

MANAS: Multi-Agent Neural Architecture Search

Vasco Lopes · Fabio Maria Carlucci ·
Pedro M Esperança · Marco Singh ·
Antoine Yang · Victor Gabillon · Hang
Xu · Zewei Chen · Jun Wang

Received: date / Accepted: date

Abstract The Neural Architecture Search (NAS) problem is typically formulated as a graph search problem where the goal is to learn the optimal operations over edges in order to maximize a graph-level global objective. Due to the large architecture parameter space, efficiency is a key bottleneck preventing NAS from its practical use. In this work, we address the issue by framing NAS as a multi-agent problem where agents control a subset of the network and coordinate to reach optimal architectures. We provide two distinct lightweight implementations, with reduced memory requirements (1/8th of state-of-the-art), and performances above those of much more computationally expensive methods. Theoretically, we demonstrate vanishing regrets of the form $\mathcal{O}(\sqrt{T})$, with T being the total number of rounds. Finally, we perform experiments on CIFAR-10 and ImageNet, and aware that random search and random sampling are (often ignored) effective baselines, we conducted additional experiments on 3 alternative datasets, with complexity constraints, and 2 network configurations, and achieve competitive results in comparison with the baselines and other methods.

Keywords Neural Architecture Search · Multi Arm Bandits · AutoML · Computer Vision · Object Recognition

V. Lopes,
NOVA Lincs, Universidade da Beira Interior

F.M. Carlucci, P.M. Esperança, M. Singh, A. Yang, V. Gabillon, X. Hang, Z. Chen
Huawei Noah's Ark Lab
E-mail: fabiom.carlucci@gmail.com

J. Wang
University College London

1 Introduction

Determining an optimal architecture is key to accurate deep neural networks (DNNs) with good generalisation properties [39, 19, 18, 17, 11, 31]. Neural architecture search (NAS), which has been formulated as a graph search problem, can potentially reduce the need for application-specific expert designers allowing for a wide-adoption of sophisticated networks in various industries. [53] presented the first modern algorithm automating structure design, and showed that resulting architectures can indeed outperform human-designed state-of-the-art convolutional networks [20, 26]. However, even in the current settings where flexibility is limited by expertly-designed search spaces, NAS problems are computationally very intensive with early methods requiring hundreds or thousands of GPU-days to discover state-of-the-art architectures [53, 37, 24, 25].

Researchers have used a wealth of techniques ranging from reinforcement learning, where a controller network is trained to sample promising architectures [53, 54, 33, 4], to evolutionary algorithms that evolve a population of networks for optimal DNN design [36, 25, 27], to optimization on random graphs [38]. Alas, these approaches are inefficient and can be extremely computationally and/or memory intensive as some require all tested architectures to be trained from scratch. Weight-sharing, introduced in ENAS [33], can alleviate this problem. Even so, these techniques cannot easily scale to large datasets, e.g., ImageNet, relying on human-defined heuristics for architecture transfer. Recently, low-fidelity estimates, performance predictors and guiding mechanisms have also been studied to improve the search cost and reduce the memory and computation required [30, 28, 44, 32, 45, 29]. More, gradient-based frameworks enabled efficient solutions by introducing a continuous relaxation of the search space. For example, DARTS [26] uses this relaxation to optimise architecture parameters using gradient descent in a bi-level optimisation problem, while SNAS [46] updates architecture parameters and network weights under one generic loss. Still, due to memory constraints the search has to be performed on 8 cells, which are then stacked 20 times for the final architecture. This solution is a coarse approximation to the original problem as shown in Section 6 of this work and in [47, 50, 22], receiving some criticisms outlined in [51, 47, 41]. In fact, we show that searching directly over 20 cells leads to a reduction in test error (8% relative to [26]). ProxylessNAS [6] is one exception, as it can search for the final models directly; nonetheless they still require twice the amount of memory used by our proposed algorithm, while offering no theoretical guarantees.

To enable the possibility of large-scale joint optimisation of deep architectures we contribute MANAS, the first multi-agent learning algorithm for neural architecture search. Our algorithm combines the memory and computational efficiency of multi-agent systems, which is achieved through action coordination with the theoretical rigour of online machine learning, allowing us to balance exploration versus exploitation optimally. Due to its distributed nature, MANAS enables large-scale optimisation of deeper networks while learning different

operations per cell. Theoretically, we demonstrate that MANAS implicitly coordinates learners to recover vanishing regrets, guaranteeing convergence. Empirically, we show that our method achieves state-of-the-art accuracy results among methods using the same evaluation protocol but with significant reductions in memory (1/8th of [26]) and search time (70% of [26]).

The multi-agent (MA) framework is inherently scalable and allows us to tackle an optimization problem that would be extremely challenging to solve efficiently otherwise: the search space of a single cell is 8^{14} and there is no fast way of learning the joint distribution, as needed by a single controller. More cells to learn exacerbates the problem, and this is why MA is required, as for each agent the size of the search space is always constant.

In short, our contributions can be summarised as: (1) framing NAS as a multi-agent learning problem (MANAS) where each agent supervises a subset of the network; agents coordinate through a credit assignment technique which infers the quality of each operation in the network, without suffering from the combinatorial explosion of potential solutions. (2) Proposing two lightweight implementations of our framework that are theoretically grounded. The algorithms are computationally and memory efficient, and achieve state-of-the-art results on CIFAR-10 and ImageNet when compared with competing methods. Furthermore, MANAS allows search *directly* on large datasets (e.g. ImageNet). (3) Presenting 3 news datasets for NAS evaluation to minimise algorithmic overfitting; offering a fair comparison with the often ignored random search [22] and random sampling [47, 50] baselines; and presenting a complexity constraint analysis of MANAS.

2 Related work

MANAS derives its search space from DARTS [26] and is therefore most related to other gradient-based NAS methods that use the same search space. SNAS [46] appears similar at a high level, but has important differences: 1) it uses GD to learn the architecture parameters. This requires a differentiable objective (MANAS does not) and leads to 2) having to forward all operations (see their Eqs.5,6), thus negating any memory advantages (which MANAS has), and effectively requiring repeated cells and preventing search on ImageNet. Sequent gradient-base proposals improve upon baselines by introducing regularization mechanisms to improve the final performance of the generated architectures [51, 8, 9], whilst still suffering from the aforementioned problems.

ENAS [33] is also very different: its use of RL implies dependence on past states (the previous operations in the cell). It explores not only the stochastic reward function but also the relationship between states, which is where most of the complexity lies. Furthermore, RL has to balance exploration and exploitation by relying on sub-optimal heuristics, while MANAS, due to its theoretically optimal approach from online learning, is more sample efficient. Finally, ENAS uses a single LSTM (which adds complexity and problems such as exploding/vanishing gradients) to control the entire process, and is

thus following a monolithic approach. Indeed, at a high level, our multi-agent framework can be seen as a way of decomposing the monolithic controller into a set of simpler, independent sub-policies. This provides a more scalable and memory efficient approach that leads to higher accuracy, as confirmed by our experiments.

3 Preliminary: Neural Architecture Search

We consider the NAS problem as formalised in DARTS [26]. At a higher level, the architecture is composed of a *computation cell* that is a building block to be learned and stacked in the network. The cell is represented by a directed acyclic graph with V nodes and N edges; edges connect all nodes i, j from i to j where $i < j$. Each vertex $\mathbf{x}^{(i)}$ is a latent representation for $i \in \{1, \dots, V\}$. Each directed edge (i, j) (with $i < j$) is associated with an operation $o^{(i,j)}$ that transforms $\mathbf{x}^{(i)}$. Intermediate node values are computed based on all of its predecessors as $\mathbf{x}^{(j)} = \sum_{i < j} o^{(i,j)}(\mathbf{x}^{(i)})$. For each edge, an architect needs to intelligently select one operation $o^{(i,j)}$ from a finite set of K operations, $\mathcal{O} = \{o_k(\cdot)\}_{k=1}^K$, where operations represents some function to be applied to $\mathbf{x}^{(i)}$ to compute $\mathbf{x}^{(j)}$, e.g., convolutions or pooling layers. To each $o_k^{(i,j)}(\cdot)$ is associated a set of operational weights $w_k^{(i,j)}$ that needs to be learned (e.g. the weights of a convolution filter). Additionally, a parameter $\alpha_k^{(i,j)} \in \mathbb{R}$ characterises the importance of operation k within the pool \mathcal{O} for edge (i, j) . The sets of all the operational weights $\{w_k^{(i,j)}\}$ and architecture parameters (edge weights) $\{\alpha_k^{(i,j)}\}$ are denoted by \mathbf{w} and $\boldsymbol{\alpha}$, respectively. DARTS defined the operation $\bar{o}^{(i,j)}(\mathbf{x})$ as

$$\bar{o}^{(i,j)}(\mathbf{x}) = \sum_{k=1}^K \frac{e^{\alpha_k^{(i,j)}}}{\sum_{k'=1}^K e^{\alpha_{k'}^{(i,j)}}} \cdot o_k^{(i,j)}(\mathbf{x}) \quad (1)$$

in which $\boldsymbol{\alpha}$ encodes the network architecture; and the optimal choice of architecture is defined by

$$\boldsymbol{\alpha}^* = \min_{\boldsymbol{\alpha}} \mathcal{L}^{(\text{val})}(\boldsymbol{\alpha}, \mathbf{w}^*(\boldsymbol{\alpha})) \quad \text{s.t.} \quad \mathbf{w}^*(\boldsymbol{\alpha}) = \arg \min_{\mathbf{w}} \mathcal{L}^{(\text{train})}(\boldsymbol{\alpha}, \mathbf{w}). \quad (2)$$

The final objective is to obtain a *sparse* architecture $\mathcal{Z}^* = \{\mathcal{Z}^{(i,j)}\}, \forall i, j$ where $\mathcal{Z}^{(i,j)} = [z_1^{(i,j)}, \dots, z_K^{(i,j)}]$ with $z_k^{(i,j)} = 1$ for k corresponding to the best operation and 0 otherwise. That is, for each pair (i, j) a *single operation* is selected.

4 Online Multi-agent Learning for AutoML

NAS suffers from a combinatorial explosion in its search space. A recently proposed approach to tackle this problem is to approximate the discrete optimisation variables (i.e., edges in our case) with continuous counterparts

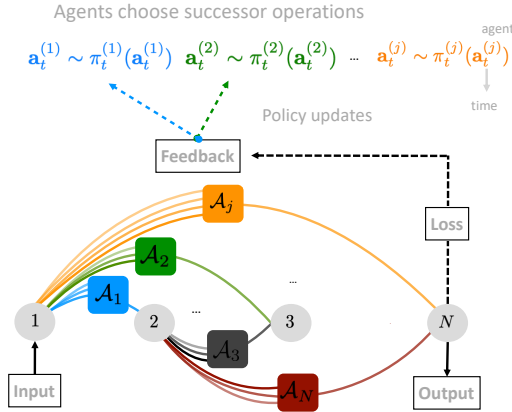


Fig. 1 MANAS with single cell. Between each pair of nodes, an agent \mathcal{A}_i selects action $a^{(i)}$ according to $\pi^{(i)}$. Feedback from the validation loss is used to update the policy.

and then use gradient-based optimisation methods. DARTS [26] introduced this method for NAS, though it suffers from two important drawbacks. First, the algorithm is memory and computationally intensive ($\mathcal{O}(NK)$ with K being total number of operations between a pair of nodes and N the number of nodes) as it requires loading all operation parameters into GPU memory. As a result, DARTS only optimises over a small subset of 8 repeating cells, which are then stacked together to form a deep network of 20. Naturally, such an approximation is bound to be sub-optimal. Second, evaluating an architecture on validation requires the optimal set of network parameters. Learning these, unfortunately, is highly demanding since for an architecture \mathcal{Z}_t , one would like to compute $\mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t, \mathbf{w}_t^*)$ where $\mathbf{w}_t^* = \arg \min_{\mathbf{w}} \mathcal{L}_t^{(\text{train})}(\mathbf{w}, \mathcal{Z}_t)$. DARTS, uses *weight sharing* that updates \mathbf{w}_t once per architecture, with the hope of tracking \mathbf{w}_t^* over learning rounds. Although this technique leads to significant speed up in computation, it is not clear how this approximation affects the validation loss function.

Next, we detail a novel methodology based on a combination of multi-agent and online learning to tackle the above two problems (Figure 1). Multi-agent learning scales our algorithm, reducing memory consumption by an order of magnitude from $\mathcal{O}(NK)$ to $\mathcal{O}(N)$; and online learning enables rigorous understanding of the effect of tracking \mathbf{w}_t^* over rounds.

4.1 NAS as a multi-agent problem

To address the computational complexity we use the weight sharing technique used in DARTS. However, we handle in a more theoretically grounded way the effect of approximating $\mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t, \mathbf{w}_t^*)$ by $\mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t, \mathbf{w}_t)$. Indeed, such an

approximation can lead to arbitrary bad solutions due to the uncontrollable weight component. To analyse the learning problem with no stochastic assumptions on the process generating $\nu = \{\mathcal{L}_1, \dots, \mathcal{L}_T\}$ we adopt an adversarial online learning framework.

Algorithm 1 GENERAL FRAMEWORK: [steps with asterisks (*) are specified in section 5]

- 1: **Initialize:** π_1^i is uniform random over all $j \in \{1, \dots, N\}$. And random \mathbf{w}_1 weights.
 - 2: **For** $t = 1, \dots, T$
 - 3: * Agent \mathcal{A}_i samples $\mathbf{a}_t^i \sim \pi_t^i(\mathbf{a}_t^i)$ for all $i \in \{1, \dots, N\}$, forming architecture \mathcal{Z}_t .
 - 4: Compute the training loss $\mathcal{L}_t^{(\text{train})}(\mathbf{a}_t) = \mathcal{L}_t^{(\text{train})}(\mathcal{Z}_t, \mathbf{w}_t)$
 - 5: Update \mathbf{w}_{t+1} for all operation \mathbf{a}_t^i in \mathcal{Z}_t from \mathbf{w}_t using back-propagation.
 - 6: Compute the validation loss $\mathcal{L}_t^{(\text{val})}(\mathbf{a}_t) = \mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t, \mathbf{w}_{t+1})$
 - 7: * Update π_{t+1}^i for all $i \in \{1, \dots, N\}$ using $\mathcal{Z}_1, \dots, \mathcal{Z}_t$ and $\mathcal{L}_1^{(\text{val})}, \dots, \mathcal{L}_t^{(\text{val})}$.
 - 8: **Recommend** \mathcal{Z}_{T+1} , after round T , where $\mathbf{a}_{T+1}^i \sim \pi_{T+1}^i(\mathbf{a}_{T+1}^i)$ for all $i \in \{1, \dots, N\}$.
-

NAS as Multi-Agent Combinatorial Online Learning. In Section 3, we defined a NAS problem where one out of K operations needs to be recommended for each pair of nodes (i, j) in a DAG. In this section, we associate *each pair* of nodes with an *agent* in charge of exploring and quantifying the quality of these K operations, to ultimately recommend one. The only feedback for each agent is the loss that is associated with a global architecture \mathcal{Z} , which depends on all agents' choices.

We introduce N decision makers, $\mathcal{A}_1, \dots, \mathcal{A}_N$ (see Figure 1 and Algorithm 1). At training round t , each agent chooses an operation (e.g., convolution or pooling filter) according to its local action-distribution (or policy) $\mathbf{a}_t^j \sim \pi_t^j$, for all $j \in \{1, \dots, N\}$ with $\mathbf{a}_t^j \in \{1, \dots, K\}$. These operations have corresponding operational weights \mathbf{w}_t that are learned in parallel. Altogether, these choices $\mathbf{a}_t = \mathbf{a}_t^1, \dots, \mathbf{a}_t^N$ define a sparse graph/architecture $\mathcal{Z}_t \equiv \mathbf{a}_t$ for which a validation loss $\mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t, \mathbf{w}_t)$ is computed and used by the agents to update their policies. After T rounds, an architecture is recommended by sampling $\mathbf{a}_{T+1}^j \sim \pi_{T+1}^j$, for all $j \in \{1, \dots, N\}$. These dynamics resemble bandit algorithms where the actions for an agent \mathcal{A}_j are viewed as separate arms. This framework leaves open the design of **1)** the sampling strategy π^j and **2)** how π^j is updated from the observed loss.

Minimization of worst-case regret under any loss. The following two notions of regret motivate our proposed NAS method. Given a policy π the *cumulative regret* $\mathcal{R}_{T,\pi}^*$ and the *simple regret* $r_{T,\pi}^*$ after T rounds and under the worst

possible environment ν , are:

$$\mathcal{R}_{T,\pi}^* = \sup_{\nu} \mathbb{E} \sum_{t=1}^T \mathcal{L}_t(\mathbf{a}_t) - \min_{\mathbf{a}} \sum_{t=1}^T \mathcal{L}_t(\mathbf{a}), \quad (3)$$

$$r_{T,\pi}^* = \sup_{\nu} \mathbb{E} \sum_{t=1}^T \mathcal{L}_t(\mathbf{a}_{T+1}) - \min_{\mathbf{a}} \sum_{t=1}^T \mathcal{L}_t(\mathbf{a}) \quad (4)$$

where the expectation is taken over both the losses and policy distributions, and $\mathbf{a} = \{\mathbf{a}^{(A_j)}\}_{j=1}^N$ denotes a joint action profile. The simple regret leads to minimising the loss of the recommended architecture a_{T+1} , while minimising the cumulative regret adds the extra requirement of having to sample, at any time t , architectures with close-to-optimal losses. We discuss in the appendix 7 how this requirement could improve in practice the tracking of \mathbf{w}_t^* by \mathbf{w}_t . We let $\mathcal{L}_t(\mathbf{a}_t)$ be potentially adversarially designed to account for the difference between \mathbf{w}_t^* and \mathbf{w}_t and make no assumption on its convergence. Our models and solutions in Section 5 are designed to be robust to arbitrary $\mathcal{L}_t(\mathbf{a}_t)$.

Because of the discrete nature of the NAS problem, during search the loss can take on large values or alternate between large and small values arbitrarily. Gradient-descent methods perform best under smooth loss functions, which is not the case in NAS. The worst-case regret minimization is a theoretically-grounded objective which we make use of in order to provide guarantees on the convergence of the algorithm when no assumptions are made on the process generating the losses.

5 Adversarial Implementations

In the following subsections we will describe our proposed approaches for NAS when considering adversarial losses. We present two algorithms, MANAS and MANAS-LS, that implement two different *credit assignment techniques* specifying the update rule in line 7 of Algorithm 1. The first approximates the validation loss as a linear combination of edge weights, while the second handles non-linear losses.

Note that *adversarial* in this context refers to the adversarial multi-arm bandit [3] framework: we model the fact that a weight-sharing supernet returns noisy rewards as having an adversary that explicitly tries to confuse the learner. Adversarial multi-arm bandit is the strongest generalization of the bandit problem, as it removes all assumptions on the distribution. Our MA formulation and algorithm explicitly account for this adversarial nature and provide a principled solution that is provably robust.

5.1 MANAS-LS

Linear Decomposition of the Loss. A simple credit assignment strategy is to approximate edge-importance (or edge-weight) by a vector $\beta_s \in \mathbb{R}^{KN}$

representing the importance of all K operations for each of the N agents. β_s is an arbitrary, potentially adversarially-chosen vector and varies with time s to account for the fact that the operational weights w_s are learned online and to avoid any restrictive assumption on their convergence. The relation between the observed loss $\mathcal{L}_s^{(\text{val})}$ and the architecture selected at each sampling stage s is modeled through a linear combination of the architecture’s edges (agents’ actions) as

$$\mathcal{L}_s^{(\text{val})} = \beta_s^\top \mathbf{Z}_s \quad (5)$$

where $\mathbf{Z}_s \in \{0, 1\}^{KN}$ is a vectorised one-hot encoding of the architecture \mathcal{Z}_s (active edges are 1, otherwise 0). After evaluating S architectures, at round t we estimate β by solving the following via least-squares:

$$\text{Credit assignment: } \tilde{\mathbf{B}}_t = \arg \min_{\beta} \sum_{s=1}^S \left(\mathcal{L}_s^{(\text{val})} - \beta^\top \mathbf{Z}_s \right)^2. \quad (6)$$

The solution gives an efficient way for agents to update their corresponding action-selection rules and leads to implicit coordination. Indeed, in Appendix C we demonstrate that the worst-case regret \mathcal{R}_T^* (3) can actually be decomposed into an agent-specific form $\mathcal{R}_T^i(\pi^i, \nu^i)$ defined in the appendix: $\mathcal{R}_T^* = \sup_{\nu} \mathcal{R}_T(\pi, \nu) \iff \sup_{\nu^i} \mathcal{R}_T^i(\pi^i, \nu^i)$, $i = 1, \dots, N$. This decomposition allows us to significantly reduce the search space complexity by letting each agent \mathcal{A}_i determine the best operation for the corresponding graph edge.

Zipf Sampling for $r_{T,\pi}^$.* \mathcal{A}_i samples an operation k proportionally to the inverse of its estimated rank $\langle \widetilde{k} \rangle_t^i$, where $\langle \widetilde{k} \rangle_t^i$ is computed by sorting the operations of agent \mathcal{A}_i w.r.t $\tilde{\mathbf{B}}_t^i[k]$, as

$$\text{Sampling policy: } \pi_{t+1}^i[k] = 1 / \langle \widetilde{k} \rangle_t^i \overline{\log K} \quad \text{where } \overline{\log K} = 1 + 1/2 + \dots + 1/K.$$

Zipf explores efficiently, is anytime, parameter free, minimises optimally the simple regret in multi-armed bandits when the losses are adversarially designed and adapts optimally to stationary losses [1].

We prove for this new algorithm an exponentially decreasing simple regret $r_T^* = \mathcal{O}(e^{-T/H})$, where H is a measure of the complexity for discriminating sub-optimal solutions as $H = N(\min_{j \neq k_i^*, 1 \leq i \leq N} \mathbf{B}_T^i[j] - \mathbf{B}_T^i[k_i^*])$, where $k_i^* = \min_{1 \leq j \leq K} \mathbf{B}_T^i[j]$ and $\mathbf{B}_T^i[j] = \sum_{t=1}^T \beta_t^{(\mathcal{A}_i)}[j]$. The proof is given in Appendix D.1.

5.2 MANAS

Coordinated Descent for Non-Linear Losses. In some cases the linear approximation may be crude. An alternative is to make no assumptions on the loss function and have each agent directly associate the quality of their actions with the loss $\mathcal{L}_t^{(\text{val})}(\mathbf{a}_t)$. This results in all the agents performing a coordinated

descent approach to the problem. Each agent updates for operation k its $\tilde{\mathbf{B}}_t^i[k]$ as

$$\text{Credit assignment: } \tilde{\mathbf{B}}_t^i[k] = \tilde{\mathbf{B}}_{t-1}^i[k] + \mathcal{L}_t^{(\text{val})} \cdot \mathbb{1}_{\mathbf{a}_t^i=k} / \boldsymbol{\pi}_t^i[k]. \quad (7)$$

Softmax Sampling for $\mathcal{R}_{T,\pi}^$.* Following EXP3 [2], actions are sampled from a softmax distribution (with temperature η) w.r.t. $\tilde{\mathbf{B}}_t^i[k]$:

$$\text{Sampling policy: } \boldsymbol{\pi}_{t+1}^i[k] = \exp\left(\eta \tilde{\mathbf{B}}_t^i[k]\right) / \sum_{j=1}^K \exp\left(\eta \tilde{\mathbf{B}}_t^i[j]\right).$$

Using this sampling strategy, EXP3 [2] is run for each agent in parallel. If the regret of each agent is computed by considering the rest of the agent as fixed, then each agent has regret $\mathcal{O}(\sqrt{TK \log K})$ which sums over agents to $\mathcal{O}(N\sqrt{TK \log K})$. The proof is given in Appendix D.1

On credit assignment. Our MA formulation provides a gradient-free, credit assignment strategy. Gradient methods are more susceptible to bad initialisation and can get trapped in local minima more easily than our approach, which, not only explores more widely the search space, but makes this search optimally according to multi-armed bandit derived regret minimization. Concretely, MANAS can easily escape from local minima as the reward is scaled by the probability of selecting an action (Eq. 7). Thus, the algorithm has a higher chance of revising its estimate of the quality of a solution based on new evidence. This is important as one-shot methods (such as MANAS and DARTS) change the network—and thus the environment—throughout the search process. Put differently, MANAS’ optimal exploration-exploitation allows the algorithm to move away from ‘good’ solutions towards ‘very good’ solutions that do not live in the former’s proximity; in contrast, gradient methods will tend to stay in the vicinity of a ‘good’ discovered solution.

6 Experiments

We (1) compare MANAS against existing NAS methods on the well established CIFAR-10 dataset; (2) evaluate MANAS on ImageNet; (3) compare MANAS, DARTS, Random Sampling and Random Search with WS [22] on 3 new datasets (Sport-8, Caltech-101, MIT-67); and (4) evaluate MANAS with inference time as complexity constraint. Descriptions of the datasets and details of the search are provided in the Appendix. We report the performance of two algorithms, MANAS and MANAS-LS, as described in Section 5. Note that, with the exception of results marked as *+AutoAugment*, all experiments were run with the same final training protocol as DARTS [26], for fair comparison.

Search Spaces. We use the same CNN search space as [26]. Since MANAS is memory efficient, it can search for the final architecture without needing to stack *a posteriori* repeated cells; thus, all our cells are unique. For fair comparison, we use 20 cells on CIFAR-10 and 14 on ImageNet. Experiments on Sport-8, Caltech-101 and MIT-67 in Section 6.3 use both 8 and 14 cell networks.

Search Protocols. For datasets other than ImageNet, we use 500 epochs during the search phase for architectures with 20 cells, 400 epochs for 14 cells, and 50 epochs for 8 cells. All other hyperparameters are as in [26]. For ImageNet, we use 14 cells and 100 epochs during search. In our experiments on the three new datasets we rerun the DARTS code to optimise an 8 cell architecture; for 14 cells we simply stacked the best cells for the appropriate number of times.

Synthetic experiment. To illustrate the theoretical properties of MANAS we apply it to the Gaussian Squeeze task, a problem where agents must coordinate their actions in order to optimize a global objective function that depends on the actions of each agent [48, 10]. Specifically, N homogeneous agents determine their individual actions $a^{(j)}$ to jointly optimize the objective $G(x) = xe^{-\frac{(x-\mu)^2}{\sigma^2}}$ where $x = \sum_{j=1}^N a^{(j)}$. This synthetic setup has the same characteristics of the multi-agent NAS problem, namely a group of agents *implicitly* coordinating their actions to achieve a global objective, and is therefore a good experiment to showcase the theoretical properties of the MANAS algorithm.

We confirm that (1) MANAS progresses steadily towards zero regret while the Random Search baseline struggles to move beyond the initial starting point; (2) MANAS stays well within the theoretical cumulative regret bound (Figure 2).

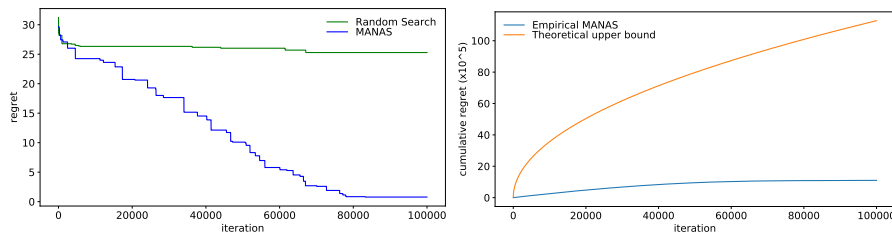


Fig. 2 Left: Regret for the Gaussian Squeeze Domain experiment with 100 agents, 10 actions, $\mu = 1$, $\sigma = 10$. Right: Theoretical bound for the MANAS cumulative regret ($2N\sqrt{TK \log K}$; see Appendix D.2) and the observed counterpart for the Gaussian Squeeze Domain experiment with 100 agents, 10 actions, $\mu = 1$, $\sigma = 10$.

Table 1 Comparison with state-of-the-art image classifiers on CIFAR-10. The four row blocks represent: human-designed, NAS, MANAS search with DARTS training protocol and best searched MANAS retrained with extended protocol (AutoAugment + 1500 Epochs + 50 Channels). Unless specified, all architectures use 20 cells.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC [19]	3.46	25.6	—	manual
NASNet-A [54]	2.65	3.3	1800	RL
AmoebaNet-B [36]	2.55	2.8	3150	evolution
PNAS [24]	3.41	3.2	225	SMBO
ENAS [33]	2.89	4.6	0.5	RL
SNAS [46]	2.85	2.8	1.5	gradient
DARTS, 1st order [26]	3.00	3.3	1.5 [†]	gradient
DARTS, 2nd order [26]	2.76	3.3	4 [†]	gradient
SDARTS-ADV [8]	2.61	3.3	4.3	gradient
DARTS- [9]	2.63	3.5	4 [†]	gradient
GDAS[14]	3.75	3.4	0.17	gradient
NPENAS-BO[43]	2.52	4.0	2.5	evolution
EffNet[42]	3.49	2.54	3	evolution
BANANAS[44]	2.64	—	11.8	BO + predictor
Random + cutout [26]	3.29	3.2	—	—
Random Search WS [22]	2.85	4.3	9.7	random search
MANAS (8 cells)	3.05	1.6	0.8 [†]	MA
MANAS	2.63	3.4	2.8 [†]	MA
MANAS-LS	2.52	3.4	4 [†]	MA
MANAS + AutoAugment	1.97	3.4	—	MA
MANAS-LS + AutoAugment	1.85	3.4	—	MA

[†] Search cost is for 4 runs and test error is for the best result (for a fair comparison with other methods).

6.1 Results on CIFAR-10

Evaluation. To evaluate our NAS algorithm, we follow DARTS’s protocol: we run MANAS 4 times with different random seeds and pick the best architecture based on its validation performance. We then randomly reinitialize the weights and retrain for 600 epochs. During search we use half of the training set as validation. To fairly compare with more recent methods, we also re-train the best searched architecture using AutoAugment and Extended Training [12].

Results. Both MANAS implementations perform well on this dataset (Table 1). Our algorithm is designed to perform comparably to [26] but with an order of magnitude less memory. However, MANAS actually achieves higher accuracy. The reason for this is that DARTS is forced to search for an 8 cell architecture and subsequently stack the same cells 20 times; MANAS, on the other hand, can directly search on the final number of cells leading to better results. We also report our results when using only 8 cells: even though the network is much smaller, it still performs competitively with 1st-order 20-cell DARTS. This is explored in more depth in Section 6.3. In terms of memory usage with a batch size of 1, MANAS 8 cells required only 1GB of GPU memory, while DARTSv1 utilized more than 8.5GB and DARTSv2 required 9.6GB, making

Table 2 Comparison with state-of-the-art image classifiers on ImageNet (mobile setting). The four row blocks represent: human-designed, NAS, MANAS search with DARTS training protocol and best searched MANAS retrained with extended protocol (AutoAugment + 600 Epochs + 60 Channels).

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
ShuffleNet 2x (v2) [52]	26.3	5	—	manual
NASNet-A [54]	26.0	5.3	1800	RL
AmoebNet-C [36]	24.3	6.4	3150	evolution
PNAS [24]	25.8	5.1	225	SMBO
SNAS [46] (search on C10)	27.3	4.3	1.5	gradient
DARTS [26] (search on C10)	26.7	4.7	4	gradient
GDAS [14] (search on C10)	27.5	4.4	0.17	gradient
EffNet [42] (search on C10)	27.1	—	3	evolution
NASP [49] (search on C10)	26.3	9.5	0.2	proximal
Random sampling	27.75	2.5	—	—
MANAS (search on C10)	26.47	2.6	2.8	MA
MANAS (search on IN)	26.15	2.6	110	MA
MANAS (search on C10) + AutoAugment	26.81	2.6	—	MA
MANAS (search on IN) + AutoAugment	25.26	2.6	—	MA

both versions of DARTS unpractical to work with datasets with larger image sizes.

[6] is another method designed as an efficient alternative to DARTS; unfortunately the authors decided to a) use a different search space (PyramidNet backbone; [17]) and b) offer no comparison to random sampling in the given search space. For these reasons we feel a numerical comparison to be unfair. Furthermore our algorithm uses half the GPU memory (they sample 2 paths at a time) and does not require the reward to be differentiable. Lastly, we observe similar gains when training the best MANAS/MANAS-LS architectures with an extended protocol (AutoAugment + 1500 Epochs + 50 Channels, in addition to the DARTS protocol).

6.2 Results on ImageNet

Evaluation. To evaluate the results on ImageNet we train the final architecture for 250 epochs. We report the result of the best architecture out of 4, as chosen on the validation set for a fair comparison with competing methods. As search and augmentation are very expensive we use only MANAS and not MANAS-LS, as the former is computationally cheaper and performs slightly better on average.

Results. We provide results for networks searched both on CIFAR-10 and directly on ImageNet, which is made possible by the computational efficiency of MANAS (Table 2). When compared to SNAS, DARTS, GDAS and other methods, using the same search space, MANAS achieves state-of-the-art results both with architectures searched directly on ImageNet and also with architectures transferred from CIFAR-10. We observe similar improvements when

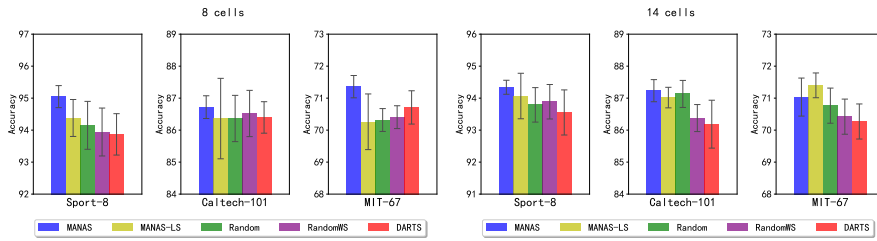


Fig. 3 Comparing MANAS, random sampling, random search with WS [22] and DARTS [26] on 8 and 14 cells. Average results of 8 runs. Note that DARTS was only optimised for 8 cells due to memory constraints.

training the best MANAS architecture with an extended training protocol (AutoAugment + 600 Epochs + 60 Channels, in addition to the DARTS protocol), resulting in a final test error of 25.26% when directly searching on ImageNet.

6.3 Results on new datasets: Sport-8, Caltech-101, MIT-67

Evaluation. The idea behind NAS is that of finding the optimal architecture, given *any* sets of data and labels. Limiting the evaluation of current methods to CIFAR-10 and ImageNet could potentially lead to algorithmic overfitting. Indeed, recent results suggest that the search space was engineered in a way that makes it very hard to find a bad architecture [22, 50, 47, 40]. To mitigate this, we propose testing NAS algorithms on 3 datasets (composed of regular sized images) that were never before used in this setting, but have been historically used in the CV field: Sport-8, Caltech-101 and MIT-67, described briefly in the Appendix. For these set of experiments we run the algorithm 8 times and report mean and std. We perform this both for 8 and 14 cells; we do the same with DARTS (which, due to memory constraints can only search for 8 cells). As baselines, we consider random search and random sampling. For the latter we simply sample uniformly 8 architectures from the search space. To efficiently implement random search, we follow [22] and perform experiments on random search with WS. Each proposed architecture is trained from scratch for 600 epochs as in the previous section.

Results. MANAS manages to outperform the random baselines and significantly outperform DARTS, especially on 14 cells (Figure 3): this clearly shows that the optimal cell architecture for 8 cells is *not* the optimal one for 14 cells.

6.4 Results with Complexity Constraint

Evaluation. To evaluate the results of MANAS in a complexity constraint setting, we added the inference time of the generated architectures as a complexity constraint to the training. For this, we update the training loss

Table 3 Results of MANAS with complexity constraints using different penalty (λ) values on CIFAR-10.

λ	Test Error (%)	Inference Time (ms)
0	2.91	1.255 \pm 0.012
0.1	3.12	1.196 \pm 0.009
0.25	2.94	1.190 \pm 0.008
0.5	3.04	1.183 \pm 0.006
0.75	2.54	1.179 \pm 0.005
1	2.69	1.164 \pm 0.006

with the constraint of the inference time that the generated architecture takes to classify an image: $\mathcal{L}_t^{(\text{train})}(\mathbf{a}_t) = \mathcal{L}_t^{(\text{train})}(\mathcal{Z}_t, \mathbf{w}_t) + \lambda L_t(\mathcal{Z}_t, \mathbf{w}_t)$, where L_t is the inference time to classify one image, and λ defines the importance given to L_t . Here, the λ serves the purpose of varying the importance given to the inference time whilst searching. By increasing λ , the inference time constraint has a higher importance.

Results. We evaluate MANAS with different λ values using a single GPU (Table 3), and observe that by increasing the importance given to the inference, MANAS consistently generates architectures with a lower inference time, with similar accuracies. This experiment shows that MANAS can be extended for searching with multiple objectives, by modifying the training loss.

7 Discussion

Random Baselines. Clearly, in specific settings, random sampling performs very competitively. On one hand, since the search space is very large (between 8^{112} and 8^{280} architectures exist in the DARTS experiments), finding the global optimum is practically impossible. Why is it then that the randomly sampled architectures are able to deliver nearly state-of-the-art results? Previous experiments [50, 22] together with the results presented here seem to indicate that the available operations and meta-structure have been carefully chosen and, as a consequence, most architectures in this space generate very good results. This suggests that human effort has simply transitioned from finding a good architecture to finding a good search space—a problem that needs careful consideration in future work. Random search with WS [22], has also shown to perform competitively but it is clearly sub-optimal compared to our multi-agent framework.

On fair evaluation. It is worth stressing that we performed all comparisons using the same final training protocol. This is extremely relevant as there has been a recent trend to boost results simply by stacking more training tricks on to the evaluation protocol. As such, any improvement in the final accuracy is solely due to how the network was trained rather than the quality of the search method used or the architecture discovered [47].

Agent coordination, combinatorial explosion and approximate credit assignment. Our set-up introduces multiple agents in need of coordination. Centralised critics use explicit coordination and learn the value of coordinated actions across all agents [35], but the complexity of the problem grows exponentially with the number of possible architectures \mathcal{Z} , which equals K^N . We argue instead for an implicit approach where coordination is achieved through a joint loss function depending on the actions of all agents. This approach is scalable as each agent searches its local action space—small and finite—for optimal action-selection rules. Both credit assignment methods proposed learn, for each operation k belonging to an agent \mathcal{A}_i , a quantity $\tilde{\mathbf{B}}_t^i[k]$ (similar to α in Section 3) that quantifies the contribution of the operation to the observed losses.

8 Conclusions

We presented MANAS, a theoretically grounded multi-agent online learning framework for NAS. We proposed two extremely lightweight implementations that, within the same search space, outperform state-of-the-art while reducing memory consumption by an order of magnitude compared to [26]. We provide vanishing regret proofs for our algorithms. Furthermore, we evaluate MANAS on 3 new datasets, empirically showing its effectiveness in a variety of settings. Finally, we confirm concerns raised in recent works [50, 22, 47] claiming that NAS algorithms often achieve minor gains over random architectures. We however demonstrate, that MANAS still produces competitive results with limited computational budgets.

Funding

Financial support to the authors was received from “FCT - Fundação para a Ciência e Tecnologia”, through the research grant “2020.04588.BD” [Vasco Lopes]; and from Huawei Technologies R&D (UK) Ltd [all other authors].

Conflicts of interest

The authors declare that they have no conflict of interest.

Ethics approval

Not required.

Consent to participate

Not required.

Consent for publication

Not required.

Availability of data

All data used is publicly available.

Code availability

Code will be publicly available.

Authors' contributions

Conceptualization: Vasco Lopes, Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Antoine Yang, Jun Wang; **Methodology:** Vasco Lopes, Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Antoine Yang, Victor Gabillon, Hang Xu, Zewei Chen; **Formal analysis and investigation:** Vasco Lopes, Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Antoine Yang, Victor Gabillon; **Writing - original draft preparation:** Fabio Maria Carlucci, Pedro M Esperança, Marco Singh, Antoine Yang, Victor Gabillon; **Writing - review and editing:** Vasco Lopes, Fabio Maria Carlucci, Pedro M Esperança; **Supervision:** Jun Wang.

References

1. Abbasi-Yadkori, Y., Bartlett, P., Gabillon, V., Malek, A., Valko, M.: Best of both worlds: Stochastic & adversarial best-arm identification. In: Conference on Learning Theory (COLT) (2018)
2. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The nonstochastic multiarmed bandit problem. *SIAM journal on computing* **32**(1), 48–77 (2002)
3. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: Gambling in a rigged casino: The adversarial multi-armed bandit problem. In: Proceedings of IEEE 36th Annual Foundations of Computer Science, pp. 322–331. IEEE (1995)
4. Bender, G., Liu, H., Chen, B., Chu, G., Cheng, S., Kindermans, P.J., Le, Q.V.: Can weight sharing outperform random architecture search? an investigation with tunas. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 14323–14332 (2020)
5. Bubeck, S., Cesa-Bianchi, N., et al.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning* **5**(1), 1–122 (2012)

6. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct neural architecture search on target task and hardware. In: International Conference on Learning Representations (ICLR) (2019)
7. Cesa-Bianchi, N., Lugosi, G.: Combinatorial bandits. *Journal of Computer and System Sciences* **78**(5), 1404–1422 (2012)
8. Chen, X., Hsieh, C.: Stabilizing differentiable architecture search via perturbation-based regularization. In: Proceedings of the 37th International Conference on Machine Learning, ICML 2020 (2020)
9. Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., Yan, J.: DARTS-: robustly stepping out of performance collapse without indicators. In: 9th International Conference on Learning Representations, ICLR (2021)
10. Colby, M.K., Kharaghani, S., HolmesParker, C., Tumer, K.: Counterfactual exploration for improving multiagent learning. In: Autonomous Agents and Multiagent Systems (AAMAS 2015), pp. 171–179. International Foundation for Autonomous Agents and Multiagent Systems (2015)
11. Conneau, A., Schwenk, H., Barrault, L., Lecun, Y.: Very deep convolutional networks for text classification. In: European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pp. 1107–1116 (2017)
12. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation policies from data. arXiv:1805.09501 (2018)
13. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition (CVPR), pp. 248–255 (2009)
14. Dong, X., Yang, Y.: Searching for a robust neural architecture in four GPU hours. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR. Computer Vision Foundation / IEEE (2019)
15. Fei-Fei, L., Fergus, R., Perona, P.: Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding* **106**(1), 59–70 (2007)
16. Freedman, D.A.: On tail probabilities for martingales. *The Annals of Probability* pp. 100–118 (1975)
17. Han, D., Kim, J., Kim, J.: Deep pyramidal residual networks. In: Computer Vision and Pattern Recognition (CVPR), pp. 5927–5935 (2017)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
19. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Computer Vision and Pattern Recognition (CVPR), pp. 4700–4708 (2017)
20. Ko, B.: Imagenet classification leaderboard. <https://kobiso.github.io/Computer-Vision-Leaderboard/imagenet> (2019)
21. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)

22. Li, L., Talwalkar, A.: Random search and reproducibility for neural architecture search. *arXiv:1902.07638* (2019)
23. Li, L.J., Fei-Fei, L.: What, where and who? classifying events by scene and object recognition. In: *International Conference on Computer Vision (ICCV)*, pp. 1–8 (2007)
24. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive neural architecture search. In: *European Conference on Computer Vision (ECCV)*, pp. 19–34 (2018)
25. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. In: *International Conference on Learning Representations (ICLR)* (2018)
26. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable architecture search. In: *International Conference on Learning Representations (ICLR)* (2019)
27. Lopes, V., Alexandre, L.A.: Towards less constrained macro-neural architecture search. *arXiv preprint arXiv:2203.05508* (2022)
28. Lopes, V., Alirezazadeh, S., Alexandre, L.A.: EPE-NAS: Efficient performance estimation without training for neural architecture search. In: *International Conference on Artificial Neural Networks* (2021)
29. Lopes, V., Santos, M., Degardin, B., Alexandre, L.A.: Efficient guided evolution for neural architecture search. In: *Proceedings of the Genetic and Evolutionary Computation Conference* (2022)
30. Mellor, J., Turner, J., Storkey, A., Crowley, E.J.: Neural architecture search without training. In: *International Conference on Machine Learning* (2021)
31. Merity, S., Keskar, N.S., Socher, R.: Regularizing and optimizing LSTM language models. In: *International Conference on Learning Representations (ICLR)* (2018)
32. Ning, X., Tang, C., Li, W., Zhou, Z., Liang, S., Yang, H., Wang, Y.: Evaluating efficient performance estimators of neural architectures. *Advances in Neural Information Processing Systems* **34** (2021)
33. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameter sharing. In: *International Conference on Machine Learning (ICML)*, pp. 4092–4101 (2018)
34. Quattoni, A., Torralba, A.: Recognizing indoor scenes. In: *Computer Vision and Pattern Recognition (CVPR)*, pp. 413–420 (2009)
35. Rashid, T., Samvelyan, M., Witt, C.S., Farquhar, G., Foerster, J., Whiteson, S.: QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: *International Conference on Machine Learning (ICML)*, pp. 4292–4301 (2018)
36. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. *arXiv:1802.01548* (2018)
37. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: *International Conference on Machine Learning (ICML)*, pp. 2902–2911 (2017)
38. Ru, R., Esperança, P.M., Carlucci, F.M.: Neural architecture generator optimization. *Advances in Neural Information Processing Systems* **33**, 12057–12069 (2020)

39. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, Inception-ResNet and the impact of residual connections on learning. In: AAAI Conference on Artificial Intelligence (2017)
40. Wan, X., Ru, B., Esperança, P.M., Li, Z.: On redundancy and diversity in cell-based neural architecture search. In: International Conference on Learning Representations (2022)
41. Wan, X., Ru, B., Esperança, P.M., Li, Z.: On redundancy and diversity in cell-based neural architecture search. In: International Conference on Learning Representations (ICLR) (2022)
42. Wang, B., Xue, B., Zhang, M.: Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification. *IEEE transactions on neural networks and learning systems* (2021)
43. Wei, C., Niu, C., Tang, Y., Wang, Y., Hu, H., Liang, J.: Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems* (2022)
44. White, C., Neiswanger, W., Savani, Y.: Bananas: Bayesian optimization with neural architectures for neural architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence (2021)
45. White, C., Zela, A., Ru, R., Liu, Y., Hutter, F.: How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems* **34** (2021)
46. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: Stochastic neural architecture search. In: International Conference on Learning Representations (ICLR) (2019)
47. Yang, A., Esperança, P.M., Carlucci, F.M.: NAS evaluation is frustratingly hard. In: International Conference on Learning Representations (ICLR) (2020)
48. Yang, Y., Luo, R., Li, M., Zhou, M., Zhang, W., Wang, J.: Mean field multi-agent reinforcement learning. In: International Conference on Machine Learning (ICML) (2018)
49. Yao, Q., Xu, J., Tu, W., Zhu, Z.: Efficient neural architecture search via proximal iterations. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI. AAAI Press (2020)
50. Yu, K., Sciuto, C., Jaggi, M., Musat, C., Salzmann, M.: Evaluating the search phase of neural architecture search. In: International Conference on Learning Representations (ICLR) (2019)
51. Zela, A., Elsken, T., Saikia, T., MARRAKCHI, Y., Brox, T., Hutter, F.: Understanding and robustifying differentiable architecture search. In: International Conference on Learning Representations (ICLR) (2020)
52. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6848–6856 (2018)
53. Zoph, B., Le, Q.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (ICLR) (2017)

54. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Computer Vision and Pattern Recognition (CVPR), pp. 8697–8710 (2018)

A Datasets

CIFAR-10. The CIFAR-10 dataset [21] is a dataset with 10 classes and consists of 50,000 training images and 10,000 test images of size 32×32 . We use standard data pre-processing and augmentation techniques, i.e. subtracting the channel mean and dividing the channel standard deviation; centrally padding the training images to 40×40 and randomly cropping them back to 32×32 ; and randomly flipping them horizontally.

ImageNet. The ImageNet dataset [13] is a dataset with 1000 classes and consists of 1,281,167 training images and 50,000 test images of different sizes. We use standard data pre-processing and augmentation techniques, i.e. subtracting the channel mean and dividing the channel standard deviation, cropping the training images to random size and aspect ratio, resizing them to 224×224 , and randomly changing their brightness, contrast, and saturation, while resizing test images to 256×256 and cropping them at the center.

Sport-8. This is an action recognition dataset containing 8 sport event categories and a total of 1579 images [23]. The tiny size of this dataset stresses the generalization capabilities of any NAS method applied to it.

Caltech-101. This dataset contains 101 categories, each with 40 to 800 images of size roughly 300×200 [15].

MIT-67. This is a dataset of 67 classes representing different indoor scenes and consists of 15,620 images of different sizes [34].

In experiments on Sport-8, Caltech-101 and MIT-67, we split each dataset into a training set containing 80% of the data and a test set containing 20% of the data. For each of them, we use the same data pre-processing techniques as for ImageNet.

B Implementation details

B.1 Methods

MANAS. Our code is based on a modified variant of [26]. To set the temperature and gamma, we used as starting estimates the values suggested by [5]: $t = \frac{1}{\eta}$ with $\eta = 0.95 \frac{\sqrt{\ln(K)}}{nK}$ (K number of actions, n number of architectures seen in the whole training); and $\gamma = 1.05 \frac{K \ln(K)}{n}$. We then tuned them to increase validation accuracy during the search.

MANAS-LS. For our Least-Squares solution, we alternate between one epoch of training (in which all β are frozen and the ω are updated) and one or more epochs in which we build the Z matrix from Section 4 (in which both β and ω are frozen). The exact number of iterations we perform in this latter step is dependant on the size of both the dataset and the searched architecture: our goal is simply to have a number of rows greater than the number of columns for Z . We then solve $\tilde{\mathbf{B}}_t = (\mathbf{Z}\mathbf{Z}^\top)^\dagger \mathbf{Z}\mathbf{L}$, and repeat the whole procedure until the end of training. This method requires no additional meta-parameters.

Number of agents. In both MANAS variants, the number of agents is defined by the search space and thus is not tuned. Specifically, for the image datasets, there exists one agent for each pair of nodes, tasked with selecting the optimal operation. As there are 14 pairs in each cell, the total number of agents is $14 \times C$, with C being the number of cells (8, 14 or 20, depending on the experiment).

B.2 Computational resources

ImageNet experiments were performed on multi-GPU machines loaded with $8 \times$ Nvidia Tesla V100 16GB GPUs (used in parallel). All other experiments were performed on single-GPU machines loaded with $1 \times$ GeForce GTX 1080 8GB GPU.

C Factorizing the Regret

Factorizing the Regret. Let us firstly formulate the multi-agent combinatorial online learning in a more formal way. Recall, at each round, agent \mathcal{A}_i samples an action from a fixed discrete collection $\{\mathbf{a}_j^{(\mathcal{A}_i)}\}_{j=1}^K$. Therefore, after each agent makes a choice of its action at round t , the resulting network architecture \mathcal{Z}_t is described by joint action profile $\vec{\mathbf{a}}_t = [\mathbf{a}_{j_1}^{(\mathcal{A}_1), [t]}, \dots, \mathbf{a}_{j_N}^{(\mathcal{A}_N), [t]}]$ and thus, we will use \mathcal{Z}_t and $\vec{\mathbf{a}}_t$ interchangeably. Due to the discrete nature of the joint action space, the validation loss vector at round t is given by $\vec{\mathcal{L}}_t^{(\text{val})} = \left(\mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t^{(1)}), \dots, \mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t^{(K^N)}) \right)$ and for the environment one can write $\nu = \left(\vec{\mathcal{L}}_1^{(\text{val})}, \dots, \vec{\mathcal{L}}_T^{(\text{val})} \right)$. The interconnection between joint policy π and an environment ν works in a sequential manner as follows: at round t , the architecture $\mathcal{Z}_t \sim \pi_t(\cdot | \mathcal{Z}_1, \mathcal{L}_1^{(\text{val})}, \dots, \mathcal{Z}_{t-1}, \mathcal{L}_{t-1}^{(\text{val})})$ is sampled and validation loss $\mathcal{L}_t^{(\text{val})} = \mathcal{L}_t^{(\text{val})}(\mathcal{Z}_t)$ is observed¹. As we mentioned previously, assuming linear contribution of each individual actions to the validating loss, one goal is to find a policy π that keeps the regret:

$$\mathcal{R}_T(\pi, \nu) = \mathbb{E} \left[\sum_{t=1}^T \beta_t^\top \mathbf{Z}_t - \min_{\mathbf{Z} \in \mathcal{F}} \left[\sum_{t=1}^T \beta_t^\top \mathbf{Z} \right] \right] \quad (8)$$

small with respect to all possible forms of environment ν . We reason here with the cumulative regret the reasoning applies as well to the simple regret. Here, $\beta_t \in \mathbb{R}_+^{K^N}$ is a contribution vector of all actions and \mathbf{Z}_t is binary representation of architecture \mathcal{Z}_t and $\mathcal{F} \subset [0, 1]^{K^N}$ is set of all feasible architectures². In other words, the quality of the policy is defined with respect to worst-case regret:

$$\mathcal{R}_T^* = \sup_{\nu} \mathcal{R}_T(\pi, \nu) \quad (9)$$

¹ Please notice, the observed reward is actually a random variable

² We assume that architecture is feasible if and only if each agent chooses exactly one action.

Notice, that linear decomposition of the validation loss allows to rewrite the total regret (8) as a sum of agent-specific regret expressions $\mathcal{R}_T^{(\mathcal{A}_i)}(\boldsymbol{\pi}^{(\mathcal{A}_i)}, \nu^{(\mathcal{A}_i)})$ for $i = 1, \dots, N$:

$$\begin{aligned} \mathcal{R}_T(\boldsymbol{\pi}, \nu) &= \mathbb{E} \left[\sum_{t=1}^T \left(\sum_{i=1}^N \beta_t^{(\mathcal{A}_i), \top} \mathbf{Z}_t^{(\mathcal{A}_i)} - \sum_{i=1}^N \min_{\mathbf{Z}^{(\mathcal{A}_i)} \in \mathcal{B}_{\|\cdot\|_0, 1}^{(K)}(\mathbf{0})} \left[\sum_{t=1}^T \beta_t^{(\mathcal{A}_i), \top} \mathbf{Z}^{(\mathcal{A}_i)} \right] \right) \right] \\ &= \sum_{i=1}^N \mathbb{E} \left[\sum_{t=1}^T \beta_t^{(\mathcal{A}_i), \top} \mathbf{Z}_t^{(\mathcal{A}_i)} - \min_{\mathbf{Z}^{(\mathcal{A}_i)} \in \mathcal{B}_{\|\cdot\|_0, 1}^{(K)}(\mathbf{0})} \left[\sum_{t=1}^T \beta_t^{(\mathcal{A}_i), \top} \mathbf{Z}^{(\mathcal{A}_i)} \right] \right] \\ &= \sum_{i=1}^N \mathcal{R}_T^{(\mathcal{A}_i)}(\boldsymbol{\pi}^{(\mathcal{A}_i)}, \nu^{(\mathcal{A}_i)}) \end{aligned}$$

where $\boldsymbol{\beta}_t = [\beta_t^{(\mathcal{A}_1), \top}, \dots, \beta_t^{(\mathcal{A}_N), \top}]^\top$ and $\mathbf{Z}_t = [\mathbf{Z}_t^{(\mathcal{A}_1), \top}, \dots, \mathbf{Z}_t^{(\mathcal{A}_N), \top}]^\top$, $\mathbf{Z} = [\mathbf{Z}^{(\mathcal{A}_1), \top}, \dots, \mathbf{Z}^{(\mathcal{A}_N), \top}]^\top$ are decomposition of the corresponding vectors on agent-specific parts, joint policy $\boldsymbol{\pi}(\cdot) = \prod_{i=1}^N \boldsymbol{\pi}^{(\mathcal{A}_i)}(\cdot)$, and joint environment $\nu = \prod_{i=1}^N \nu^{(\mathcal{A}_i)}$, and $\mathcal{B}_{\|\cdot\|_0, 1}^{(K)}(\mathbf{0})$ is unit ball with respect to $\|\cdot\|_0$ norm centered at $\mathbf{0}$ in $[0, 1]^K$. Moreover, the worst-case regret (9) also can be decomposed into agent-specific form:

$$\mathcal{R}_T^* = \sup_{\nu} \mathcal{R}_T(\boldsymbol{\pi}, \nu) \iff \sup_{\nu^{(\mathcal{A}_i)}} \mathcal{R}_T^{(\mathcal{A}_i)}(\boldsymbol{\pi}^{(\mathcal{A}_i)}, \nu^{(\mathcal{A}_i)}), \quad i = 1, \dots, N.$$

This decomposition allows us to significantly reduce the search space and apply the two following algorithms for each agent \mathcal{A}_i in a completely parallel fashion.

D Theoretical Guarantees

D.1 MANAS-LS

First, we need to be more specific on the way to obtain the estimates $\tilde{\beta}_t^{(\mathcal{A}_i)}[k]$.

In order to obtain theoretical guaranties we considered the least-square estimates as in [7] as

$$\tilde{\beta}_t = \mathcal{L}_t^{(\text{val})} \mathbf{P}^\dagger \mathbf{Z}_t \text{ where } \mathbf{P} = \mathbb{E} [\mathbf{Z} \mathbf{Z}^\top] \text{ with } \mathbf{Z} \text{ has law } \boldsymbol{\pi}_t(\cdot) = \prod_{i=1}^N \boldsymbol{\pi}_t^{(\mathcal{A}_i)}(\cdot) \quad (10)$$

Our analysis is under the assumption that each $\beta_t \in \mathbb{R}^{KN}$ belongs to the linear space spanned by the space of sparse architecture \mathcal{Z} . This is not a strong assumption as the only condition on a sparse architecture comes with the sole restriction that one operation for each agent is active.

Theorem 1 *Let us consider neural architecture search problem in a multi-agent combinatorial online learning form with N agents such that each agent has K actions. Then after T rounds, MANAS-LS achieves joint policy $\{\boldsymbol{\pi}_t\}_{t=1}^T$ with expected simple regret (Equation 3) bounded by $\mathcal{O}(e^{-T/H})$ in any adversarial environment with complexity bounded by $H = N(\min_{j \neq k_i^*, i \in \{1, \dots, N\}} \mathbf{B}_T^{(\mathcal{A}_i)}[j] - \mathbf{B}_T^{(\mathcal{A}_i)}[k_i^*])$, where $k_i^* = \min_{j \in \{1, \dots, K\}} \mathbf{B}_T^{(\mathcal{A}_i)}[j]$.*

Proof In Equation 10 we use the same constructions of estimates $\tilde{\beta}_t$ as in ComBand. Using Corollary 14 in [7] we then have that $\tilde{\mathbf{B}}_t$ is an unbiased estimates of \mathbf{B}_t .

Given the adversary losses, the random variables $\tilde{\beta}_t$ can be dependent of each other and $t \in [T]$ as $\boldsymbol{\pi}_t$ depends on previous observations at previous rounds. Therefore, we use the Azuma inequality for martingale differences by [16].

Without loss of generality we assume that the loss $\mathcal{L}_t^{(\text{val})}$ are bounded such that $\mathcal{L}_t^{(\text{val})} \in [0, 1]$ for all t . Therefore we can bound the simple regret of each agent by the probability of misidentifying of the best operation $P(k_i^* \neq a_{T+1}^{\mathcal{A}_i})$.

We consider a fixed adversary of complexity bounded by H . For simplicity, and without loss of generality, we order the operations from such that $\mathbf{B}_T^{(\mathcal{A}_i)}[1] < \mathbf{B}_T^{(\mathcal{A}_i)}[2] \leq \dots \leq \mathbf{B}_T^{(\mathcal{A}_i)}[K]$ for all agents.

We denote for $k > 1$, $\Delta_k = \mathbf{B}_T^{(\mathcal{A}_i)}[k] - \mathbf{B}_T^{(\mathcal{A}_i)}[k_i^*]$ and $\Delta_1 = \Delta_2$.

We also have λ_{\min} as the smallest nonzero eigenvalue of \mathbf{M} where \mathbf{M} is $\mathbf{M} = E[\mathbf{Z}\mathbf{Z}^T]$ where \mathbf{Z} is a random vector representing a sparse architecture distributed according to the uniform distribution.

$$\begin{aligned} P(k_i^* \neq a_{T+1}^{\mathcal{A}_i}) &= P\left(\exists k \in \{1, \dots, K\} : \tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[1] \geq \tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[k]\right) \\ &\leq P\left(\exists k \in \{1, \dots, K\} : \mathbf{B}_T^{(\mathcal{A}_i)}[k] - \tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[k] \geq \frac{T\Delta_k}{2} \text{ or } \tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[1] - \mathbf{B}_T^{(\mathcal{A}_i)}[1] \geq \frac{T\Delta_1}{2}\right) \\ &\leq P\left(\tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[1] - \mathbf{B}_T^{(\mathcal{A}_i)}[1] \geq \frac{T\Delta_1}{2}\right) + \sum_{k=2}^K P\left(\mathbf{B}_T^{(\mathcal{A}_i)}[k] - \tilde{\mathbf{B}}_T^{(\mathcal{A}_i)}[k] \geq \frac{T\Delta_k}{2}\right) \\ &\stackrel{\text{(a)}}{\leq} \sum_{k=1}^K \exp\left(-\frac{(\Delta_k)^2 T}{2N \log(K)/\lambda_{\min}}\right) \\ &\leq K \exp\left(-\frac{(\Delta_1)^2 T}{2N \log(K)/\lambda_{\min}}\right), \end{aligned}$$

where (a) is using Azuma's inequality for martingales applied to the sum of the random variables with mean zero that are $\tilde{\beta}_{k,t} - \beta_{k,t}$ for which we have the following bounds on the range. The range of $\tilde{\beta}_{k,t}$ is $[0, N \log(K)/\lambda_{\min}]$. Indeed our sampling policy is uniform with probability $1/\log(K)$ therefore one can bound $\tilde{\beta}_{k,t}$ as in [7, Theorem 1] Therefore we have $|\tilde{\beta}_{k,t} - \beta_{k,t}| \leq N \log(K)/\lambda_{\min}$.

We recover the result with a union bound on all agents.

D.2 MANAS

We consider a simplified notion of regret that is a regret per agent where each agent is considering the rest of the agents as part of the adversarial environment. Let us fix our new objective as to minimise

$$\sum_{i=1}^N \mathcal{R}_T^{*,i}(\pi^{\mathcal{A}_i}) = \sum_{i=1}^N \sup_{\mathbf{a}_{-i}, \nu} \mathbb{E} \left[\sum_{t=1}^T \mathcal{L}_t^{(\text{val})}(\mathbf{a}_t^{\mathcal{A}_i}, \mathbf{a}_{-i}) - \min_{\mathbf{a} \in \{1, \dots, K\}} \left[\sum_{t=1}^T \mathcal{L}_t^{(\text{val})}(\mathbf{a}, \mathbf{a}_{-i}) \right] \right],$$

where \mathbf{a}_{-i} is a fixed set of actions played by all agents to the exception of agent \mathcal{A}_i for the T rounds of the game and ν contains all the losses as $\nu = \{\mathcal{L}_t^{(\text{val})}(\mathbf{a})\}_{t \in \{1, \dots, T\}, \mathbf{a} \in \{1, \dots, K^N\}}$.

We then can prove the following bound for that new notion of regret.

Theorem 2 *Let us consider neural architecture search problem in a multi-agent combinatorial online learning form with N agents such that each agent has K actions. Then after T rounds, MANAS achieves joint policy $\{\pi_t\}_{t=1}^T$ with expected cumulative regret bounded by $\mathcal{O}(N\sqrt{TK \log K})$.*

Proof First we look at the problem for each given agent \mathcal{A}_i and we define and look at

$$\mathcal{R}_T^{*,i}(\pi^{\mathcal{A}_i}, \mathbf{a}_{-i}) = \sup_{\nu} \mathbb{E} \left[\sum_{t=1}^T \mathcal{L}_t^{(\text{val})}(\mathbf{a}_t^{\mathcal{A}_i}, \mathbf{a}_{-i}) - \min_{\mathbf{a} \in \{1, \dots, K\}} \left[\sum_{t=1}^T \mathcal{L}_t^{(\text{val})}(\mathbf{a}, \mathbf{a}_{-i}) \right] \right],$$

We want to relate that the game that agent i plays against an adversary when the actions of all the other agents are fixed to \mathbf{a}_{-i} to the vanilla EXP3 setting. To be more precise on why this is the EXP3 setting, first we have that $\mathcal{L}_t^{(\text{val})}(\mathbf{a}_t)$ is a function of \mathbf{a}_t that can take K^N arbitrary values. When we fix \mathbf{a}_{-i} , $\mathcal{L}_t^{(\text{val})}(\mathbf{a}_t^{(\mathcal{A}_i)}, \mathbf{a}_{-i})$ is a function of $\mathbf{a}_t^{(\mathcal{A}_i)}$ that can only take K arbitrary values.

One can redefine $\mathcal{L}_t^{\text{val}}(\mathbf{a}_t^{(\mathcal{A}_i)}) = \mathcal{L}_t^{(\text{val})}(\mathbf{a}_t^{(\mathcal{A}_i)}, \mathbf{a}_{-i})$ and then the game boils down to the vanilla adversarial multi-arm bandit where each time the learner plays $\mathbf{a}_t^{(\mathcal{A}_i)} \in \{1, \dots, K\}$ and observes/incurs the loss $\mathcal{L}_t^{\text{val}}(\mathbf{a}_t^{(\mathcal{A}_i)})$. Said differently this defines a game where the new ν' contains all the losses as $\nu' = \{\mathcal{L}_t^{\text{val}}(\mathbf{a}_t^{(\mathcal{A}_i)})\}_{t \in \{1, \dots, T\}, \mathbf{a}_t^{(\mathcal{A}_i)} \in \{1, \dots, K\}}$.

For all \mathbf{a}_{-i}

$$\mathcal{R}_T^{*,i}(\text{EXP3}, \mathbf{a}_{-i}) \leq 2\sqrt{TK \log(K)}$$

Then we have

$$\begin{aligned} \mathcal{R}_T^{*,i}(\text{EXP3}) &\leq \sup_{\mathbf{a}_{-i}} 2\sqrt{TK \log(K)} \\ &= 2\sqrt{TK \log(K)} \end{aligned}$$

Then we have

$$\sum_{i=1}^N \mathcal{R}_T^{*,i}(\text{EXP3}) \leq 2N\sqrt{TK \log(K)}$$

E Relation between weight sharing and cumulative regret

Ideally we would like to obtain for any given architecture \mathcal{Z} the value $\mathcal{L}_{\text{val}}(\mathcal{Z}, \mathbf{w}^*(\mathcal{Z}))$. However obtaining $\mathbf{w}^*(\mathcal{Z}) = \arg \min_{\mathbf{w}} \mathcal{L}_{\text{train}}(\mathbf{w}, \mathcal{Z})$ for any given fixed \mathcal{Z} would already require heavy computations. In our approach the \mathbf{w}_t that we compute and update is actually common to all \mathcal{Z}_t as \mathbf{w}_t replaces $\mathbf{w}^*(\mathcal{Z}_t)$. This is a simplification that leads to learning a weight \mathbf{w}_t that tend to minimise the loss $\mathbb{E}_{\mathcal{Z} \sim \pi_t} [\mathcal{L}_{\text{val}}(\mathcal{Z}, \mathbf{w}(\mathcal{Z}))]$ instead of minimising $\mathcal{L}_{\text{val}}(\mathcal{Z}_t, \mathbf{w}(\mathcal{Z}_t))$. If π_t is concentrated on a fixed \mathcal{Z} then these two previous expressions would be close. Moreover when π_t is concentrated on \mathcal{Z} then \mathbf{w}_t will approximate accurately $\mathbf{w}^*(\mathcal{Z})$ after a few steps. Note that this gives an argument for using sampling algorithm that minimise the cumulative regret as they naturally tend to play almost all the time one specific architecture. However there is a potential pitfall of converging to a local minimal solution as \mathbf{w}_t might not have learned well enough to compute accurately the loss of other and potentially better architectures.