Hot-started NAS for Task-specific Embedded Applications

Lotte Hendrickx, Wiebe Van Ranst, Toon Goedemé
EAVISE-PSI-ESAT, KU Leuven
Jan Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium
lotte.hendrickx@kuleuven.be

Abstract

Neural architecture search (NAS) has proven its worth in discovering new neural networks. Combining the possibility to satisfy multiple objectives in one search, it is especially useful for getting the most out of embedded devices with limited resources. However, research into small and efficient neural networks precedes NAS. We investigate the influence of combining this pre-existing knowledge with NAS techniques, for which we propose to hot-start the NAS search with a human-designed optimal network. Our experiments show that doing so speeds up the NAS process significantly, but the resulting optimal model at the end is only marginally better. Since embedded devices are often used for a specific task, we also explore the impact of using a task-specific dataset in the NAS process. Our experiments demonstrate that for a constrained problem, a smaller network can be found as compared to a general problem.

1. Introduction

The past few years, Neural Architecture Search (NAS) has created new opportunities in deep learning. NAS provides the possibility to discover the best neural network for various objectives automatically. In the past, experts handcrafted neural networks to perform well under various conditions, making them more general purpose. This idea of designing more general purpose networks greatly shows off their potential and in turn many are interested in using them for their specific use case. However, these existing neural networks may not be the best fit for the very specific demands of a real-life, industrial application. Indeed, many AI-powered consumer devices have a fixed task, which never changes. An example is a surveillance camera with built-in person detector, or a LiDAR-navigating vacuum cleaning robot. For these kind of applications, a general purpose neural network architecture is a non-optimal choice in terms of compute vs accuracy trade-off. We believe that for a fixed, predefined task, both the neural network architecture as well as the model parameters can both be optimised. With the help of NAS, we can search for a bespoke solution to tackle these application-specific challenges.

The previously mentioned examples are all cases of embedded applications. The resources on embedded devices are scarce, and memory often is one of the biggest bottlenecks. We are using NAS to find lightweight models to suit these limited conditions.

Many strategies have already been proposed and implemented to perform the task of neural architecture search, delivering satisfying results in a great deal of cases. In this work, we do not focus on creating a completely new algorithm from scratch, but rather we are more interested in discovering how useful NAS can really be for real world applications. Following this reasoning, we propose to use an existing NAS method, extend it and investigate its potential in our experiments.

Most of these search strategies, much like neural networks, have been developed using academic datasets such as CIFAR10. These academic datasets contain a multitude of different categories that are often not related at all. While this offers a good baseline when comparing the performance of different strategies, they don't reveal much when talking about a more application specific use case. In many real life, task-specific, scenarios this variation in data may simply not be present (and therefore needed) in the final application. For instance, a traffic monitoring camera is always mounted with a similar point-of-view on the road, rendering the amount of viewpoint variance much lower compared to the academic datasets. In this paper, we examine how this shift towards such a constrained target dataset influences the final outcome of a NAS search. Previous work has proven that constrained datasets can lead to smaller neural networks [24].

Moreover, human experts have spent tremendous efforts on designing well performing and at the same time efficient neural networks and gained insights about this along the way. In contrast, NAS often starts blind, without this expert knowledge. It is true that this enables the search strategy to explore all possibilities, but on the other hand, the architectures and building blocks designed by experts have proven their performance in the past. Given that NAS can be very time consuming due to this intensive exploration of the search space, we question if it would be useful to rely on this expert knowledge to get to a solution faster, by hot-starting a NAS with human designed architectures.

2. Related Work

The field of NAS has been studied extensively for years. In this chapter, we give a brief summary of relevant and related work.

Many algorithms have been implemented over the years to serve the purpose of NAS. Some of the biggest categories include reinforcement learning (RL) [1, 30, 37], genetic algorithms (GA) [21,26,35] and other techniques such as one-shot learning [20, 25, 34]. In recent years much effort has gone into making all of these algorithms faster, leading to results in just a few hours to days. Papers presenting an introduction and comprehensive comparison are available to provide a more in depth overview of the current state-of-the-art [2, 10, 12].

NSGA-Net [22] is such a NAS strategy using a genetic algorithm. This search strategy is based on NSGA-II, a multi-objective genetic algorithm. A genetic algorithm [21], a subclass of evolutionary algorithms, is a heuristic for optimisation inspired by nature. The algorithm uses nature-inspired operators such as mutation and crossover to reach an optimal solution based on natural selection.

NSGA-Net [22] is one of the NAS algorithms that uses a genetic algorithm. This search strategy is based on NSGA-II, a multi-objective genetic algorithm. A genetic algorithm, a subclass of evolutionary algorithms, is a heuristic for optimisation inspired by nature. The algorithm uses nature-inspired operators such as mutation and crossover to reach an optimal solution based on natural selection. We will use this technique as the starting point for our NAS experiments.

In section 1 we have referred to finding the ’best’ neural network possible. Originally, this meant achieving the lowest error possible. In many cases we can broaden this scope to fulfill multiple requirements, leading to multi-objective NAS . Typically this means focusing on other aspects of the performance as well, such as the amount of operations needed (FLOPs or MACs [6, 19, 23, 31], model size [5] or latency [30, 33]. This way we can utilise NAS to optimally use the resources of devices that only have a limited amount of them available.

Much research has been done in the field of deep learning in general the last few years. NAS often discards this, in order not to bias the algorithm used in the search. The idea has risen that it might not always be necessary to completely discard this information but instead we might leverage it to our advantage. This can be used in the form of hot-starting the algorithm instead of random initialisation. Among the existing networks, some have novel ideas that make them perform better on limited hardware. One of the most used examples in this case is the MobileNet-family [13, 14, 27]. MobileNetV2 introduces the efficient depth-wise separable convultions, which we will add to our NAS approach. It is used so often that it can be found in popular deep learning frameworks like TensorFlow [32]. Hot-starting the NAS algorithm isn’t common, but one such approach is [16]. They use pre-trained PyTorch models for their RL approach.

As previously mentioned, multi-objective search has mainly become the norm in NAS. This has more recently led to an interest in NAS for smaller, embedded devices that can for example be found in IoT and TinyML applications. MCUNet [19] is a search strategy for off-the-shelf small devices. It has two components: a one-shot based NAS strategy on a pre-optimised search space and an inference optimisation engine instead of relying on out of the box deep learning libraries. Whilst yielding satisfying results, the search strategy is rather complex and involves numerous optimisation steps outside of the actual search strategy. MCUNet for computer vision focuses on large-scale image recognition in the form of the ImageNet dataset. Other approach such as MnasNet [30] and [4] employ a more vanilla search strategy and optimise for latency on the target device on the classic academic datasets.

Another solution involves the co-design of both hardware and software [3, 11, 17, 18] to reach an optimal solution.

To reduce the search time, some NAS strategies rely on accuracy estimation. When using academic datasets such as CIFAR10, benchmarks can be used for his purpose [9, 28, 36]. For our application-specific case, we need to obtain real-time accuracy on our specific dataset.

Most of the previously mentioned work has been performed in academic settings using academic computer vision datasets. Apart from that, medical images and applications seem to be the most popular subject for NAS. Research on task-specific datasets, especially on embedded devices, is limited. One such recent example involves forest animals [15].

In this paper we leverage an existing GA NAS strategy (NSGA-Net) to search for a task-specific and lightweight embedded neural network.

The contributions of this paper are the following:

- We adapt NSGA-Net for hot-starting from a known architecture and investigate its impact on the final result.
- We add depth-wise separable convolutions to NSGA-Net.
- We investigate the influence of using NAS for a constrained problem.

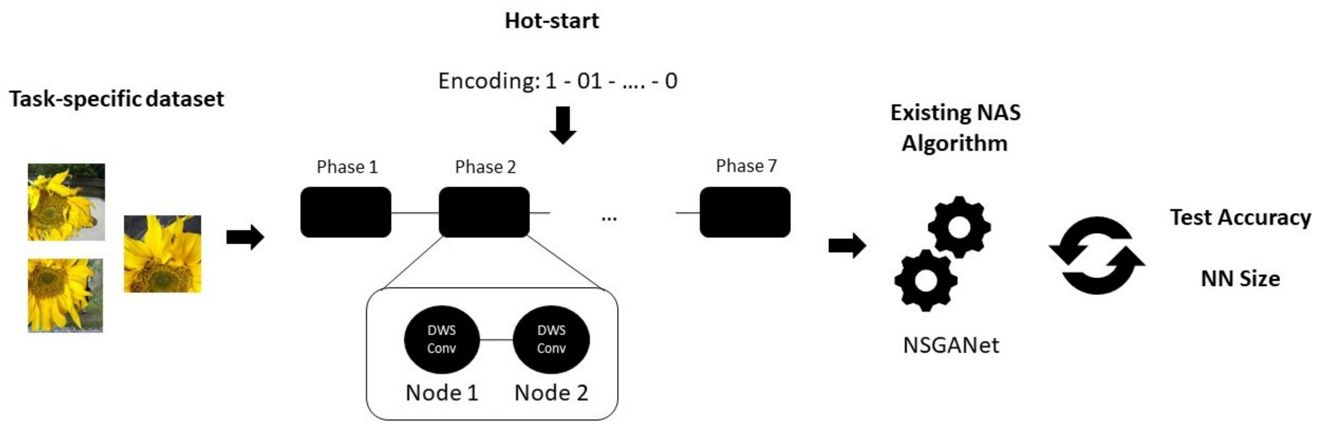


Figure 1. Hot-started NAS approach for task-specific applications

- We propose to optimise the neural network for accuracy and size, as memory is often the bottleneck in embedded applications.
- We investigate the sense and nonsense of hot-starting NAS with a human-designed architecture.

In section 3 we will explain our own search strategy and in section 4 the dataset and its use case is explained. Our implementation is described in section 5 and the results are discusses in section 6. Lastly we will look at further challenges in section 7.

3. Strategy

In this chapter we will lay out our approach for our NAS experiments. Figure 1 shows a schematic summary.

3.1. NAS Approach

We have already mentioned our interest in using existing NAS approaches in section 1. This immediately introduces a few challenges. As mentioned in section 2, NAS can be performed using various search strategies. Furthermore, to ensure a correct implementation, we are only interested in search strategies with code available online. Other requirements we have are multi-objective search and a relatively fast search time.

NSGA-Net satisfies all our demands and uses a relatively straightforward approach. We will expand on the original code provided by [22].

NSGA-Net uses the NSGA-II algorithm. This stands for Non-Dominated Sorting Genetic Algorithm II [8], an algorithm that has been widely used in all sorts of optimisation problems. Non-dominated sorting sorts the population into Pareto-fronts for the objectives. In a second step, crowd distance sorting is used to select individuals with respect to fitness and diversity. We choose the NSGA-Net approach because it uses a genetic algorithm that has proven its worth in various domains before, as well as delivering good results for the task of NAS. We are interested in genetic algorithms because of their nature-inspired ability to inherit traits and evolve from there. This is in line with our idea of hot-starting to benefit from prior knowledge.

The original implementation of NSGA-Net randomly initialises the algorithm. For our experiments, we adapted this algorithm such that it can be hot-started from an architecture similar to MobileNetV2. This choice is motivated by the fact that MobileNetV2 is an an architecture that has been targeted for implementation on edge devices, such as the Kendryte K210, before as well as its widespread use in lightweight frameworks like TensorFlow Lite. MobileNetV2 introduced the idea of the depth-wise separable convolution, which led to a reduction in parameters in the network. This idea of the depth-wise separable convolution is useful for devices that have limited compute resources available compared to the GPUs that are used when training a neural network. We added these depth-wise separable convolutions to the original NSGA-Net implementation. Because of the limitations imposed by the structure of NSGA-Net, explained in subsection 5.2, we can't implement all details without changing major parts of NSGA-Net. For this reason, we can't implement the skip connections when these are present.

3.2. Multi-objective Search

NSGA-Net is a multi-objective search strategy. This means that we can satisfy more than one objective using this search strategy. The first objective we want to satisfy is a neural network that can correctly classify the images it processes. To do this, we will try to minimise the classification error on the test dataset.

The second one relates to the application of our neural networks, on which we will elaborate in section 4. Em-

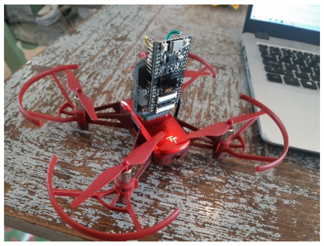


Figure 2. DJI Tello drone used in [7] for flower pollination

bedded devices generally have a rather limited amount of resources available. These resources include compute resources such as the CPU, but also the amount of onboard memory can be a limiting factor. To make the most of these, FLOPs and MACs or size are obvious choices like explained in section 2. For our implementation described in section 4 the bottleneck was the available onboard memory. So, we adapted our NAS approach to measure the size of the neural network as the second objective in our search. Both need to be minimised in our experiments to obtain the best architecture for the application.

4. Dataset

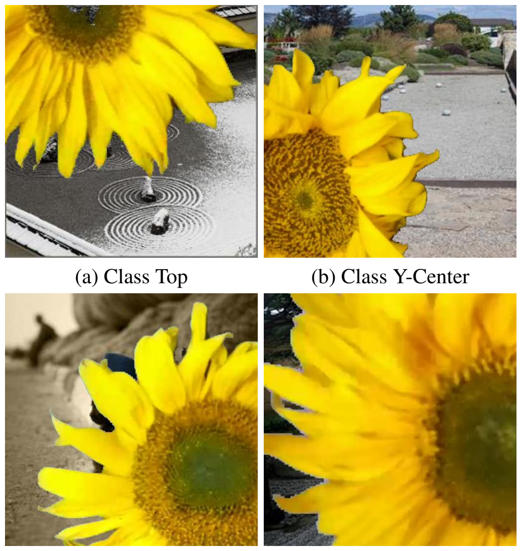
Our application is a nanodrone (see Figure 2) that can automatically navigate to flowers and pollinate them [7]. During the second phase of the navigation procedure, a direct visual servoing approach is used to exactly positioned the drone w.r.t. the flower. We do this positioning using a neural network classifier that directly predicts control commands for the drone with the objective of keeping the flower centered, while slowly approaching the flower.

We conduct our experiments using a dataset created by [7]. This dataset only contains images of sunflowers and is therefore very limited in its diversity. The differences between the images are the position of the flower and the background of the image. Using this dataset, we want to enable drones to automatically detect sunflowers and navigate to the optimal position to pollinate them.

The classes of the dataset represent the moves the drone should make in order to reach this optimal position. These classes are: Top, Bottom, Left, Right, X-Center and Y-Center. The original image size is 416 by 416 pixels and the images are provided in RGB format.

The dataset consists of 4800 labeled images, partitioned using a 10-90 divide for test and training data. Figure 3 shows a few examples of images present in this dataset.

A common approach to pre-training is using a general purpose dataset like ImageNet when little data or only a dataset with limited variation is available. This a step that we will ignore on purpose. We employ this technique because we don't need the features learned by the pre-training for our limited task, and the pre-training is a very time con-



(a) Class Top (b) Class Y-Center (c) Class Bottom (d) Class Right

Figure 3. Examples of images present in the sunflower dataset.

suming step because of the size of these general purpose datasets. Other than that we want to rely purely on the performance of the architecture found and not introduce a bias that could be related to using pre-trained weights or pre-training with a dataset that is different from our target dataset.

5. Set-up and Implementation

In this chapter we will discuss how we implemented and conducted our experiments.

5.1. Training Parameters and Set-Up

Our experiments are conducted using a NVIDIA Tesla V100 GPU with 32 GB VRAM. We use the dataset described in section 4 and resize the images to 32 by 32 in order to have the same size images as the CIFAR10 dataset. We don't follow CIFAR10 in its black and white input images. Our dataset is in RGB format and we want to keep the three input channels.

We use parameters of NSGA-Net that the original authors found to yield the best results. We do change the amount of epochs to 20 and batch size to 250 to fit our own training set-up.

The parameters for the genetic algorithm are as follows: we let the algorithm run for 40 generations with a population size of 30 and produce an offspring of 40 candidates each generation.

5.2. Hot-Start

An approach in NAS is only looking for an optimal block and repeating that same block multiple times to make up a neural network. The other approach is to search for a whole neural network at once. NSGA-Net combines both approaches. NSGA-Net searches for the whole architecture, but the architecture is made up of blocks, called phases. MobileNetV2, apart from the depth-wise separable convolutions, also features repeats that are characterised by the channels of the convolutions. We can leverage this feature to define our blocks and emulate a MobileNetV2-like architecture in the NSGA-Net imposed structures. Each repeat sequence forms its own block, that in turn is made up of nodes. The amount of nodes in each block is determined by the amount of repeats in MobileNetV2. In the nodes we introduce the depth-wise separable convolutions instead of the regular convolutions used by NSGA-Net.

The actual hot-start of the algorithm happens by initialising the first generation of the algorithm with the MobileNetV2-like structure. This has to be done by providing the correct genome representation of our hot-start architecture. We derive this representation ourselves, based on the seven blocks defined by the repeats and the corresponding channels of MobileNetV2. Figure 1 shows a visual representation of this hot-start implementation.

In order to execute the multi-objective search we have described in section 3, we need the size of the neural network. We use a Python package called torchsummary [29] to provide this information, as it is not included with PyTorch.

6. Results

To study the impact of all our proposed changes, we have conducted a few different experiments. In this chapter we will analyse and compare these results.

6.1. Vanilla NSGA-Net

In order to investigate the full impact of the hot-start only, we run both a hot-started (Figure 4) and randomly initialised (Figure 5) version of our NAS search. We observe that the hot-started version starts with a smaller model as well as a higher test accuracy and reaches better results more quickly. However, as time continues, the gap becomes smaller. After 40 generations a similar optimal point is reached. We see that hot-starting the NAS steers the search quickly in a good direction: the spread is smaller for the hot-started scenario.

6.2. Depth-Wise Separable

MobileNetV2 introduces the depth-wise separable convolutions. In the original implementation this comes with a channel expansion between the input and output channels. For a first comparison, we keep the expansion factor at 1. Again, we perform the experiments hot-started (Figure 6) and randomly initialised (Figure 7).

Figure 4. Results of hot-started vanilla NSGA-Net on flower dataset.

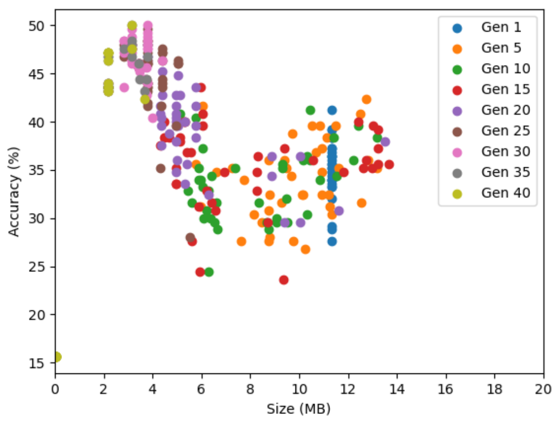
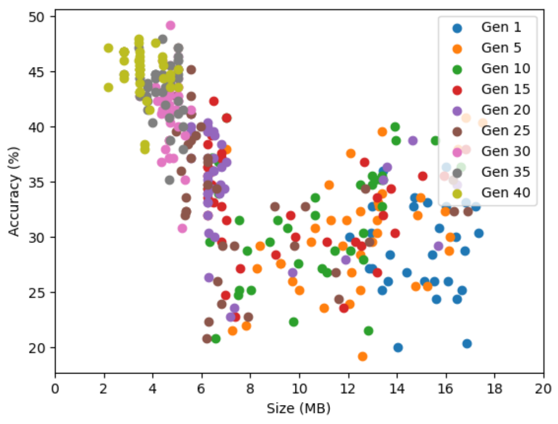


Figure 5. Results of randomly initialised vanilla NSGA-Net on flower dataset.



The original starting size of the neural network is smaller at the start for the hot-start than with random initialisation. Initially these hot-started models have a higher test accuracy. But after 40 generations both approaches move again to a similar optimal point. We observe that hot-starting this search has a positive influence. Hot-starting causes a faster convergence to the optimal, e.g. for generation 20 in the experimental results, we observe a accuracy spread between 25% and 50% accuracy when randomly initialized, while it is between 37% and 50% in the hot-started scenario. In the later generations the hot-started neural networks converge more to the same point.

Figure 6. Results of hot-started NSGA-Net with depth-wise separable convolutions on flower dataset.

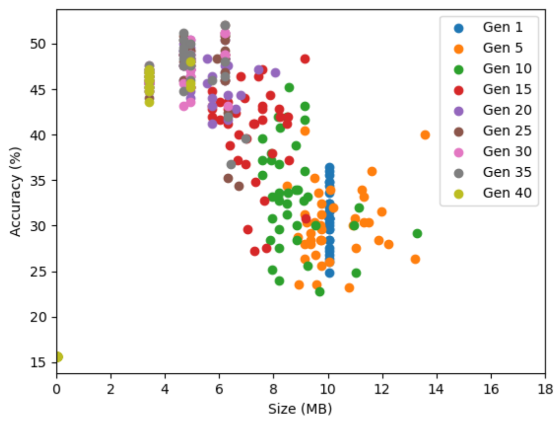
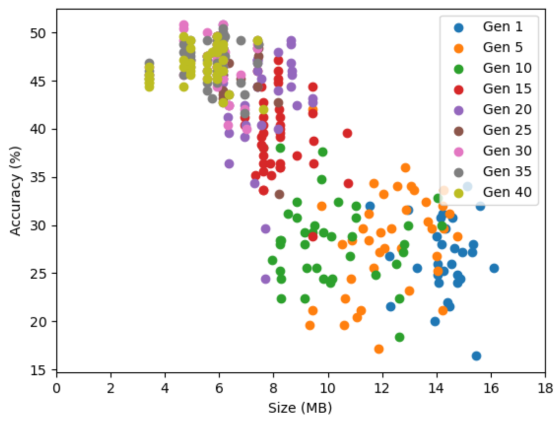


Figure 7. Results of randomly initialised NSGA-Net with depth-wise separable convolutions on flower dataset.



6.3. Depth-Wise Separable Expanded

The next step we implemented is the addition of the expansion factor in MobileNetV2. Results for the hot-started experiments are shown in Figure 8 and Figure 9 shows the results for random initialisation. Again the starting size is smaller for the hot-started neural networks, but bigger than without the expansion factor added. The hot-started models move faster to the optimum, though they both reach the optimum after 40 generations.

6.4. Comparison

Table 1 presents a comparison of the results of all the previously mentioned experiments. The comparison shows the

Figure 8. Results of hot-started NSGA-Net with expanded depth-wise separable convolutions on flower dataset.

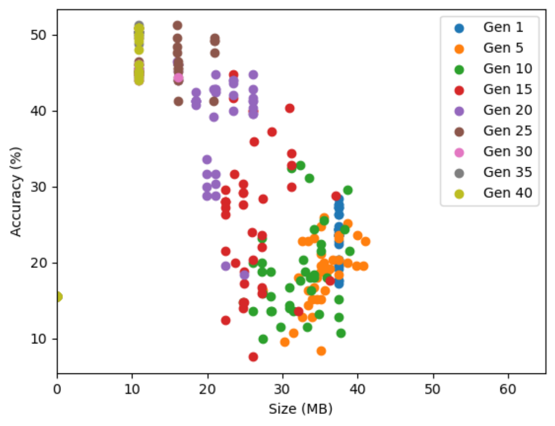
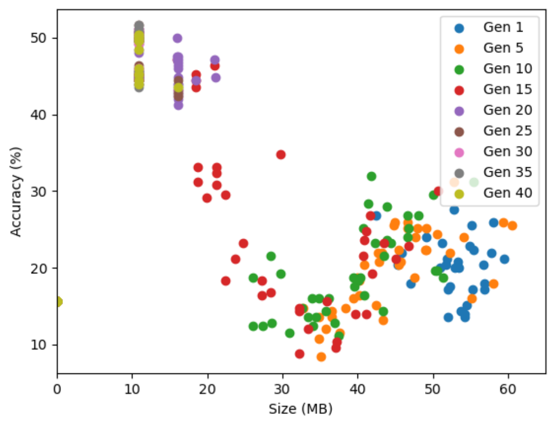


Figure 9. Results of randomly initialised NSGA-Net with expanded depth-wise separable convolutions on flower dataset.



optima when focusing on a accuracy vs size trade-off. We observe that, especially when reaching the optimal point, the model size boils down to discontinuous values, showing as vertical clusters in the plots. These models are different optimized models, but contain exactly the same number of parameters. For each of these vertical clusters, the optimal point is easily to be found as the one with the top accuracy.

For comparison, we also counted the FLOPs of each model, as computed in [22].

We indeed see that hot-starting or not yields a comparable optimal model, but with hot-starting it is slightly better. Previous experiments showed that in the latter case, this optimum is more quickly reached.

		Size (MB)	Accuracy (%)	FLOPs (M)
Vanilla	Initialisation	11.3	39.2	43.8
	Random initialisation optimum	**2.2**	**47.2**	**10.3**
	Hot-started optimum	**2.2**	**47.2**	**10.3**
Depth-wise separable	Initialisation	10.5	36.4	15.6
	Random initialisation optimum	3.4	46.4	7.1
	Hot-started optimum	**3.4**	**47.2**	**7.1**
Depth-wise separable expanded	Initialisation	37.5	27.6	64.3
	Random initialisation optimum	10.9	50.4	14.8
	Hot-started optimum	**10.9**	**50.8**	**14.8**

Table 1. Comparison of all highest results, focusing on size vs accuracy trade-off.

6.5. Limited dataset vs CIFAR10

We want to see what the impact of a limited dataset is on the final outcome of a NAS search. The original authors of NSGA-Net [22] present their findings on the CIFAR10 dataset, which we will use as a comparison for a more general purpose dataset. Their results show that they start with three phases and also have three phases in their final results. We start with seven, but end up with one in our later generations.

We conclude that for our limited dataset, the reductions are much bigger than for a general purpose dataset. For our task-specific dataset the final neural networks are between 3 to 5 times smaller when compared to the starting point.

This conclusion is in line with the findings of Ophoff *et al.* [24], who demonstrated similarly a higher optimisation factor achieved for constrained problems by pruning.

7. Conclusion

In this paper we have presented the results of a hot-started NAS approach for a task-specific embedded application. To do this we have adapted an existing NAS approach called NSGA-Net. We have examined the impact of hot-starting the algorithm compared to random initialisation. Next, we have looked at the results of adding depth-wise separable convolutions, a technique targeting devices that are limited in resources. Lastly we explore the effect of using NAS on a constrained dataset.

The benefit of hot-starting the NAS algorithm lies in its ability to reach better results more quickly as well as starting from a better starting point. But, we observe that in the end, both approaches reach a similar optimum.

Adding depth-wise separable convolutions leads to a slightly larger final result, but with similar accuracy. The depth-wise separable convolutions do indeed lead to a lower amount of FLOPs compared to regular convolutions. When the expansion factor is added, size increases but accuracy goes up.

Finally we conclude that using a task-specific dataset leads to significantly smaller neural networks.

Future research opportunities include using metrics measured on an embedded target device.

Acknowledgements

This work has been supported by the Flemish Government under the AI Research Program. The authors thank the company MAGICS for supplying the sunflower dataset.

References

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing Neural Network Architectures using Reinforcement Learning. *arXiv:1611.02167 [cs]*, Mar. 2017. arXiv: 1611.02167. 2

[2] Dilyara Baymurzina, Eugene Golikov, and Mikhail Burtsev. A review of neural architecture search. *Neurocomputing*, 474:82–93, Feb. 2022. 2

[3] Oliver Bringmann, Wolfgang Ecker, Ingo Feldner, Adrian Frischknecht, Christoph Gerum, Timo Hämäläinen, Muhammad Abdullah Hanif, Michael J. Klaiber, Daniel Mueller-Gritschneder, Paul Palomero Bernardo, Sebastian Prebeck, and Muhammad Shafique. Automated HW/SW co-design for edge AI: state, challenges and steps ahead. In *Proceedings of the 2021 International Conference on Hardware/Software Codesign and System Synthesis*, pages 11–20, Virtual Event, Sept. 2021. ACM. 2

[4] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. *arXiv:1812.00332 [cs, stat]*, Feb. 2019. 2

[5] Thomas Cassimon, Simon Vanneste, Stig Bosmans, Siegfried Mercelis, and Peter Hellinckx. Designing resource-constrained neural networks using neural architecture search targeting embedded devices. *Internet of Things*, page 100234, May 2020. 2

[6] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Hailong Ma. Multi-Objective Reinforced Evolution in Mobile Neural Architecture Search. *arXiv:1901.01074 [cs]*, Jan. 2019. arXiv: 1901.01074. 2

[7] Hulens D, Van Ranst W., Cao Y., and Goedemé T. The autonomous pollination drone. *Proceedings of the 2nd Winter IFSA Conference on Automation, Robotics & Communications for Industry 4*, 2:38–41, Feb. 2022. 4

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. 3

[9] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search, 2020. 2

[10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A Survey. *arXiv:1808.05377 [cs, stat]*, Apr. 2019. arXiv: 1808.05377. 2

[11] Cong Hao, Jordan Dotzel, Jinjun Xiong, Luca Benini, Zhiru Zhang, and Deming Chen. Enabling Design Methodologies and Future Trends for Edge AI: Specialization and Codesign. *IEEE Design Test*, 38(4):7–26, Aug. 2021. Conference Name: IEEE Design Test. 2

[12] Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, Jan. 2021. 2

[13] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 2

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017. 2

[15] Liang Jia, Ye Tian, and Junguo Zhang. Domain-aware neural architecture search for classifying animals in camera trap images. *Animals*, 12(4), 2022. 2

[16] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. Standing on the Shoulders of Giants: Hardware and Neural Architecture Co-Search with Hot Start. *arXiv:2007.09087 [cs, eess, stat]*, July 2020. arXiv: 2007.09087. 2

[17] Weiwen Jiang, Lei Yang, Edwin Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. Hardware/Software Co-Exploration of Neural Architectures. *arXiv:1907.04650 [cs]*, Jan. 2020. arXiv: 1907.04650. 2

[18] Siyi Li, Yanan Sun, Gary G. Yen, and Mengjie Zhang. Automatic Design of Convolutional Neural Network Architectures Under Resource Constraints. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2021. Conference Name: IEEE Transactions on Neural Networks and Learning Systems. 2

[19] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. MCUNet: Tiny Deep Learning on IoT Devices. *arXiv:2007.10319 [cs]*, July 2020. 2

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. *arXiv:1806.09055 [cs, stat]*, Apr. 2019. arXiv: 1806.09055. 2

[21] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, and Gary Yen. A Survey on Evolutionary Neural Architecture Search. *arXiv:2008.10937 [cs]*, Aug. 2020. arXiv: 2008.10937. 2

[22] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. NSGA-Net: Neural Architecture Search using Multi-Objective Genetic Algorithm. *arXiv:1810.03522 [cs]*, Apr. 2019. arXiv: 1810.03522. 2, 3, 6, 7

[23] Zhichao Lu, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. Multi-Objective Evolutionary Design of Deep Convolutional Neural Networks for Image Classification. *arXiv:1912.01369 [cs]*, Sept. 2020. arXiv: 1912.01369. 2

[24] Tanguy Ophoff, Cedric Gullentops, Kristof Van Beeck, and Toon Goedeme. Investigating the Potential of Network Optimization for a Constrained Object Detection Problem. *Journal of Imaging*, 7(4), 2021. Publisher: MDPI. 1, 7

[25] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. *arXiv:1802.03268 [cs, stat]*, Feb. 2018. arXiv: 1802.03268. 2

[26] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-Scale Evolution of Image Classifiers. *arXiv:1703.01041 [cs]*, June 2017. arXiv: 1703.01041. 2

[27] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018. 2

[28] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search, 2020. 2

[29] sksq96. pytorch-summary, 2021. 5

[30] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-Aware Neural Architecture Search for Mobile. *arXiv:1807.11626 [cs]*, May 2019. arXiv: 1807.11626. 2

[31] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, Sept. 2020. arXiv: 1905.11946. 2

[32] TensorFlow. Tensorflow 2 detection model zoo, 2021. 2

[33] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. *arXiv:1812.03443 [cs]*, May 2019. arXiv: 1812.03443. 2

[34] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Zhengsu Chen, Lanfei Wang, An Xiao, Jianlong Chang, Xiaopeng Zhang, and Qi Tian. Weight-Sharing Neural Architecture Search: A Battle to Shrink the Optimization Gap. *arXiv:2008.01475 [cs]*, Aug. 2020. arXiv: 2008.01475. 2

[35] Lingxi Xie and Alan Yuille. Genetic CNN. *arXiv:1703.01513 [cs]*, Mar. 2017. arXiv: 1703.01513. 2

[36] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search, 2019. 2

[37] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv:1611.01578 [cs]*, Feb. 2017. arXiv: 1611.01578. 2