OFA³: Automatic Selection of the Best Non-dominated Sub-networks for Ensembles

Anonymous[1]

[1]Anonymous Institution

Abstract Advancement of Neural Architecture Search (NAS) has the potential to significantly improve the efficiency and performance of machine learning systems, as well as enable the exploration of new architectures and applications across a wide range of fields, including computer vision, natural language processing, speech recognition, robotics, and more. A promising direction for developing more scalable and adaptive neural network architectures is the Once-for-All (OFA), a NAS framework that decouples the training and the search stages, meaning that one super-network is trained once, and then multiple searches can be performed according to different deployment scenarios. More recently, the OFA² strategy improved the search stage of the OFA framework by exploring the multi-objective nature of the problem: a set of non-dominated sub-networks are all obtained at once, with distinct trade-offs involving hardware constraints and accuracy. In this work, we propose OFA³, building high-performance ensembles by solving the problem of how to automatically select the optimal subset of the already obtained non-dominated sub-networks. Particularly when components of the ensemble can run in parallel, our results dominate any other configuration of the available sub-networks, taking accuracy and latency as the conflicting objectives.

1 Introduction

The task of designing neural networks requires a lot of trial and error, as well as a deep understanding of the problem at hand. And even then, there is no guarantee that we will arrive at the optimal architecture. Searching for a way to automate the designing of artificial neural architectures arises the Neural Architecture Search (NAS) framework [1]. In other words, it is an algorithm that can automatically find the optimal neural network architecture for a given task without human intervention. By automating the process of designing neural networks, we can save a lot of time and resources while potentially arriving at more efficient and effective architectures. NAS works by searching through a vast space of possible architectures, evaluating their performance, and iteratively refining the search until it finds the best architecture for the given task [1].

To find the best architecture, NAS frameworks use advanced techniques such as reinforcement learning [2], evolutionary algorithms [3], and gradient-based optimization [4] to search through the space of possible architectures more efficiently and effectively. Additionally, By combining these searched models, it is possible to create a more comprehensive representation that captures multiple levels of abstraction. That is called Ensemble learning, which can help increase the diversity of the architectures that are generated by the NAS process [1]. By combining multiple architectures, the ensemble can explore a wider range of design choices and trade-offs, potentially leading to better overall performance [5]. However, when one considers real-world scenarios, there are often multiple objectives that need to be optimized simultaneously, such as performance, model size, latency, and deployment.

A way to address such a challenge is the Multi-objective Neural Network Search (MO-NAS) which uses a multi-objective optimization approach that considers multiple conflicting objectives simultaneously [6]. MO-NAS typically works by generating a set of candidate architectures that trade-off between different objectives, using techniques such as evolutionary algorithms [3] or

 © 2023 the authors, released under CC BY 4.0

Pareto optimization [7]. These candidate architectures are evaluated on a validation set, and the ones that are deemed Pareto-optimal, meaning they cannot be improved on any objective without sacrificing performance on another, are considered the solution and returned to the user.

MO-NAS has the advantage of allowing us to explore a wider range of trade-offs between different objectives. For instance, we might be able to find architectures that are both highly accurate and efficient [8], or architectures that can perform well on multiple related tasks simultaneously [9]. Also, it can help identify potential trade-offs between objectives, allowing researchers and practitioners to make more informed decisions about the design of their neural networks.

Another factor is the computational cost of generating the optimal architecture. The search process can be computationally expensive and time-consuming, requiring significant resources and infrastructure. Therefore, there is a gain in considering the deployment scenario and available resources when designing the search process. In OFA^2 it was presented a multiobjective search with the evolutionary algorithm NSGA-II [10], for both accuracy and latency, consistently outperforms the architectures from random search. The work not only finds better architectures in terms of top-1 accuracy and latency but also returns a set of solutions instead of a single one, each of them being optimal considering a specific trade-off among the objective functions . In this work, we propose OFA^3, a technique to automatically select the best sub-networks among a population of non-dominated architectures to form efficient ensembles. We also provide experiments regarding the latency of the ensemble considering the scenarios of sum and maximum latencies. We show that when the components of the ensemble can run in parallel, thus considering the maximum latency scenario, our results dominate any other configuration of the available single neural networks, resulting in better architectures overall.

2 Related Works

Due to the diverse composition of our work, we discuss the background accordingly to some of the algorithm classes involved.

Neural Architecture Search. The research field of automating the machine learning process [11], including selecting algorithms, hyperparameters, and architectures, dates back to the early 2000s [12]. However, the focus of those works was mostly on the algorithm and hyperparameter selection, and NAS was not yet a major research topic [1]. In 2019, two new NAS methods were proposed, called Efficient Neural Architecture Search (ENAS) [8] and Once-for-All (OFA) [13]. ENAS involved using a shared weight scheme to reduce the search space, while OFA involved training a large pre-defined network that can be efficiently adapted to different tasks and hardware platforms. ENAS approaches, by sharing weight to reduce the number of parameters, significantly reduce the time and cost required for NAS. While ENAS is computationally efficient, it can be limited by the search space. Meanwhile, OFA can find a family of architectures that can perform well across various resource constraints. Their weakness is that it can be challenging to scale to larger datasets or more complex models. Nowadays, NAS has advanced significantly with the development of new search algorithms and optimization techniques, with the advance in automating the architecture design process, it is possible to find an expressive number of surveys on NAS [1] [3] [14] [15].

Multi-objective optimization approach. Emerged as a promising path to address the challenges of traditional NAS methods (e.g., search space, computational cost, generalization, evaluation, interpretability, among others) is the multi-objective optimization approach. Accordingly, it was intuitive to resort to evolutionary multi-objective optimization (EMO) algorithms as in [10] [16]. Yet, using EMO algorithms for NAS still lacks complex characteristics (e.g., multiple modalities, many objectives) since there is no general formulation and analysis of NAS from the optimization point of view and implemented environments to perform MOPs with NAS [17]. However, until the late 2010s, multi-objective optimization gained popularity in NAS, and since then, several

multiobjective optimization approaches have been proposed, including NSGA-Net [18], DARTS+ [19], and MOEAs [9].

Evolutionary algorithms. In the mid to late 2010s, researchers started exploring the use of reinforcement learning and evolutionary algorithms for NAS. Works like in [20] [21] [22] guide the mutation and recombination process of candidate architectures as the evolution process. Some methods [23] [24] [25] involved training a controller or a population of architectures using evolutionary algorithms and evaluating the performance of the generated architectures trying to improve by exploring the search space incrementally and sequentially, typically from shallow to hard, in the validation set. Another works [26] [27] [28], tries to improve the quality of discrete decisions in the process of searching architectures by utilizing a performance predictor to select promising candidates.

To our knowledge, the existing techniques cannot be scaled to many different deployment scenarios. That means the architectural search process must be repeated, and the model needs retraining in light of new inference hardware platforms. Also, the overall model size is huge, and the download bandwidth is considerable because the separately trained models do not share weights.

3 Methodology

In the traditional Neural Architecture Search (NAS) pipeline, we have three main components: search space, search strategy and performance estimation strategy [29]. We first choose or propose a search space that defines all neural networks architectures available to the framework. Then, we choose a search strategy that will guide the exploration and exploitation of the search space. Finally, a performance estimator measures how good the architecture found is and helps to guide the search towards better architectures.

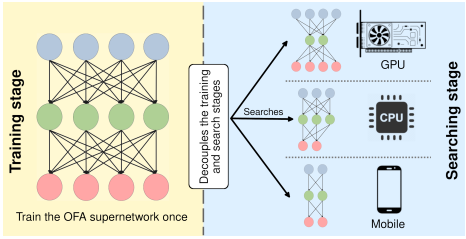


Figure 1: Once-for-All (OFA) framework overview. A single OFA supernetwork is trained only once, and then multiple searches can be performed according to the different deployment scenarios.

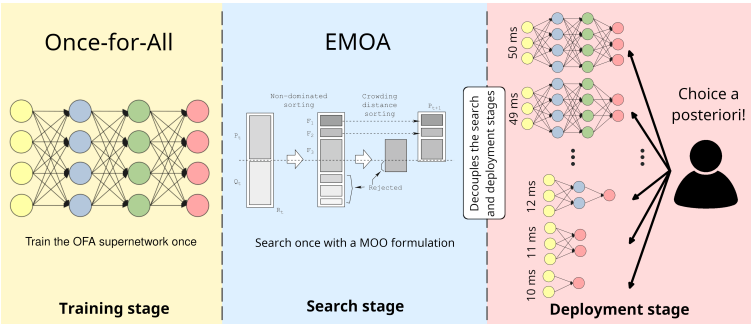


Figure 2: OFA² overview. Instead of manually performing one search for each deployment scenario, an EMOA performs the search of the architectures for all deployment scenarios at once.

OFA. The Once-for-All (OFA) NAS framework [13] works a bit differently of the traditional NAS pipeline. More specifically, the OFA framework decoupled the training and search stages of the NAS pipeline. The biggest advantage of this decoupling is that a single supernetwork is trained only once, and then multiple searches can be done in this supernetwork finding smaller sub-networks according to different deployment scenarios. The deployment scenario can be defined by a hardware constraint, such as latency or FLOPS. Figure 1 illustrates the OFA framework.

OFA². More recently, a new search strategy called OFA² [30] was proposed for the OFA framework. Instead of performing multiple searches on the OFA supernetwork, the OFA² optimizes the search stage of the OFA framework by solving it as multi-objective optimization problem (MOOP) using an evolutionary multi-objective optimization algorithm (EMOA). In other words, a single search is enough to find a representative set of architectures with distinct trade-offs among the objective functions optimized, that being latency and accuracy. Figure 2 illustrates the OFA² framework.

OFA³. Both the OFA and the OFA² consider only single architectures during the search stage of the framework. In this work we aim to give one step further considering the search stage of the framework by proposing a strategy to automatically select the networks found by the OFA² search to form efficient ensembles. The approach used to solve this problem is similar to the one used in the OFA², that is, we use an EMOA to jointly optimize the accuracy and latency, by maximizing the former and minimizing the latter. The OFA³ does not focus on the search stage of the framework. Instead, we use the neural networks found by OFA² during the search stage, and the problem now is how to select among these architectures the best set to form ensembles giving a hardware restriction.

Ensembles. Ensemble [31] is a technique where we combine predictions from more than one model to produce an ideally better output prediction. In this work, we propose to solve the problem of determining the number of components in an ensemble and the components themselves as a multi-objective optimization problem. The idea is to find a set of neural networks whose individuals complement each other [32], reducing the variance of the error at the output, thus leading to better machine learning models [33]. The idea of using an evolutionary algorithm to obtain ensembles is not innovative [34]. However, most of the approaches that proposes it, use the evolutionary process just to create diversity among the individuals of the ensemble and does not propose the ensemble formation problem as a multi-objective problem itself [35] [36].

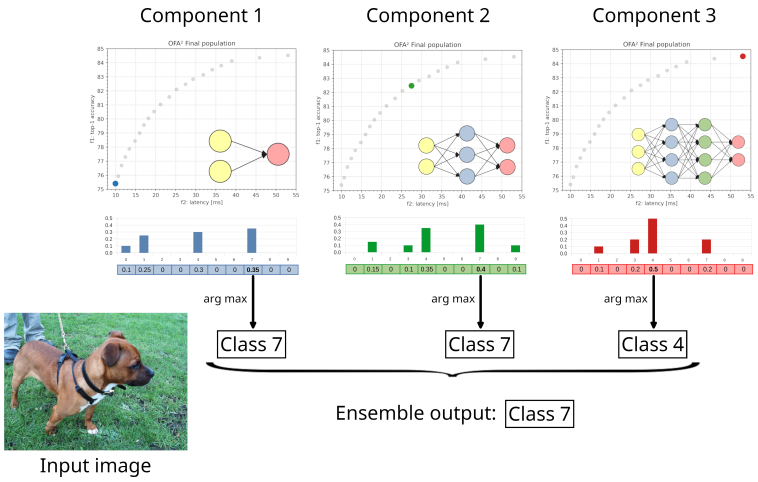


Figure 3: Hard voting scheme for majority voting ensembles.

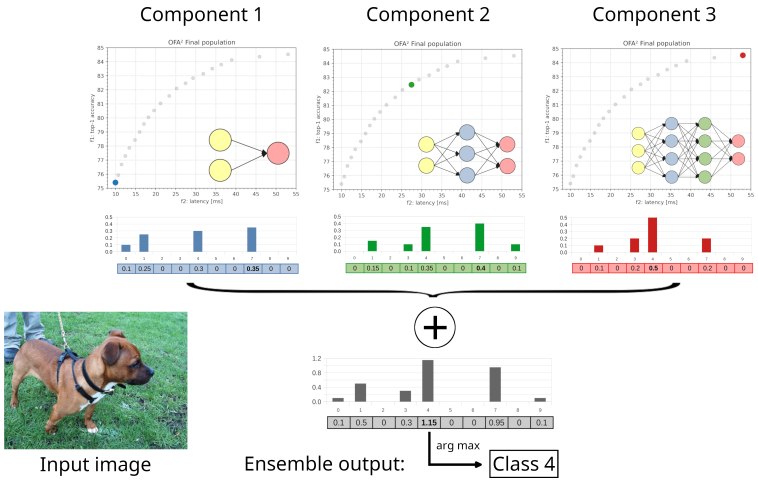


Figure 4: Soft voting scheme for majority voting ensembles.

Voting schemes. The simplest way to determine the output of an ensemble is by majority vote. Considering the majority vote ensembles, there are two main approaches for classification problems: the hard voting and the soft voting. In the hard voting, the output of the ensemble is decided according to the most voted top-1 class among the participants of the ensemble. Draw in votes between more than one class must be handled somehow, for example by checking the occurrences of these classes on the second most likely output of each model (top-2 output) and deciding by the most frequent. If there is still a draw in votes, we can keep checking the top-3, top-4, and so on, outputs. In this scheme, the output of each neural network has the same importance, regardless how certain the model is about its output. Figure 3 illustrates the hard voting scheme considering an ensemble with 3 components and an image classification problem with 10 classes. The soft voting scheme, on the other hand, takes into account the probability assigned to each class on the output. For this, in case the last layer of the neural networks is not a softmax already (like in the case of the OFA supernetwork), this layer is appended to the last layer of the neural networks. In order to decide the output of the ensemble, we sum the probabilities for each class among all participants of the ensemble, and then take the output with the highest accumulated value. Figure 4 illustrates the soft voting scheme. This helps to alleviate the problem of the hard voting scheme, which is the fact that a vote from a model with a low confidence in its output has the same weight of a vote from a model with a high confidence. This voting scheme provides a way to weight the vote of each architecture according to the confidence of the model in its output class, which can be beneficial in some cases.

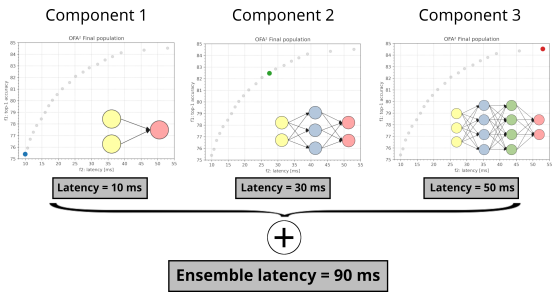


Figure 5: Scenario considering the sum of latency.

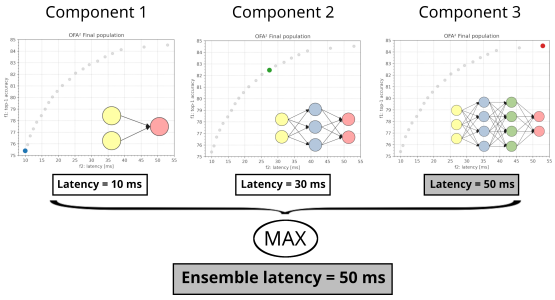


Figure 6: Scenario considering the maximum latency.

Dataset. The dataset used for training the OFA supernetwork, used for the search stage of the OFA² strategy, and used for the ensemble search of the OFA³ is the ImageNet [37], a standard dataset in the computer vision area that consists of 1,281,167 images in the training set, 50,000 images in the validation set and 100,000 images in the test set, organized in 1,000 categories. The training set was used to train the OFA supernetwork. A subset of 10k images of the training set was set apart as a holdout validation set to train the accuracy predictor of the OFA framework, used during the OFA² search. The 50k validation set is used to search the ensembles during the OFA³ multi-objective optimization.

Objective functions. The objective functions optimized are the same as the OFA² search: accuracy and latency. For the accuracy, we use the performance of the ensembles on the validation set of ImageNet, considering the soft voting scheme. For the latency, we use the latency predictor provided by the OFA framework as a latency lookup table [38].
Latency. We consider two approaches with respect to the latency of the ensembles: the sum and the maximum latency. In the first approach, the latency of the ensemble is defined to be the sum of all architectures participating in the ensemble. This strategy is based on the premise that we have a single hardware with limited amount of memory to implement the ensemble, and therefore we need to load each model one at a time to evaluate its performance. Figure 5 illustrates this scenario considering an ensemble with 3 neural networks. In the second approach, the latency of the ensemble is equal to the model's latency that has the highest value among those networks participating of the ensemble. This strategy is based on the premise that parallelization is viable, and therefore all models can be evaluated simultaneously. Figure 6 illustrates this scenario considering the same ensemble with 3 architectures.

Candidate architectures. The neural networks candidates to participate on the ensembles are the 100 architectures obtained from the OFA² search. Figure 7 shows these 100 architectures. The dots in blue corresponds to the performance of the neural networks with the accuracy predictor (trained on a holdout set of the training set of the ImageNet) and the dots in orange represents the performance of the same neural networks evaluated on the 50k validation set of ImageNet.

4 Experiments & Results

Encoding. The encoding used is a simple array with 100 binary genes, one for each neural network from the pool of 100 candidate architectures to form the ensemble, meaning that if a specific gene has a value 0, then the neural network represented by that gene does not participate of the ensemble, and if the gene has a value of 1, then the neural network at that index participates on

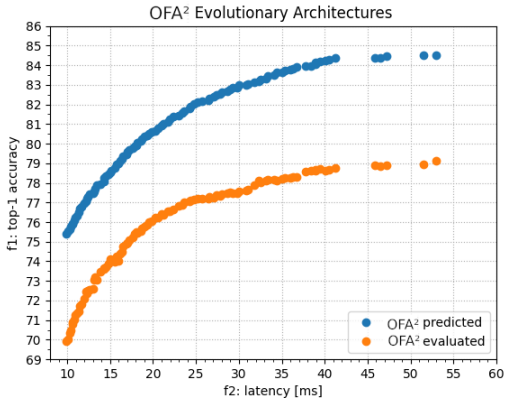


Figure 7: OFA² predicted and evaluated architectures.

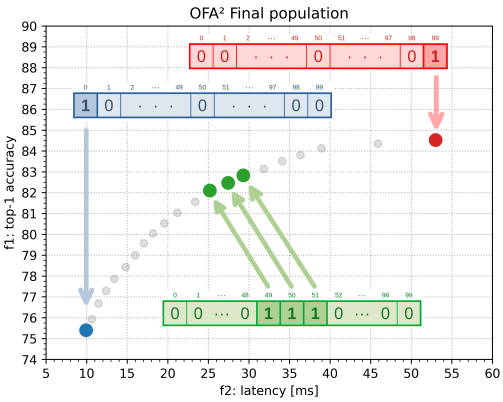


Figure 8: Encoding used to search the ensembles.

the ensemble. Figure 8 illustrates three examples of encodings and their respective sets of neural networks participating of the ensembles. The encoding in blue has a single gene with the value 1 in its first position, meaning that only the first neural network of the population will participate of that ensemble. The encoding in green has three consecutive genes with the value of 1 in the middle of the array, meaning that the three neural networks represented by these genes will take part in the ensemble. The encoding in red has all genes at value 0, except the last one, meaning that only the last neural network of the population participates of this ensemble.

Algorithm. The algorithm chosen is the NSGA-II [10], although any other EMOA algorithm could also be used. Four operators are used during the iterations of evolutionary algorithms: sampling, mutation, crossover and selection. The sampling operator is related to the initialization of the algorithm, and we decided to start it with all individuals of the population having all their 100 genes set to one. This means that all neural networks candidates participates on the ensemble at the initial population, and the reason this configuration of encoding was chosen is to force the algorithm to search efficient ensembles, since this start is the most inefficient possible. The mutation operator used is the bitflip with probability of 1%. The crossover operator (also known as recombination) used is the uniform crossover, which means that the value of each gene of the child solution is randomly taken from one of the parents' solution with equal probability. Finally, the selection operator defines a criterion for choosing the individuals of the current population that will be used to generate the offspring, that is, the next generation of individuals. Since we are using the NSGA-II algorithm, the selection is based on non-dominated sorting and crowding distance metrics. The algorithm ran for a total of 1,000 generations and the results are discussed in the next section.

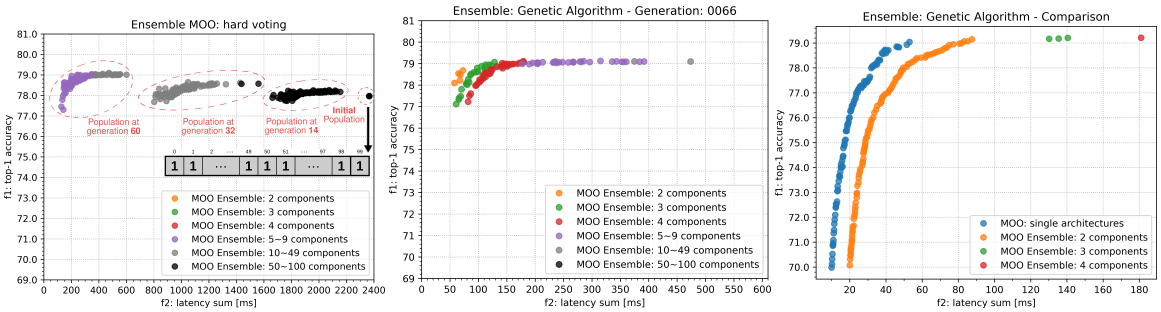
Other details. The implementation of the code was done with the `pymoo` multi-objective optimization framework [39] and the PyTorch framework [40]. For the evaluation of the neural network architectures, an NVIDIA Quadro RTX 8000 with 48 GB of memory was used. We consider both the sum and maximum latency approaches, but just the soft voting scheme to decide the output of the ensemble.

5 Results

5.1 Sum latency

For the sum of latency scenario, the only restriction used during the optimization is that an individual must have at least two genes with value one, meaning that no single architectures are allowed. The single black point on the extreme right of Figure 9 (a) illustrates the initial population. We have just a single point because all ensembles represented by this first population are equal, having therefore the same accuracy and latency. In the same figure, we can see the populations for generations 14, 32 and 60, represented by the different cloud of points, as indicated. We can see that at the generation 14, only ensembles with 50 or more components are present. Then at generation 32 almost all ensembles have between 10 and 49 components, and at generation 60, roughly half of the population have from 5 to 9 neural networks. We then plot the individuals at generation 66 in Figures 9 (b), in which we start seeing different ensemble compositions, with 2, 3 and 4 components. Finally, Figure 9 (c) illustrates the final population of ensembles after 1,000 generations.

Figure 9 (c) also compares the final population of the evolutionary process against the single architectures obtained by the OFA^2 search, used as the foundation of the ensembles. We can see that the ensembles found form a typical Pareto-front for multi-objective optimization problems with two conflicting objectives. Most of these ensembles are, however, dominated by the single architectures, with the exception of ensembles with more than 80 ms of total latency, where the accuracy of the ensembles surpass the accuracy of the single architectures. This dominance of



(a) Progression of the initial ensemble populations.

(b) Population at generation 66.

(c) Population at generation 1,000.

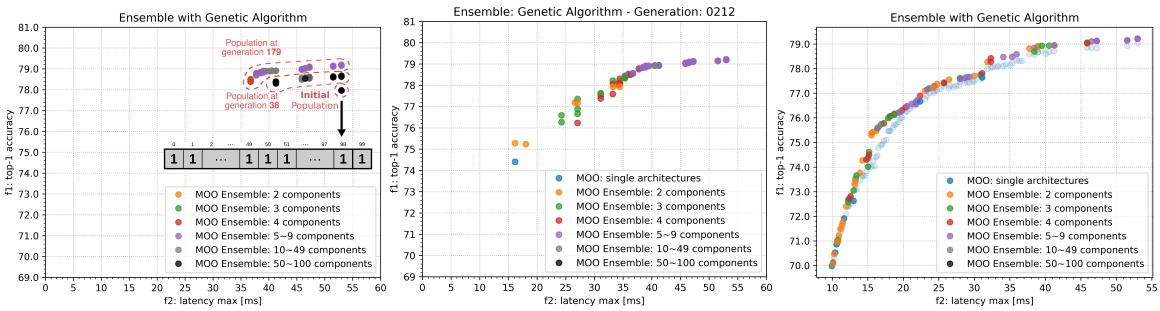
Figure 9: Progression of the populations of individuals reprensenting ensembles for the sum of latencies approach. (a): Population of ensembles at generations 0, 14, 32 and 60. (b): Population of ensembles after 66 iterations. (c): Last population of ensembles, after 1,000 generations.

single architectures can be explained due to the difference of scale between the objective functions. Take for example the two smallest neural networks candidates on participating of the ensembles. The first one presents a latency of 9.9 ms and accuracy of 69.98 %, while the second one presents a latency of 10.1 ms and accuracy of 70.12 %. When summing the latencies of these two architectures, we have a total latency of 20.0 ms. If we take the architecture with the highest latency under 20 ms, we have a single neural network with 76.01 % of accuracy and 19.82 ms of latency. It is unlikely that and ensemble of two architectures with around 70 % of accuracy surpass the 76 % of the single architecture with equivalent latency, even more considering that these two neural networks of around 70 % of accuracy are probably similar to each other on the decision space. We argue that the sum of latencies penalize too much the ensembles, and that in this scenario, it may be better to use single architectures instead.

5.2 Maximum latency

To alleviate the problem of difference in scale between latency and accuracy presented during the sum latency approach, we propose the same experiment but taking the maximum latency of the neural networks participating on the ensemble to be the latency of the ensemble itself. The lowest black point in Figure 10 (a) illustrates the initial population, again with all genes equal to 1, meaning that all neural networks candidates are participating of the ensembles. Then, in the same figure we see two clouds of points illustrating the populations at generations 38 and 179, as indicated. At generation 38 we start seeing ensembles with less than 50 components, and at generation 179 a few ensembles with 4, 3 and even 2 neural networks started to appear. Figure 10 (b) shows the progression of the populations at generations 212. Figure 10 (c) shows the last population of ensembles, after 1,000 generations. On the contrary to what was done with the sum latency approach, here we do not restrict the optimization to ensembles with 2 components or more. In fact, we can see that in the final population there are some individuals that are single architectures, meaning that other ensembles with more neural networks with lower latency perform actually worse than that specific single network.

Figure 10 (c) also compares the ensembles found by the evolutionary algorithm against the OFA^2 single architectures used as the foundation to form the ensembles. Here we can clearly see advantages of the ensembles over the single architectures, with the former dominating the latter for almost all of the latencies. The only exception happens at the beginning of the curve, where the accuracies and latencies of ensembles and single architectures are similar. In fact, some of the ensembles found by the evolutionary algorithm in this region are actually single architectures. This can be explained since the pool of neural networks to form the ensembles increases proportionally



(a) Progression of the initial ensemble populations.

(b) Population at generation 212.

(c) Population at generation 1,000.

Figure 10: Progression of the populations of individuals reprensenting ensembles for the maximum of latencies approach. (a): Population of ensembles at generations 0, 38 and 179. (b): Population of ensembles after 212 iterations. (c): Last population of ensembles, after 1,000 generations.

with the latency of the ensembles. For example, for the ensemble with the highest latency, all neural networks are available to join the ensemble, while at the ensemble with the lowest latency, there is no ensemble at all, with only one neural network being available to form the "ensemble". It is interesting to note that the evolutionary algorithm tends to reduce the number of neural networks of the ensembles with the generations, even though the first generation started with all neural networks being part of the ensembles. This indicates that ensembles with fewer components may perform better than ensembles with all neural networks [41].

6 Conclusion

In this work, we presented OFA^3, an extension for the OFA and OFA^2 NAS framework that formulates selection of neural networks to form ensembles as a multi-objective optimization problem, automatically finding efficient ensembles from OFA^2. This framework discovers architectures with improved top-1 accuracy and latency. Additionally, experiments related to ensembles such as tests with encodings and operators in the evolutionary algorithm were provided. We show that OFA^3 searched architecture consistently outperforms the architectures found by the original OFA and the OFA^2 search for most cases while being computationally much cheaper. This is because all networks from OFA^2 are searched at once in a single run and then automatically combined in one framework as ensembles with OFA^3. Additionally, we demonstrate an encoding for evolutionary algorithms that address the issue of choosing the appropriate models to assemble a compelling ensemble and the number of neural networks to include while keeping limitations (such as latency) within specific bounds. Finally, we are making all source codes publicly available and providing all the documentation for the community to use these techniques.

References

[1] T. Elsken, J. H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," *Journal of machine learning research*, vol. 20, no. 55, pp. 1–21, 2019, Accessed: Mar. 23, 2023. [Online]. Available: `http://jmlr.org/papers/v20/18-598.html`

[2] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," Feb. 15, 2017. `http://arxiv.org/abs/1611.01578` (accessed Mar. 24, 2023).

[3] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A Survey on Evolutionary Neural Architecture Search," *Ieee transactions on neural networks and learning systems*, vol. 34, no. 2, pp. 550–570, Feb. 2023, doi: 10.1109/TNNLS.2021.3100554.

[4] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh, "DrNAS: Dirichlet Neural Architecture Search," Mar. 15, 2021. http://arxiv.org/abs/2006.10355 (accessed Mar. 24, 2023).

[5] R. Adamski, "Multi-objective differentiable neural architecture search : Busca multiobjetivo e diferenciável de arquitetura de rede neural," [s.n.], 2022. Accessed: Mar. 24, 2023. [Online]. Available: https://repositorio.unicamp.br/acervo/detalhe/1243400

[6] A.-C. Cheng, C. H. Lin, D.-C. Juan, W. Wei, and M. Sun, "InstaNAS: Instance-Aware Neural Architecture Search," *Proceedings of the aaai conference on artificial intelligence*, vol. 34, no. 04, 04, pp. 3577–3584, Apr. 2020, doi: 10.1609/aaai.v34i04.5764.

[7] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "DPP-Net: Device-aware Progressive Search for Pareto-optimal Neural Architectures," 2018, pp. 517–531. Accessed: Mar. 23, 2023. [Online]. Available: https://openaccess.thecvf.com/content_ECCV_2018/html/Jin-Dong_Dong_DPP-Net_Device-aware_Progressive_ECCV_2018_paper.html

[8] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient Neural Architecture Search via Parameters Sharing," in *Proceedings of the 35th International Conference on Machine Learning*, Jul. 2018, pp. 4095–4104. Accessed: Mar. 24, 2023. [Online]. Available: https://proceedings.mlr.press/v80/pham18a.html

[9] Y. Tian *et al.*, "Evolutionary Large-Scale Multi-Objective Optimization: A Survey," *Acm comput. surv.*, vol. 54, no. 8, pp. 174:1–174:34, Oct. 2021, doi: 10.1145/3470971.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *Ieee transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.

[11] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, Aug. 2013, pp. 847–855. doi: 10.1145/2487575.2487629.

[12] W. H. Hsu, "Control of Inductive Bias in Supervised Learning Using Evolutionary Computation: A Wrapper-Based Approach," 2003. https://www.igi-global.com/chapter/control-inductive-bias-supervised-learning/www.igi-global.com/chapter/control-inductive-bias-supervised-learning/7595 (accessed Mar. 24, 2023).

[13] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for All: Train One Network and Specialize it for Efficient Deployment," Apr. 2020. Accessed: Nov. 08, 2022. [Online]. Available: https://iclr.cc/virtual_2020/poster_HylxE1HKwS.html

[14] D. Baymurzina, E. Golikov, and M. Burtsev, "A review of neural architecture search," *Neurocomputing*, vol. 474, pp. 82–93, Feb. 2022, doi: 10.1016/j.neucom.2021.12.014.

[15] D. Liu and Y. Cao, "Federated Neural Architecture Search Evolution and Open Problems: An Overview," in *Bio-Inspired Computing: Theories and Applications*, 2022, pp. 330–345. doi: 10.1007/978-981-19-1253-5_25.

[16] E. Zitzler, M. Laumanns, and S. Bleuler, "A Tutorial on Evolutionary Multiobjective Optimization," in *Metaheuristics for Multiobjective Optimisation*, 2004, pp. 3–37. doi: 10.1007/978-3-642-17144-4_1.

[17] Z. Lu, R. Cheng, Y. Jin, K. C. Tan, and K. Deb, "Neural Architecture Search as Multiobjective Optimization Benchmarks: Problem Formulation and Performance Assessment," *Ieee transactions on evolutionary computation*, p. 1, 2022, doi: 10.1109/TEVC.2022.3233364.

[18] Z. Lu *et al.*, "NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm (Extended Abstract)," Jul. 2020, vol. 5, pp. 4750–4754. doi: 10.24963/ijcai.2020/659.

[19] H. Liang *et al.*, "DARTS+: Improved Differentiable Architecture Search with Early Stopping," Oct. 20, 2020. http://arxiv.org/abs/1909.06035 (accessed Mar. 24, 2023).

[20] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical Representations for Efficient Architecture Search," Feb. 2018. Accessed: Mar. 24, 2023. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA-

[21] R. Miikkulainen *et al.*, "Chapter 15 - Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, R. Kozma, C. Alippi, Y. Choe, and F. C. Morabito, Eds. Academic Press, 2019, pp. 293–312. doi: 10.1016/B978-0-12-815480-9.00015-3.

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized Evolution for Image Classifier Architecture Search," *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 01, pp. 4780–4789, Jul. 2019, doi: 10.1609/aaai.v33i01.33014780.

[23] F. Huang, J. Ash, J. Langford, and R. Schapire, "Learning Deep ResNet Blocks Sequentially using Boosting Theory," in *Proceedings of the 35th International Conference on Machine Learning*, Jul. 2018, pp. 2058–2067. Accessed: Mar. 24, 2023. [Online]. Available: `https://proceedings.mlr.press/v80/huang18b.html`

[24] C. Liu *et al.*, "Progressive Neural Architecture Search," 2018, pp. 19–34. Accessed: Mar. 24, 2023. [Online]. Available: `https://openaccess.thecvf.com/content_ECCV_2018/html/Chenxi_Liu_Progressive_Neural_Architecture_ECCV_2018_paper.html`

[25] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS," Dec. 12, 2019. `http://arxiv.org/abs/1912.06059` (accessed Mar. 24, 2023).

[26] C. Wei, C. Niu, Y. Tang, Y. Wang, H. Hu, and J. Liang, "NPENAS: Neural Predictor Guided Evolution for Neural Architecture Search," *Ieee transactions on neural networks and learning systems*, pp. 1–15, 2022, doi: 10.1109/TNNLS.2022.3151160.

[27] Y. Peng, A. Song, V. Ciesielski, H. M. Fayek, and X. Chang, "PRE-NAS: Evolutionary Neural Architecture Search With Predictor," *Ieee transactions on evolutionary computation*, vol. 27, no. 1, pp. 26–36, Feb. 2023, doi: 10.1109/TEVC.2022.3227562.

[28] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural Architecture Optimization," in *Advances in Neural Information Processing Systems*, 2018, vol. 31. Accessed: Mar. 24, 2023. [Online]. Available: `https://papers.nips.cc/paper_files/paper/2018/hash/933670f1ac8ba969f32989c312faba75-Abstract.html`

[29] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-030-05318-5.

[30] R. C. Ito and F. J. Von Zuben, "OFA^2: A Multi-Objective Perspective for the Once-for-All Neural Architecture Search," Mar. 23, 2023.

[31] L. Hansen and P. Salamon, "Neural network ensembles," *Ieee transactions on pattern analysis and machine intelligence*, vol. 12, no. 10, pp. 993–1001, Oct. 1990, doi: 10.1109/34.58871.

[32] M. P. Perrone and L. N. Cooper, "When networks disagree: Ensemble methods for hybrid neural networks," in *How We Learn; How We Remember: Toward an Understanding of Brain and Neural Systems*, vol. Volume 10, WORLD SCIENTIFIC, 1995, pp. 342–358. doi: 10.1142/9789812795885_0025.

[33] S. Geman, E. Bienenstock, and R. Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, Jan. 1992, doi: 10.1162/neco.1992.4.1.1.

[34] Y. Liu, X. Yao, and T. Higuchi, "Evolutionary ensembles with negative correlation learning," *Ieee transactions on evolutionary computation*, vol. 4, no. 4, pp. 380–387, Nov. 2000, doi: 10.1109/4235.887237.

[35] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: A survey and categorisation," *Information fusion*, vol. 6, no. 1, pp. 5–20, Mar. 2005, doi: 10.1016/j.inffus.2004.04.004.

[36] G. Brown, "Diversity in neural network ensembles," 2004. Accessed: Feb. 01, 2023. [Online]. Available: `https://www.semanticscholar.org/paper/Diversity-in-neural-network-ensembles-Brown/b2329bfeaff2c9edbe4891ad56e4a4e03ad4fa59`

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255. doi: 10.1109/CVPR.2009.5206848.

[38] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware," Feb. 22, 2019. http://arxiv.org/abs/1812.00332 (accessed Mar. 23, 2023).

[39] J. Blank and K. Deb, "Pymoo: Multi-Objective Optimization in Python," *Ieee access*, vol. 8, pp. 89497–89509, 2020, doi: 10.1109/ACCESS.2020.2990567.

[40] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019, vol. 32. Accessed: Jan. 19, 2023. [Online]. Available: https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[41] H. Bonab and F. Can, "Less Is More: A Comprehensive Framework for the Number of Components of Ensemble Classifiers," *Ieee transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2735–2745, Sep. 2019, doi: 10.1109/TNNLS.2018.2886341.

7 Submission Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   (b) Did you describe the limitations of your work?

   (c) Did you discuss any potential negative societal impacts of your work?

   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/`

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results?

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? We still need to organize the GitHub repository. Straight after that, we will provide all source code, seeds to run experiments, Jupyter Notebooks, etc.

   (b) Did you include the raw results of running the given instructions on the given code and data? In a form of a Jupyter Notebook

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? In a form of a Jupyter Notebook

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented?

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)?

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code?

   (g) Did you run ablation studies to assess the impact of different components of your approach? [N/A]

   (h) Did you use the same evaluation protocol for the methods being compared?

   (i) Did you compare performance over time? In a form of a Jupyter Notebook

   (j) Did you perform multiple runs of your experiments and report random seeds? In a form of a Jupyter Notebook

   (k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?

   (l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [N/A]

   (m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? In a form of a Jupyter Notebook

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

   (a) If your work uses existing assets, did you cite the creators?

   (b) Did you mention the license of the assets?

   (c) Did you include any new assets either in the supplemental material or as a URL? We still need to organize the GitHub repository. Straight after that, we will provide all source code, seeds to run experiments, Jupyter Notebooks, etc.

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects…

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

After careful reflection, the authors have determined that this work presents no notable negative impacts to society or the environment.