of Neural Networks. *arXiv: 1711.09846, arXiv:1711.09846 [cs]* (November). http://arxiv.org/abs/1711.09846. (Cited on pages 9, 38, 41, 60, 61, 63, 68, 86, 89).

Jamieson, Kevin G., & Ameet Talwalkar. 2016. Non-stochastic Best Arm Identification and Hyperparameter Optimization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016,* 51: 240–248. JMLR Workshop and Conference Proceedings. JMLR.org. http://proceedings.mlr.press/v51/jamieson16.html. (Cited on pages 9, 88).

Jankowiak, Martin, Geoff Pleiss & Jacob R. Gardner. 2020. Parametric Gaussian Process Regressors. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event,* 119: 4702–4712. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v119/jankowiak20a.html. (Cited on page 91).

Jin, Haifeng, Qingquan Song & 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, KDD 2019, Anchorage, AK, USA, August 4-8, 2019,* 1946–1956. ACM. https://doi.org/10.1145/3292500.3330648. (Cited on page 40).

Jin, Yuchen, Tianyi Zhou, Liangyu Zhao, Yibo Zhu, Chuanxiong Guo, Marco Canini & Arvind Krishnamurthy. 2021. AutoLRS: Automatic Learning-Rate Schedule by Bayesian Optimization on the Fly. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=SlrqM9_lyju. (Cited on page 63).

Jordan, M. I., & T. M. Mitchell. 2015. Machine Learning: Trends, Perspectives, and Prospects. *Science* 349, no. 6245 (July 17, 2015): 255–260. https://doi.org/10.1126/science.aaa8415. (Cited on page 1).

Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen & Timo Aila. 2020. Analyzing and Improving the Image Quality of StyleGAN. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020,* 8107–8116. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR42600.2020.00813. (Cited on pages 106, 108).

Kingma, Diederik P., & Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. http://arxiv.org/abs/1412.6980. (Cited on page 3).

Klyuchnikov, Nikita, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov & Evgeny Burnaev. 2022. NAS-Bench-NLP: Neural Architecture Search Benchmark for Natural Language Processing. *IEEE Access* 10: 45736–45747. (Cited on page 38).

Konečný, Jakub, H. Brendan McMahan, Daniel Ramage & Peter Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence, October. https://doi.org/10.48550/arXiv.1610.02527. (Cited on pages 10, 107).

Kornblith, Simon, Jonathon Shlens & Quoc V. Le. 2019. Do Better ImageNet Models Transfer Better? In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2661–2671. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR.2019.00277. (Cited on page 22).

Krizhevsky, Alex, & Geoffrey Hinton. 2009. Learning Multiple Layers of Features from Tiny Images. *Publisher: Toronto, ON, Canada, Master's thesis, University of Toronto*, https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. (Cited on pages 20, 46, 61, 67, 74, 78, 86, 94, 100).

Kukenys, Ignas, & Brendan McCane. 2008. Classifier Cascades for Support Vector Machines. In *2008 23rd International Conference Image and Vision Computing New Zealand*, 1–6. IEEE. (Cited on page 17).

LeCun, Yann, Yoshua Bengio, et al. 1995. Convolutional Networks for Images, Speech, and Time Series. *The Handbook of Brain Theory and Neural Networks* 3361 (10): 1995. (Cited on page 14).

Li, Junyi, Bin Gu & Heng Huang. 2022. A Fully Single Loop Algorithm for Bilevel Optimization without Hessian Inverse. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, 7426–7434. AAAI Press. https://doi.org/10.1609/AAAI.V36I7.20706. (Cited on pages 60, 63).

Li, Liam, Kevin G. Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Bentzur, Moritz Hardt, Benjamin Recht & Ameet Talwalkar. 2020. A System for Massively Parallel Hyperparameter Tuning. In *Proceedings of the Third Conference on Machine Learning and Systems, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2020/hash/a06f20b349c6cf09a6b171c71b88bbfc-Abstract.html. (Cited on pages 9, 88, 89, 94).

Li, Liam, & Ameet Talwalkar. 2019. Random Search and Reproducibility for Neural Architecture Search. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*, 115: 367–377. Proceedings of Machine Learning Research. AUAI Press. http://proceedings.mlr.press/v115/li20c.html. (Cited on pages 31, 39, 51).

Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh & Ameet Talwalkar. 2018. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *Journal of Machine Learning Research* 18 (185): 1–52. ISSN: 1533-7928. http://jmlr.org/papers/v18/16-558.html. (Cited on pages 62, 88).

Liang, Jason Zhi, Santiago Gonzalez, Hormoz Shahrzad & Risto Miikkulainen. 2021. Regularized Evolutionary Population-Based Training. In *GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, July 10-14, 2021,* 323–331. ACM. https://doi.org/10.1145/3449639.3459292. (Cited on pages 38, 40).

Liaw, Richard, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez & Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *CoRR* abs/1807.05118. arXiv: 1807.05118. http://arxiv.org/abs/1807.05118. (Cited on page 129).

Lindauer, Marius, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass & Frank Hutter. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *Journal of Machine Learning Research* 23 (54): 1–9. ISSN: 1533-7928. http://jmlr.org/papers/v23/21-0888.html. (Cited on pages 88, 94).

Liu, Chenxi, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang & Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I,* 11205: 19–35. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-030-01246-5_2. (Cited on page 7).

Liu, Hanxiao, Karen Simonyan & Yiming Yang. 2019. DARTS: Differentiable Architecture Search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. https://openreview.net/forum?id=S1eYHoC5FX. (Cited on pages 8, 14, 16, 25, 38, 40, 45).

Liu, Yahui, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri & Marco De Nadai. 2021. Efficient Training of Visual Transformers with Small Datasets. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 23818–23830. https://proceedings.neurips.cc/paper/2021/hash/c81e155d85dae5430a8cee6f2242e82c-Abstract.html. (Cited on page 127).

Liu, Zhuang, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell & Saining Xie. 2022. A ConvNet for the 2020s. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022,* 11966–11976. IEEE. https://doi.org/10.1109/CVPR52688.2022.01167. (Cited on pages 74, 94, 100).

Lorraine, Jonathan, Paul Vicol & David Duvenaud. 2020. Optimizing Millions of Hyperparameters by Implicit Differentiation. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics,* edited by Silvia Chiappa & Roberto Calandra, 108: 1540–1552. Proceedings of Machine Learning Research. PMLR, 26–28 Aug. https://proceedings.mlr.press/v108/lorraine20a.html. (Cited on page 128).

Loshchilov, Ilya, & Frank Hutter. 2016. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. *CoRR* abs/1604.07269. arXiv: 1604.07269. http://arxiv.org/abs/1604.07269. (Cited on page 40).

Loshchilov, Ilya, & Frank Hutter. 2017. SGDR: Stochastic Gradient Descent with Warm Restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings.* OpenReview.net. https://openreview.net/forum?id=Skq89Scxx. (Cited on pages 9, 30, 60, 90, 95).

Lourenço, Helena R., Olivier C. Martin & Thomas Stützle. 2003. Iterated Local Search. In *Handbook of Metaheuristics,* 57: 320–353. International Series in Operations Research & Management Science. Kluwer / Springer. https://doi.org/10.1007/0-306-48056-5_11. (Cited on page 90).

Lu, Zhichao, Kalyanmoy Deb, Erik D. Goodman, Wolfgang Banzhaf & Vishnu Naresh Boddeti. 2020. NSGANetV2: Evolutionary Multi-objective Surrogate-Assisted Neural Architecture Search. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part I,* 12346: 35–51. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-030-58452-8_3. (Cited on pages 23, 24).

Lu, Zhichao, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb & Vishnu Naresh Boddeti. 2021. Neural Architecture Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (9): 2971–2989. (Cited on pages 16, 22–25).

Lu, Zhichao, Ian Whalen, Vishnu Boddeti, Yashesh D. Dhebar, Kalyanmoy Deb, Erik D. Goodman & Wolfgang Banzhaf. 2019. NSGA-Net: Neural Architecture Search Using Multi-Objective Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019,* 419–427. ACM. https://doi.org/10.1145/3321707.3321729. (Cited on pages 40, 41).

Luong, Ngoc Hoang, Marinus OW Grond, Han La Poutré & Peter A. N. Bosman. 2015. Scalable and Practical Multi-Objective Distribution Network Expansion Planning. *2015 IEEE Power & Energy Society General Meeting,* 1–5. (Cited on page 18).

Luong, Ngoc Hoang, Han La Poutré & Peter A. N. Bosman. 2014. Multi-Objective Gene-Pool Optimal Mixing Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014,* 357–364. ACM. https://doi.org/10.1145/2576768.2598261. (Cited on pages 18, 19).

Mallik, Neeratyoy, Edward Bergman, Carl Hvarfner, Danny Stoll, Maciej Janowski, Marius Lindauer, Luigi Nardi & Frank Hutter. 2023. PriorBand: Practical Hyperparameter Optimization in the Age of Deep Learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. http://papers.nips.cc/paper_files/paper/2023/hash/1704fe7aaff33a54802b83a016050ab8-Abstract-Conference.html. (Cited on pages 99, 129).

Martinez-Gutierrez, Juan Carlos, Youngran Kim, Sergio Salazar-Marioni, Muhammad Bilal Tariq, Rania Abdelkhaleq, Arash Niktabe, Anjan N. Ballekere, et al. 2023. Automated Large Vessel Occlusion Detection Software and Thrombectomy Treatment Times. *JAMA Neurology* 80, no. 11 (November): 1182–1190. ISSN: 2168-6149. https://doi.org/10.1001/jamaneurol.2023.3206. PMID: 37721738. (Cited on page 1).

McDermott, John. 1982. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19, no. 1 (September 1, 1982): 39–88. ISSN: 0004-3702. https://doi.org/10.1016/0004-3702(82)90021-2. (Cited on page 1).

McMahan, Brendan, Eider Moore, Daniel Ramage, Seth Hampson & Blaise Agüera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, 54: 1273–1282. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v54/mcmahan17a.html. (Cited on page 117).

Mellor, Joe, Jack Turner, Amos J. Storkey & Elliot J. Crowley. 2021. Neural Architecture Search without Training. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, 139: 7588–7598. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v139/mellor21a.html. (Cited on pages 38, 40).

Mirhoseini, Azalia, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, et al. 2021. A Graph Placement Methodology for Fast Chip Design. *Nature* 594, no. 7862 (June): 207–212. ISSN: 1476-4687. https://doi.org/10.1038/s41586-021-03544-w. (Cited on page 1).

Mitchell, Tom M. 1997. Machine Learning, International Edition. McGraw-Hill Series in Computer Science. McGraw-Hill. ISBN: 978-0-07-042807-2. https://www.worldcat.org/oclc/61321007. (Cited on page 3).

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama & Yuichi Yoshida. 2018. Spectral Normalization for Generative Adversarial Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=B1QRgziT-. (Cited on page 47).

Moritz, Philipp, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, et al. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018,* 561–577. USENIX Association. https://www.usenix.org/conference/osdi18/presentation/nishihara. (Cited on page 57).

Mosbach, Marius, Maksym Andriushchenko & Dietrich Klakow. 2021. On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net. https://openreview.net/forum?id=nzpLWnVAyah. (Cited on page 108).

Murphy, Kevin P. 2022. Probabilistic Machine Learning: An Introduction. MIT Press. http://probml.github.io/book1. (Cited on page 4).

Nair, Vinod, & Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel,* 807–814. Omnipress. https://icml.cc/Conferences/2010/papers/432.pdf. (Cited on page 5).

Narayanan, Ashwin Raaghav, Arber Zela, Tonmoy Saikia, Thomas Brox & Frank Hutter. 2021. Multi-headed Neural Ensemble Search. In *ICML Workshop on Uncertainty and Robustness in Deep Learning (UDL)*. http://lmbweb.informatik.uni-freiburg.de/Publications/2021/SB21a. (Cited on pages 16, 31).

Nesterov, Yurii E. 1983. A Method of Solving a Convex Programming Problem with Convergence Rate O(1/k²). *Soviet Mathematics Doklady* 27 (2): 372–376. http://www.mathnet.ru/php/archive.phtml?jrnid=dan&paperid=46009&option_lang=eng. (Cited on pages 3, 30).

Ng, Dianwen, Xiang Lan, Melissa Min-Szu Yao, Wing P. Chan & Mengling Feng. 2021. Federated Learning: A Collaborative Effort to Achieve Better Medical Imaging Models for Individual Sites That Have Small Labelled Datasets. *Quantitative Imaging in Medicine and Surgery* 11, no. 2 (February): 852–857. ISSN: 2223-4292. https://doi.org/10.21037/qims-20-595. (Cited on page 107).

Parker-Holder, Jack, Vu Nguyen, Shaan Desai & Stephen J. Roberts. 2021. Tuning Mixed Input Hyperparameters on the Fly for Efficient Population Based AutoRL. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 15513–15528. https://proceedings.neurips.cc/paper/2021/hash/82debd8a12b498e765a11a8e51159440-Abstract.html. (Cited on pages 10, 60, 64, 86, 89).

Parker-Holder, Jack, Vu Nguyen & Stephen J Roberts. 2020. Provably Efficient Online Hyperparameter Optimization with Population-Based Bandits. In *Advances in Neural Information Processing Systems*, 33: 17200–17211. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2020/file/c7af0926b294e47e52e46cfebe173f20-Paper.pdf. (Cited on pages 10, 60, 64–66, 68, 77, 86, 89).

Paul, Supratik, Vitaly Kurin & Shimon Whiteson. 2019. Fast Efficient Hyperparameter Tuning for Policy Gradient Methods. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 4618–4628. https://proceedings.neurips.cc/paper/2019/hash/743c41a921516b04afde48bb48e28ce6-Abstract.html. (Cited on page 63).

Petchrompo, Sanyapong, Anupong Wannakrairot & Ajith Kumar Parlikad. 2022. Pruning Pareto Optimal Solutions for Multi-Objective Portfolio Asset Management. *European Journal of Operational Research* 297 (1): 203–220. (Cited on page 18).

Pham, Hieu, Melody Guan, Barret Zoph, Quoc Le & Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *Proceedings of the 35th International Conference on Machine Learning*, 80: 4095–4104. Proceedings of Machine Learning Research. PMLR, October. https://proceedings.mlr.press/v80/pham18a.html. (Cited on pages 7, 14, 16, 38, 40).

Polyak, B. T. 1964. Some Methods of Speeding up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics* 4, no. 5 (January 1, 1964): 1–17. ISSN: 0041-5553. https://doi.org/10.1016/0041-5553(64)90137-5. (Cited on page 3).

Rasouli, Mohammad, Tao Sun & Ram Rajagopal. 2020. FedGAN: Federated Generative Adversarial Networks for Distributed Data. *arXiv:2006.07228 [cs, stat]*, June. https://doi.org/10.48550/arXiv.2006.07228. (Cited on page 107).

Ravuri, Suman, & Oriol Vinyals. 2019. Seeing is Not Necessarily Believing: Limitations of BigGANs for Data Augmentation. In *ICML Workshop on Learning from Limited Labeled Data (LLD)*. https://openreview.net/forum?id=rJMw747l_4. (Cited on page 109).

Real, Esteban, Alok Aggarwal, Yanping Huang & Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 4780–4789. AAAI Press. https://doi.org/10.1609/AAAI.V33I01.33014780. (Cited on pages 7, 14).

Real, Esteban, Chen Liang, David R. So & Quoc V. Le. 2020. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event,* 119: 8007–8019. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v119/real20a.html. (Cited on pages 7, 127).

Recht, Benjamin, Rebecca Roelofs, Ludwig Schmidt & Vaishaal Shankar. 2019. Do ImageNet Classifiers Generalize to ImageNet? In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA,* 97: 5389–5400. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v97/recht19a.html. (Cited on page 21).

Rieke, Nicola, Jonny Hancox, Wenqi Li, Fausto Milletarì, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, et al. 2020. The Future of Digital Health With Federated Learning. *Number: 1 Publisher: Nature Publishing Group, npj Digital Medicine* 3, no. 1 (September): 1–7. ISSN: 2398-6352. https://doi.org/10.1038/s41746-020-00323-1. (Cited on pages 10, 107).

Rodrigues, S, Pavol Bauer & Peter A. N. Bosman. 2016. Multi-Objective Optimization of Wind Farm Layouts–Complexity, Constraint Handling and Scalability. *Renewable and Sustainable Energy Reviews* 65: 587–609. (Cited on page 18).

Rokach, Lior. 2010. Ensemble-Based Classifiers. *Artificial Intelligence Review* 33 (1): 1–39. (Cited on page 14).

Ronneberger, Olaf, Philipp Fischer & Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015,* 234–241. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-319-24574-4. https://doi.org/10.1007/978-3-319-24574-4_28. (Cited on pages 106, 108).

Rosenblatt, Frank. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological review* 65 (6): 386. (Cited on page 5).

Russakovsky, Olga, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision* 115 (3): 211–252. (Cited on page 20).

Salimans, Tim, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen & Xi Chen. 2016. Improved Techniques for Training GANs. In *Advances in Neural Information Processing Systems,* edited by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon & R. Garnett, vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/8a3363abe792db2d8761d6403605aeb7-Paper.pdf. (Cited on page 46).

Sandler, Mark, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov & Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018,* 4510–4520. Computer Vision Foundation / IEEE Computer Society. https://doi.org/10.1109/CVPR.2018.00474. (Cited on pages 7, 21).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford & Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs],* August. https://doi.org/10.48550/arXiv.1707.06347. (Cited on pages 67, 78, 94, 100).

Shala, Gresa, Sebastian Pineda-Arango, André Biedenkapp, Frank Hutter & Josif Grabocka. 2024. HPO-RL-Bench: A Zero-Cost Benchmark for HPO in Reinforcement Learning. In *International Conference on Automated Machine Learning, 9-12 September 2024, Sorbonne Université, Paris, France,* 18/1–31. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v256/shala24a.html. (Cited on page 129).

Sharma, Rishab, Rahul Deora & Anirudha Vishvakarma. 2020. AlphaNet: An Attention Guided Deep Network for Automatic Image Matting. In *2020 International Conference on Omni-layer Intelligent Systems, COINS 2020, Barcelona, Spain, August 31 - September 2, 2020,* 1–8. IEEE. https://doi.org/10.1109/COINS49042.2020.9191371. (Cited on pages 14, 23).

Shim, Jae-hun, Kyeongbo Kong & Suk-Ju Kang. 2021. Core-set Sampling for Efficient Neural Architecture Search. arXiv: 2107.06869 [cs.LG]. https://arxiv.org/abs/2107.06869. (Cited on pages 38, 40).

Shu, Yao, Yizhou Chen, Zhongxiang Dai & Bryan Kian Hsiang Low. 2021. Going Beyond Neural Architecture Search with Sampling-based Neural Ensemble Search. *CoRR* abs/2109.02533. arXiv: 2109.02533. https://arxiv.org/abs/2109.02533. (Cited on pages 16, 31).

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, et al. 2018. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. *Science* 362, no. 6419 (December 7, 2018): 1140–1144. https://doi.org/10.1126/science.aar6404. (Cited on page 1).

Singh, Gulshan, & Kalyanmoy Deb. 2006. Comparison of Multi-Modal Optimization Algorithms Based on Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings, Seattle, Washington, USA, July 8-12, 2006,* 1305–1312. ACM. https://doi.org/10.1145/1143997.1144200. (Cited on pages 3, 15).

Sinha, Samarth, Homanga Bharadhwaj, Aravind Srinivas & Animesh Garg. 2020. D2RL: Deep Dense Architectures in Reinforcement Learning. *CoRR* abs/2010.09163. arXiv: 2010.09163. https://arxiv.org/abs/2010.09163. (Cited on page 44).

Snoek, Jasper, Hugo Larochelle & Ryan P. Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States,* 2960–2968. https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html. (Cited on pages 3, 8, 9, 88).

So, David R., Quoc V. Le & Chen Liang. 2019. The Evolved Transformer. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA,* 97: 5877–5886. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v97/so19a.html. (Cited on page 127).

Sokar, Ghada, Rishabh Agarwal, Pablo Samuel Castro & Utku Evci. 2023. The Dormant Neuron Phenomenon in Deep Reinforcement Learning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA,* 202: 32145–32168. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v202/sokar23a.html. (Cited on page 90).

Song, Xingyou, Qiuyi Zhang, Chansoo Lee, Emily Fertig, Tzu-Kuo Huang, Lior Belenki, Greg Kochanski, et al. 2024. The Vizier Gaussian Process Bandit Algorithm. *CoRR* abs/2408.11527. https://doi.org/10.48550/ARXIV.2408.11527. arXiv: 2408.11527. (Cited on page 3).

Stamoulis, Dimitrios, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu & Diana Marculescu. 2019. Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II,* 11907: 481–497. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-030-46147-8_29. (Cited on page 14).

Stanford CS231N. 2017. Tiny ImageNet. http://cs231n.stanford.edu/tiny-imagenet-200.zip. (Cited on pages 61, 74, 78, 86, 94, 100).

Streeter, Matthew. 2018. Approximation Algorithms for Cascading Prediction Models. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018,* 80: 4759–4767. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v80/streeter18a.html. (Cited on pages 17, 21, 23–25).

Sun, Chen, Abhinav Shrivastava, Saurabh Singh & Abhinav Gupta. 2017. Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017,* 843–852. IEEE Computer Society. https://doi.org/10.1109/ICCV.2017.97. (Cited on page 10).

Sun, Li, Junxiang Chen, Yanwu Xu, Mingming Gong, Ke Yu & Kayhan Batmanghelich. 2022. Hierarchical Amortized Training for Memory-efficient High Resolution 3D GAN. *arXiv:2008.01910 [cs, eess], IEEE Journal of Biomedical and Health Informatics* 26, no. 8 (August): 3966–3975. ISSN: 2168-2194, 2168-2208. https://doi.org/10.1109/JBHI.2022.3172976. (Cited on page 107).

Syswerda, Gilbert, et al. 1989. Uniform Crossover in Genetic Algorithms. *ICGA* 3: 2–9. (Cited on page 44).

Tan, Jia Mian, Haoran Liao, Wei Liu, Changjun Fan, Jincai Huang, Zhong Liu, Junchi Yan, et al. 2024. Hyperparameter Optimization: Classics, Acceleration, Online, Multi-Objective, and Tools. *Mathematical Biosciences and Engineering* 21 (6): 6289–6335. ISSN: 1551-0018. https://doi.org/10.3934/mbe.2024275. (Cited on pages 62, 86).

Tan, Mingxing, & Quoc V. Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 97: 6105–6114. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v97/tan19a.html. (Cited on pages 23–25).

Tan, Mingxing, & Quoc V. Le. 2021. EfficientNetV2: Smaller Models and Faster Training. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event,* 139: 10096–10106. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v139/tan21a.html. (Cited on pages 9, 60, 90).

Tang, Yunhao, & Krzysztof Choromanski. 2020. Online Hyper-parameter Tuning in Off-policy Learning via Evolutionary Strategies. *arXiv:2006.07554 [cs, stat],* June. https://doi.org/10.48550/arXiv.2006.07554. (Cited on page 63).

Tassa, Yuval, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, et al. 2018. DeepMind Control Suite. *CoRR* abs/1801.00690. arXiv: 1801.00690. http://arxiv.org/abs/1801.00690. (Cited on page 46).

Thambawita, Vajira, Pegah Salehi, Sajad Amouei Sheshkal, Steven A. Hicks, Hugo L. Hammer, Sravanthi Parasa, Thomas de Lange, Pål Halvorsen & Michael A. Riegler. 2022. SinGAN-Seg: Synthetic Training Data Generation for Medical Image Segmentation. *Publisher: Public Library of Science, PLOS ONE* 17, no. 5 (May): e0267976. ISSN: 1932-6203. https://doi.org/10.1371/journal.pone.0267976. (Cited on page 107).

The Article 29 Working Party on Data Protection. 2014. Opinion 05/2014 on Anonymisation Techniques. (Cited on page 126).

Thierens, Dirk, & Peter A. N. Bosman. 2011. Optimal Mixing Evolutionary Algorithms. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011,* 617–624. ACM. https://doi.org/10.1145/2001576.2001661. (Cited on pages 5, 18).

Uriot, Thomas, & Dario Izzo. 2020. Safe Crossover of Neural Networks Through Neuron Alignment. In *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020,* 435–443. ACM. https://doi.org/10.1145/3377930.3390197. (Cited on page 41).

Van Veldhuizen, David A, & Gary B Lamont. 1998. Multiobjective Evolutionary Algorithm Research: A History and Analysis. AFIT Technical Report TR-98-03. Air Force Institute of Technology, Dept. of Electrical and Computer Engineering. (Cited on pages 3, 15).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems,* vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf. (Cited on page 127).

Viola, Paul A., & Michael J. Jones. 2001. Rapid Object Detection using a Boosted Cascade of Simple Features. In *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), with CD-ROM, 8-14 December 2001, Kauai, HI, USA,* I:511–518. IEEE Computer Society. https://doi.org/10.1109/CVPR.2001.990517. (Cited on page 14).

Wan, Xingchen, Cong Lu, Jack Parker-Holder, Philip J. Ball, Vu Nguyen, Binxin Ru & Michael A. Osborne. 2022. Bayesian Generational Population-Based Training. In *International Conference on Automated Machine Learning, AutoML 2022, 25-27 July 2022, Johns Hopkins University, Baltimore, MD, USA,* 188: 14/1–27. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v188/wan22a.html. (Cited on pages 9, 10, 38, 41, 47, 60, 61, 64, 67, 78, 80, 86, 89, 91, 104).

Wan, Xingchen, Vu Nguyen, Huong Ha, Bin Xin Ru, Cong Lu & Michael A. Osborne. 2021. Think Global and Act Local: Bayesian Optimisation over High-Dimensional Categorical and Mixed Search Spaces. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event,* 139: 10663–10674. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v139/wan21b.html. (Cited on pages 64, 89).

Wang, Dilin, Chengyue Gong, Meng Li, Qiang Liu & Vikas Chandra. 2021a. AlphaNet: Improved Training of Supernets with Alpha-Divergence. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event,* 139: 10760–10771. Proceedings of Machine Learning Research. PMLR. http://proceedings.mlr.press/v139/wang21i.html. (Cited on pages 16, 21, 40).

Wang, Dilin, Meng Li, Chengyue Gong & Vikas Chandra. 2021b. AttentiveNAS: Improving Neural Architecture Search via Attentive Sampling. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, Virtual Event, June 19-25, 2021,* 6418–6427. Computer Vision Foundation / IEEE. https://doi.org/10.1109/CVPR46437.2021.00635. (Cited on pages 7, 8, 14, 16, 21, 38, 40).

Wang, Jinbao, Guoyang Xie, Yawen Huang, Jiayi Lyu, Feng Zheng, Yefeng Zheng & Yaochu Jin. 2023. FedMed-GAN: Federated Domain Translation on Unsupervised Cross-Modality Brain Image Synthesis. *Neurocomputing* 546 (August): 126282. ISSN: 0925-2312. https://doi.org/10.1016/j.neucom.2023.126282. (Cited on page 107).

Wang, Xiaofang, Dan Kondratyuk, Kris M. Kitani, Yair Movshovitz-Attias & Elad Eban. 2020. Multiple Networks are More Efficient than One: Fast and Accurate Models via Ensembles and Cascades. *CoRR* abs/2012.01988. arXiv: 2012.01988. https://arxiv.org/abs/2012.01988. (Cited on pages 15, 17, 23, 26, 28, 29).

Wei, Tao, Changhu Wang, Yong Rui & Chang Wen Chen. 2016. Network Morphism. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, 48: 564–572. JMLR Workshop and Conference Proceedings. JMLR.org. http://proceedings.mlr.press/v48/wei16.html. (Cited on page 40).

Wen, Wei, Feng Yan, Yiran Chen & Hai Li. 2020. AutoGrow: Automatic Layer Growing in Deep Convolutional Networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 833–841. ACM. https://doi.org/10.1145/3394486.3403126. (Cited on page 40).

White, Colin, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey & Frank Hutter. 2023. Neural Architecture Search: Insights from 1000 Papers. *CoRR* abs/2301.08727. https://doi.org/10.48550/ARXIV.2301.08727. arXiv: 2301.08727. (Cited on page 127).

White, Colin, Arber Zela, Robin Ru, Yang Liu & Frank Hutter. 2021. How Powerful are Performance Predictors in Neural Architecture Search? In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event*, 28454–28469. https://proceedings.neurips.cc/paper/2021/hash/ef575e8837d065a1683c022d2077d342-Abstract.html. (Cited on pages 7, 14).

Wichard, J. D., C. Merkwirth & M. Ogorzalek. 2002. Building Ensembles with Heterogeneous Models. In *Proceedings of the 7th Course of the International School on Neural Nets*. Salerno, Italy: IIASS. (Cited on page 14).

Wightman, Ross. 2019. PyTorch Image Models. https://github.com/rwightman/pytorch-image-models. https://doi.org/10.5281/zenodo.4414861. (Cited on page 26).

Wilcoxon, Frank. 1992. Individual Comparisons by Ranking Methods. In *Breakthroughs in Statistics: Methodology and Distribution*, 196–202. New York, NY: Springer. ISBN: 978-1-4612-4380-9. https://doi.org/10.1007/978-1-4612-4380-9_16. (Cited on pages 21, 45, 55, 81, 114).

Won, Jonghyeon, Hyun-Suk Lee & Jang-Won Lee. 2025. A Review on Multi-fidelity Hyperparameter Optimization in Machine Learning. *ICT Express* 11, no. 2 (April): 245–257. ISSN: 2405-9595. https://doi.org/10.1016/j.icte.2025.02.008. (Cited on pages 9, 88).

Woodland, McKell, John Wood, Brian M. Anderson, Suprateek Kundu, Ethan Lin, Eugene Koay, Bruno Odisio, et al. 2022. Evaluating the Performance of StyleGAN2-ADA on Medical Images. In *Simulation and Synthesis in Medical Imaging*, 142–153. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-031-16980-9. https://doi.org/10.1007/978-3-031-16980-9_14. (Cited on page 106).

Wortsman, Mitchell, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. 2022. Model Soups: Averaging Weights of Multiple Fine-Tuned Models Improves Accuracy Without Increasing Inference Time. *International Conference on Machine Learning*, 23965–23998. (Cited on pages 41, 50, 51).

Xiao, Han, Kashif Rasul & Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*, September. https://doi.org/10.48550/arXiv.1708.07747. (Cited on pages 61, 67, 78, 86, 94, 100).

Xu, Yuhui, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian & Hongkai Xiong. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=BJlS634tPr. (Cited on page 40).

Xu, Zhixiang, Matt J Kusner, Kilian Q Weinberger, Minmin Chen & Olivier Chapelle. 2014. Classifier Cascades and Trees for Minimizing Feature Evaluation Cost. *The Journal of Machine Learning Research* 15 (1): 2113–2144. (Cited on page 17).

Yadan, Omry. 2019. Hydra - A Framework for Elegantly Configuring Complex Applications. Github. https://github.com/facebookresearch/hydra. (Cited on page 128).

Yang, An, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, et al. 2024. Qwen2 Technical Report, September 10, 2024. https://doi.org/10.48550/arXiv.2407.10671. arXiv: 2407.10671 [cs]. (Cited on page 127).

Yang, Antoine, Pedro M. Esperança & Fabio Maria Carlucci. 2020. NAS Evaluation Is Frustratingly Hard. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=HygrdpVKvr. (Cited on page 51).

Yang, Ge, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen & Jianfeng Gao. 2021. Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 17084–17097. https://proceedings.neurips.cc/paper/2021/hash/8df7c2e3c3c3be098ef7b382bd2c37ba-Abstract.html. (Cited on page 130).

Yarats, Denis, Rob Fergus, Alessandro Lazaric & Lerrel Pinto. 2022. Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=_SJ-_yyes8. (Cited on page 46).

Yi, Xin, Ekta Walia & Paul Babyn. 2019. Generative Adversarial Network in Medical Imaging: A Review. *Medical Image Analysis* 58 (December): 101552. ISSN: 1361-8415. https://doi.org/10.1016/j.media.2019.101552. (Cited on pages 106, 107).

Yu, Jiahui, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas S. Huang, Xiaodan Song, Ruoming Pang & Quoc Le. 2020. BigNAS: Scaling up Neural Architecture Search with Big Single-Stage Models. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part VII,* 12352: 702–717. Lecture Notes in Computer Science. Springer. https://doi.org/10.1007/978-3-030-58571-6_41. (Cited on page 23).

Yu, Kaicheng, Christian Sciuto, Martin Jaggi, Claudiu Musat & Mathieu Salzmann. 2020. Evaluating The Search Phase of Neural Architecture Search. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=H1loF2NFwr. (Cited on page 39).

Yu, Rosen Ting-Ying, Cyril Picard & Faez Ahmed. 2025. GIT-BO: High-Dimensional Bayesian Optimization with Tabular Foundation Models. *1st ICML Workshop on Foundation Models for Structured Data, 2025* (June). https://openreview.net/forum?id=oJEAh2Jdgp. (Cited on page 99).

Yun, Sangdoo, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo & Junsuk Choe. 2019. CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019,* 6022–6031. IEEE. https://doi.org/10.1109/ICCV.2019.00612. (Cited on page 30).

Zahavy, Tom, Zhongwen Xu, Vivek Veeriah, Matteo Hessel, Junhyuk Oh, Hado P van Hasselt, David Silver & Satinder Singh. 2020. A Self-Tuning Actor-Critic Algorithm. *Advances in Neural Information Processing Systems* 33: 20913–20924. https://proceedings.neurips.cc/paper_files/paper/2020/hash/f02208a057804ee16ac72ff4d3cec53b-Abstract.html. (Cited on page 63).

Zaidi, Sheheryar, Tudor Berariu, Hyunjik Kim, Jörg Bornschein, Claudia Clopath, Yee Whye Teh & Razvan Pascanu. 2022. When Does Re-initialization Work? In *Proceedings on "I Can't Believe It's Not Better! - Understanding Deep Learning Through Empirical Falsification" at NeurIPS 2022 Workshops, 03 December 2022, New Orleans, Louisiana, USA,* 187: 12–26. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v187/zaidi23a.html. (Cited on pages 50, 90).

Zaidi, Sheheryar, Arber Zela, Thomas Elsken, Chris C. Holmes, Frank Hutter & Yee Whye Teh. 2021. Neural Ensemble Search for Uncertainty Estimation and Dataset Shift. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual Event,* 7898–7911. https://proceedings.neurips.cc/paper/2021/hash/41a6fd31aa2e75c3c6d427db3d17ea80-Abstract.html. (Cited on pages 14, 16, 31).

Zhao, Shengyu, Zhijian Liu, Ji Lin, Jun-Yan Zhu & Song Han. 2020. Differentiable Augmentation for Data-Efficient GAN Training. *Advances in Neural Information Processing Systems* 33: 7559–7570. https://proceedings.neurips.cc/paper/2020/hash/55479c55ebd1efd3ff125f1337100388-Abstract.html. (Cited on page 109).

Zhu, Ligeng, Zhijian Liu & Song Han. 2019. Deep Leakage from Gradients. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada,* 14747–14756. https://proceedings.neurips.cc/paper/2019/hash/60a6c4002cc7b29142def8871531281a-Abstract.html. (Cited on pages 10, 106, 107).

Zitzler, Eckart, Lothar Thiele, Marco Laumanns, Carlos M Fonseca & Viviane Grunert Da Fonseca. 2003. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* 7 (2): 117–132. (Cited on pages 21, 34).

Zoph, Barret, & Quoc V. Le. 2017. Neural Architecture Search with Reinforcement Learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=r1Ue8Hcxg. (Cited on pages 7, 14, 16, 38, 39).

Zoph, Barret, Vijay Vasudevan, Jonathon Shlens & Quoc V. Le. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 8697–8710. Computer Vision Foundation / IEEE Computer Society. https://doi.org/10.1109/CVPR.2018.00907. (Cited on page 7).

Curriculum Vitæ

Aleksandr Chebykin

23-04-1997	Born in St. Petersburg, Russia.

Education

2004 – 2014	Primary and secondary education (graduated with distinction) Gymnasium 261, St. Petersburg, Russia
2014 – 2018	Bachelor in Software Engineering (graduated with distinction) Saint Petersburg State University, St. Petersburg, Russia *Thesis:* Source Code Generation using Machine Learning Techniques
2018 – 2020	Master in Computer Science (one of the three best graduates) Technical University of Berlin, Berlin, Germany *Thesis:* Learning Neural Network Structure that Aids the Interpretability of the Underlying Processes
2021 –	PhD candidate Dutch National Research Institute for Mathematics and Computer Science, Amsterdam, the Netherlands

Experience

2015, 2016	Software Engineer, Intern Lanit-Terkom, St. Petersburg, Russia
2017	Data Scientist, Intern JetBrains, St. Petersburg, Russia
2019 – 2020	Research Assistant Technical University of Berlin, Berlin, Germany
2024 – 2025	Machine Learning Scientist, Intern InSilicoTrials, 's-Hertogenbosch, the Netherlands

Awards

2022	Best Paper award in the Neuroevolution track of GECCO 2022

List of Publications

9. Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2025a. Iterated Population Based Training with Task-Agnostic Restarts. *Accepted for publication in the proceedings of the 43rd International Conference on Machine Learning (ICML 2026).* arXiv: 2511.09190. https://arxiv.org/abs/2511.09190

8. Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2025b. To Be Greedy, or Not to Be - That Is the Question for Population Based Training Variants. *Transactions on Machine Learning Research,* ISSN: 2835-8856. https://openreview.net/forum?id=3qmnxysNbi

7. Giuseppe Pasculli, Marco Virgolin, Puja Myles, Anna Vidovszky, Charles Fisher, Elisabetta Biasin, Miranda Mourby, Francesco Pappalardo, Saverio D'Amico, Mario Torchia, Alexander Chebykin, Vincenzo Carbone, Luca Emili & Daniel Roeshammar. 2025. Synthetic data in healthcare and drug development: Definitions, regulatory frameworks, issues. *CPT: Pharmacometrics & Systems Pharmacology* 14 (5): 840–852. https://doi.org/10.1002/psp4.70021

6. Alexander Chebykin, Peter A. N. Bosman & Tanja Alderliesten. 2024. Hyperparameter-Free Medical Image Synthesis for Sharing Data and Improving Site-Specific Segmentation. In *Medical Imaging with Deep Learning, 3-5 July 2024, Paris, France,* 250: 199–219. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v250/chebykin24a.html

5. Alexander Chebykin, Arkadiy Dushatskiy, Tanja Alderliesten & Peter A. N. Bosman. 2023. Shrink-Perturb Improves Architecture Mixing During Population Based Training for Neural Architecture Search. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland,* 372: 381–388. Frontiers in Artificial Intelligence and Applications. IOS Press. https://doi.org/10.3233/FAIA230294

4. Arkadiy Dushatskiy, Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2023. Multi-Objective Population Based Training. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA,* 202: 8969–8989. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v202/dushatskiy23a.html

3. Alexander Chebykin, Tanja Alderliesten & Peter A. N. Bosman. 2022. Evolutionary Neural Cascade Search Across Supernetworks. In *GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, July 9 - 13, 2022,* 1038–1047. ACM. https://doi.org/10.1145/3512290.3528749

2. Alexander Chebykin & Iakov Kirilenko. 2018. Applying Deep Learning to C# Call Sequence Synthesis. *Proceedings of ISP RAS* 30 (3): 63–86. https://doi.org/10.15514/ISPRAS-2018-30(3)-5

1. Alexander Chebykin, Mikhail Kita & Iakov Kirilenko. 2017. DeepAPI#: CLR/C# Call Sequence Synthesis from Text Query. In *SEIM 2017: Proceedings of the Second Conference on Software*

Engineering and Information Management, 1864: 6–11. CEUR Workshop Proceedings. CEUR-WS.org. https://ceur-ws.org/Vol-1864/paper_5.pdf

SIKS dissertation series

2016
01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT

30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology

11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TU/e), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech
30 Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web

2019 01 Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
02 Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
03 Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04 Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
05 Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
06 Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07 Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
08 Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
09 Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems

10 Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
11 Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12 Jacqueline Heinerman (VUA), Better Together
13 Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
14 Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
15 Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
16 Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17 Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
18 Gerard Wagenaar (UU), Artefacts in Agile Team Communication
19 Vincent Koeman (TUD), Tools for Developing Cognitive Agents
20 Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
21 Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
22 Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
23 Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
24 Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
25 Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
26 Prince Singh (UT), An Integration Platform for Synchromodal Transport
27 Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
28 Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
29 Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
30 Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
31 Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
32 Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games
33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming

36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data Augmentation-Methods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining

26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context
28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
31 Gongjin Lan (VUA), Learning better – From Baby to Better
32 Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production

2021 01 Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
02 Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
03 Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
07 Armel Lefebvre (UU), Research data management for open science
08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks

19 Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (UL), Robust rules for prediction and description
24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing
25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research

17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences
27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment

07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity
23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis

04 Mike Huisman (UL), Understanding Deep Meta-Learning
05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
11 withdrawn
12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
20 Ritsart Anne Plantenga (UL), Omgang met Regels
21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning
28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting

31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining
33 Siddharth Mehrotra (TUD), Designing for Appropriate Trust in Human-AI interaction
34 Robert Deckers (VUA), From Smallest Software Particle to System Specification - MuDForM: Multi-Domain Formalization Method
35 Sicui Zhang (TU/e), Methods of Detecting Clinical Deviations with Process Mining: a fuzzy set approach
36 Thomas Mulder (TU/e), Optimization of Recursive Queries on Graphs
37 James Graham Nevin (UvA), The Ramifications of Data Handling for Computational Models
38 Christos Koutras (TUD), Tabular Schema Matching for Modern Settings
39 Paola Lara Machado (TU/e), The Nexus between Business Models and Operating Models: From Conceptual Understanding to Actionable Guidance
40 Montijn van de Ven (TU/e), Guiding the Definition of Key Performance Indicators for Business Models
41 Georgios Siachamis (TUD), Adaptivity for Streaming Dataflow Engines
42 Emmeke Veltmeijer (VUA), Small Groups, Big Insights: Understanding the Crowd through Expressive Subgroup Analysis
43 Cedric Waterschoot (KNAW Meertens Instituut), The Constructive Conundrum: Computational Approaches to Facilitate Constructive Commenting on Online News Platforms
44 Marcel Schmitz (OU), Towards learning analytics-supported learning design
45 Sara Salimzadeh (TUD), Living in the Age of AI: Understanding Contextual Factors that Shape Human-AI Decision-Making
46 Georgios Stathis (Leiden University), Preventing Disputes: Preventive Logic, Law & Technology
47 Daniel Daza (VUA), Exploiting Subgraphs and Attributes for Representation Learning on Knowledge Graphs
48 Ioannis Petros Samiotis (TUD), Crowd-Assisted Annotation of Classical Music Compositions

---

2025 01 Max van Haastrecht (UL), Transdisciplinary Perspectives on Validity: Bridging the Gap Between Design and Implementation for Technology-Enhanced Learning Systems
02 Jurgen van den Hoogen (JADS), Time Series Analysis Using Convolutional Neural Networks
03 Andra-Denis Ionescu (TUD), Feature Discovery for Data-Centric AI
04 Rianne Schouten (TU/e), Exceptional Model Mining for Hierarchical Data
05 Nele Albers (TUD), Psychology-Informed Reinforcement Learning for Situated Virtual Coaching in Smoking Cessation
06 Daniël Vos (TUD), Decision Tree Learning: Algorithms for Robust Prediction and Policy Optimization
07 Ricky Maulana Fajri (TU/e), Towards Safer Active Learning: Dealing with Unwanted Biases, Graph-Structured Data, Adversary, and Data Imbalance

08 Stefan Bloemheuvel (TiU), Spatio-Temporal Analysis Through Graphs: Predictive Modeling and Graph Construction
09 Fadime Kaya (VUA), Decentralized Governance Design - A Model-Based Approach
10 Zhao Yang (UL), Enhancing Autonomy and Efficiency in Goal-Conditioned Reinforcement Learning
11 Shahin Sharifi Noorian (TUD), From Recognition to Understanding: Enriching Visual Models Through Multi-Modal Semantic Integration
12 Lijun Lyu (TUD), Interpretability in Neural Information Retrieval
13 Fuda van Diggelen (VUA), Robots Need Some Education: on the complexity of learning in evolutionary robotics
14 Gennaro Gala (TU/e), Probabilistic Generative Modeling with Latent Variable Hierarchies
15 Michiel van der Meer (UL), Opinion Diversity through Hybrid Intelligence
16 Monika Grewal (TU Delft), Deep Learning for Landmark Detection, Segmentation, and Multi-Objective Deformable Registration in Medical Imaging
17 Matteo De Carlo (VUA), Real Robot Reproduction: Towards Evolving Robotic Ecosystems
18 Anouk Neerincx (UU), Robots That Care: How Social Robots Can Boost Children's Mental Wellbeing
19 Fang Hou (UU), Trust in Software Ecosystems
20 Alexander Melchior (UU), Modelling for Policy is More Than Policy Modelling (The Useful Application of Agent-Based Modelling in Complex Policy Processes)
21 Mandani Ntekouli (UM), Bridging Individual and Group Perspectives in Psychopathology: Computational Modeling Approaches using Ecological Momentary Assessment Data
22 Hilde Weerts (TU/e), Decoding Algorithmic Fairness: Towards Interdisciplinary Understanding of Fairness and Discrimination in Algorithmic Decision-Making
23 Roderick van der Weerdt (VUA), IoT Measurement Knowledge Graphs: Constructing, Working and Learning with IoT Measurement Data as a Knowledge Graph
24 Zhong Li (UL), Trustworthy Anomaly Detection for Smart Manufacturing
25 Kyana van Eijndhoven (TiU), A Breakdown of Breakdowns: Multi-Level Team Coordination Dynamics under Stressful Conditions
26 Tom Pepels (UM), Monte-Carlo Tree Search is Work in Progress
27 Danil Provodin (JADS, TU/e), Sequential Decision Making Under Complex Feedback
28 Jinke He (TU Delft), Exploring Learned Abstract Models for Efficient Planning and Learning
29 Erik van Haeringen (VUA), Mixed Feelings: Simulating Emotion Contagion in Groups
30 Myrthe Reuver (VUA), A Puzzle of Perspectives: Interdisciplinary Language Technology for Responsible News Recommendation

31 Gebrekirstos Gebreselassie Gebremeskel (RUN), Spotlight on Recommender Systems: Contributions to Selected Components in the Recommendation Pipeline
32 Ryan Brate (UU), Words Matter: A Computational Toolkit for Charged Terms
33 Merle Reimann (VUA), Speaking the Same Language: Spoken Capability Communication in Human-Agent and Human-Robot Interaction
34 Eduard C. Groen (UU), Crowd-Based Requirements Engineering
35 Urja Khurana (VUA), From Concept To Impact: Toward More Robust Language Model Deployment
36 Anna Maria Wegmann (UU), Say the Same but Differently: Computational Approaches to Stylistic Variation and Paraphrasing
37 Chris Kamphuis (RUN), Exploring Relations and Graphs for Information Retrieval
38 Valentina Maccatrozzo (VUA), Break the Bubble: Semantic Patterns for Serendipity
39 Dimitrios Alivanistos (VUA), Knowledge Graphs & Transformers for Hypothesis Generation: Accelerating Scientific Discovery in the Era of Artificial Intelligence
40 Stefan Grafberger (UvA), Declarative Machine Learning Pipeline Management via Logical Query Plans
41 Mozhgan Vazifehdoostirani (TU/e), Leveraging Process Flexibility to Improve Process Outcome - From Descriptive Analytics to Actionable Insights
42 Margherita Martorana (VUA), Semantic Interpretation of Dataless Tables: a metadata-driven approach for findable, accessible, interoperable and reusable restricted access data
43 Krist Shingjergji (OU), Sense the Classroom - Using AI to Detect and Respond to Learning-Centered Affective States in Online Education
44 Robbert Reijnen (TU/e), Dynamic Algorithm Configuration for Machine Scheduling Using Deep Reinforcement Learning
45 Anjana Mohandas Sheeladevi (VUA), Occupant-Centric Energy Management: Balancing Privacy, Well-being and Sustainability in Smart Buildings
46 Ya Song (TU/e), Graph Neural Networks for Modeling Temporal and Spatial Dimensions in Industrial Decision-making
47 Tom Kouwenhoven (UL), Collaborative Meaning-Making. The Emergence of Novel Languages in Humans, Machines, and Human-Machine Interactions
48 Evy van Weelden (TiU), Integrating Virtual Reality and Neurophysiology in Flight Training
49 Selene Báez Santamaría (VUA), Knowledge-centered conversational agents with a drive to learn
50 Lea Krause (VUA), Contextualising Conversational AI
51 Jiaxu Zhao (TU/e), Understanding and Mitigating Unwanted Biases in Generative Language Models
52 Qiao Xiao (TU/e), Model, Data and Communication Sparsity for Efficient Training of Neural Networks
53 Gaole He (TUD), Towards Effective Human-AI Collaboration: Promoting Appropriate Reliance on AI Systems

54 Go Sugimoto (VUA), MISSING LINKS Investigating the Quality of Linked Data and its Tools in Cultural Heritage and Digital Humanities
55 Sietze Kai Kuilman (TUD), AI that Glitters is Not Gold: Requirements for Meaningful Control of AI Systems
56 Wijnand van Woerkom (UU), A Fortiori Case-Based Reasoning: Formal Studies with Applications in Artificial Intelligence and Law
57 Syeda Amna Sohail (UT), Privacy-Utility Trade-Off in Healthcare Metadata Sharing and Beyond: A Normative and Empirical Evaluation at Inter and Intra Organizational Levels
58 Junhan Wen (TUD), "From iMage to Market": Machine-Learning-Empowered Fruit Supply
59 Mohsen Abbaspour Onari (TU/e), From Explanation to Trust: Modeling and Measuring Trust in Explainable Decision Support
60 Marcel Jurriaan Robeer (UU), Beyond Trust: A Causal Approach to Explainable AI in Law Enforcement
61 Shuai Wang (VUA), Links in Large Integrated Knowledge Graphs: Analysis, Refinement, and Domain Applications
62 Khaleel Asyraaf Mat Sanusi (OU), Augmenting a learning model within immersive learning environments for psychomotor skills
63 Rashid Zaman (TU/e), Online Conformance Checking on Degraded Data
64 Jens d'Hondt (TU/e), Effective and Efficient Multivariate Similarity Search
65 Aswin Balasubramaniam (UT), Disentangling Runner Drone Interaction Potentialities

---

2026 01 Pei-Yu Chen (TUD), Human-Agent Alignment Dialogues: Eliciting User Information at Runtime for Personalized Behavior Support
02 Hezha Hassan Mohammedkhan (TiU), Estimating Body Measurements of Children from 2D Images: Towards the Automatic Detection of Malnutrition
03 Kyriakos Psarakis (TUD), Democratizing Scalable Cloud Applications: Transactional Stateful Functions on Streaming Dataflows
04 Boyu Xu (UU), Exploring Indirect Relations Between Topics in Neuroscience Literature Using Augmented Reality to Inform Experimental Design
05 Koen Minartz (TU/e), Stochastic Simulation with Geometric Deep Generative Models
06 Azim Afroozeh (CWI, VUA), FastLanes: A Next-Gen File Format
07 Inès Blin (VUA), Narrative Understanding with Knowledge Graphs
08 Paul van Vulpen (UU), Debating Digital Dominance: Decentralized Technology Governance For Strategic Autonomy
09 Afrizal Doewes (TU/e), Rethinking Automated Essay Scoring: Agreement, Fairness, and Feedback
10 Nikolaos Delapaschos Kondylidis (VUA), Establishing Task-Oriented Understanding between Agents
11 Işıl Baysal Erez (UT), Handling Missing Data with Meta-Learning and Large Language Models
12 Xue Li (UvA), From Fine-tuning to Prompting: A Paradigm Shift in Knowledge Graph Construction

13 Isaac da Silva Torres (VUA), Guidelines To Flux Between Conceptual Models: Understanding Complex Digital Business Ecosystems
14 Philip Lippmann (TUD), Synthetic Data for Robust Language Modelling
15 Rashmi Khazanchi (OU), Artificial Intelligence in Education: Impact of AI-Based Systems on Mathematics Achievement
16 Carolina Ferreira Gomes Centeio Jorge (TUD), Modelling Artificial Trust for Effective Human-AI Teamwork
17 Maria Tsfasman (TUD), Towards Predicting Memory in Multimodal Group Interactions
18 Riccardo Lo Bianco (TU/e), Deep Reinforcement Learning for Automated Decision-Making in Process Management Systems
19 Israel Campero Jurado (TU/e), Innovations in Optimization and Applications in Healthcare
20 Iftitahu Ni'mah (TU/e), Contrastive Learning and Evaluation in Low Resource Scenario of Natural Language Processing
21 Francisco N.F.Q. Simoes (UU), Causality, Information, and Decision-Making
22 Ruben Verhagen (TUD), Transparent and Explainable Agents for Human-Agent Teaming
23 Eduardo Calò (UU), Automatically Expressing the Meaning of Logical Formulae in Natural Language
24 Shuai Han (UU), Improving Sample Efficiency of Reinforcement Learning: Exploiting Structural Knowledge for Decision Making
25 Francis Saa-Dittoh (VUA), From Radio to AI: African Community-Driven Development of Sustainable Information Systems
26 Mohammed Al Owayyed (TUD), Interactive Simulation-Based Learning Tools for Training Children's Helpline Counsellors
27 Bram Grooten (TU/e), Adaptive Reinforcement Learning: Lean and Dynamic Agents for Robust Generalization
28 Anouck Braggaar (TiU), Does This Answer Your question? Evaluating, Replicating, and Improving Task-Oriented Dialogue Systems
29 Afsana Khan (UM), Unlocking the Value of Data with Vertical Federated Learning
30 Yucheng Yang (TU/e), Generalization in Reinforcement Learning: Theories and Algorithms for Downstream Task and Multi-objective Trade-off Adaptation
31 Luis Pedro Silvestrin (VUA), Efficient Machine Learning for Time-Varying Data: Contributions to Time-Series Machine Learning and Transfer Learning
32 Giovanni Varricchione (UU), Declarative Specifications for Efficient and Safe Reinforcement Learning
33 Markus Funke (VUA), Carving Sustainability into Software Architecture
34 Melika Ayoughi (UvA), External and Internal Semantics for Visual Understanding
35 Clinton Cao (TUD), Modeling Behavior Patterns in Microservice Applications: Practical Applications Using Finite State Machines
36 Jonathan Kamp (VUA), Interpreting the Methods that Interpret Language Models

37 Emre Erdogan (UU), Computational Theory of Mind for Effective Hybrid Intelligence
38 Mahmoud Shokrollahi-Far (TiU), Quran Mining: Computational Stylometry of Quranic Texts
39 Aleksandr Chebykin (CWI, TUD), Efficient Evolutionary Hyperparameter Optimization for Deep Learning
40 Kateryna Ihorivna Holubinka (OU), Holistic Integration of Desktop Virtual Reality Technology in Higher Education: A Design-Based Research Approach
41 Hideaki Joko (RUN), Personalization and Evaluation of Conversational Information Access
42 Nedo Alexander Bartels (VUA), Conceptualizing Platform Revenue Models: A Taxonomy-Driven Design Approach