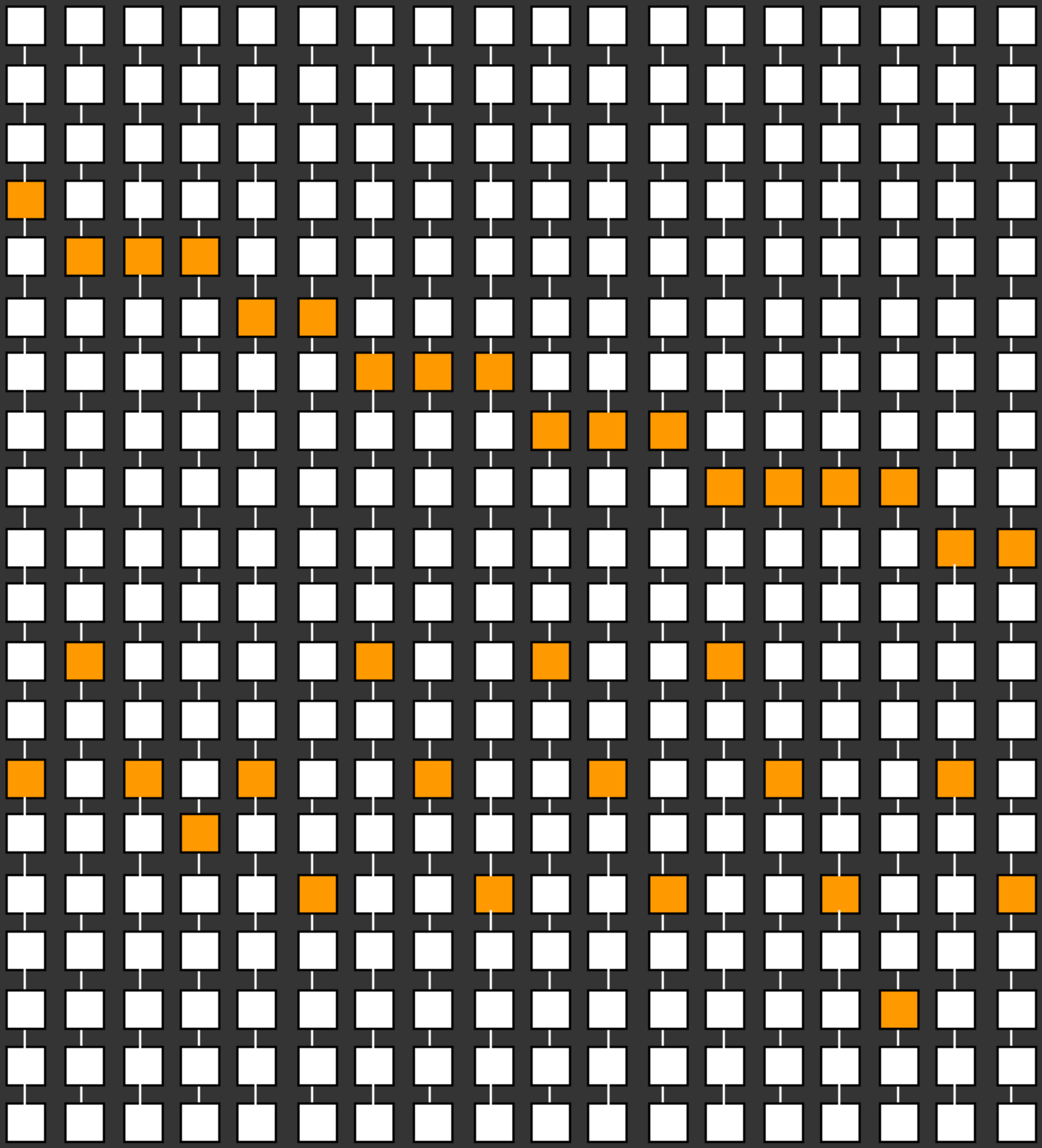
Evaluating cell-based neural architectures on embedded systems



Ilja van Ipenburg

Layout: typeset by the author using $\LaTeX$.
Cover illustration: Ilja van Ipenburg

Evaluating cell-based neural architectures on embedded systems

Ilja van Ipenburg
11862904

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Dolly Sapra MSc

Parallel Computing Systems
Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

January 29th, 2021

Abstract

Neural Architectures Search (NAS) methodologies used to discover state-of-the-art neural networks without human intervention, have seen a growing interest in recent years. One subgroup of NAS methodologies consists of *cell-based neural architectures*, which are architectures made up of convolutional cells, which are repeatedly stacked on top of each other to form a complete neural network. The way in which these cells can be stacked is defined by their meta-architecture. *Stochastic Neural Architecture Search* (SNAS) found cell architectures achieving state-of-the-art accuracy of 97.02%, while significantly reducing search time compared to other NAS methods. Cell-based neural architectures are an interesting target for usage on embedded systems, which are computer systems usually performing a single function within a larger system. These systems are often tightly constrained in resources. This research explores the effect that the architecture of the cells found by SNAS has on the accuracy, latency, and power usage on an embedded system. To answer this, architectures were found within a defined meta-architecture, and evaluated on the NVIDIA Jetson Nano. Multiple architectures were found achieving a lower latency and power usage while maintaining a comparable accuracy, one achieving 97.49% accuracy.

Contents

1 Introduction

In recent years there has been a growing interest in *Neural Architecture Search* (NAS), the automation of architecture engineering[1] for neural networks. One important subgroup of NAS consists of algorithms focused on *cell-based neural architectures*. In these architectures, cells consisting of several layers are found and repeatedly stacked on top of each other to form a complete neural network. The way these found cells are stacked and interact is called the *meta-architecture* of the neural network. This approach has certain advantages over other NAS methods, namely a smaller search space and adaptability to new datasets.

These cell-based neural architectures also seem like an interesting target for use on resource constrained embedded systems. Embedded systems can be found in almost all aspects of daily life, from "smart" home appliances to factory machines, and there is an increasing demand for deep learning capabilities on these systems. However, most neural architectures do not take resource constrained systems into account during engineering. These architectures are employed on powerful high-end GPUs, which are not viable options for many environments embedded systems are used in. Oftentimes, these systems must be cheap, able to fit in small spaces, and run on an internal battery. Consequentially, they are constrained in terms of memory, processing power, speed, and energy. Therefore, neural architectures employed on these systems are limited in size, while still having to run efficiently. Architectures like MobileNet[2] are taking these variables into account, but are still designed manually. Therefore, analyzing and evaluating NAS-found cell-based neural architectures and changes in their cell structure on embedded systems, is an interesting topic of research.

The cells chosen for this research were found by SNAS (Stochastic Neural Architecture Search)[3]. The architecture composed by these cells achieved state-of-the-art accuracy, and a search cost that is orders of magnitude lower than previous reinforcement-learning-based cell-based architectures[4]. For this, SNAS uses a differentiable search space as defined by Liu et al. (2018)[5]. This search space is represented as a directed acyclic graph, consisting of a certain number of nodes. The edges between these nodes are represented as a softmax over all possible iterations, making the search space differential and traversable through search gradient. Xie et al. (2018)[3] improved upon this search gradient, proving that it optimizes the same objective as reinforcement-learning-based NAS, but is more efficient in structural decision. SNAS finds two types of cells: a *Normal Cell*, which preserves the feature map size of the input, and a *Reduction Cell*, which reduces the feature map size. These cells are stacked on top of each other to form an architecture consisting of three blocks of normal cells, with reduction cells at 1/3 and 2/3 of the total depth of the network. However, this architecture is part of a larger meta-architecture, which defines the way these cells stack and interact. The normal and

reduction cells found by SNAS are used to answer the following research question. What effect does the architecture of the cells found by SNAS have on the accuracy, and on the latency and power usage on an embedded system? To answer this question, a sample of possible architectures within a meta-architecture is selected, trained, and evaluated, comparing the size of the network and the parameters named in the research question. The aim of this research is to see if different architectures formed from the same cells will have a better performance in terms of power and latency on an embedded system while retaining an acceptable accuracy while using the same cells.
This paper is structured as follows: First the background information of the broader topic is given, second the method and approach are outlined, third the results are layed out and discussed, and finally a conclusion follows.
The full implementation of this research can be found at https://github.com/ivanipenburg/thesis, based on code publicly released by Liu et al. (2019)[5] and Xie et al. (2019)[3].

2 Background information

2.1 Neural architecture search and cell-based neural architectures

The manual design of neural architectures is called *architecture engineering.* This process is often laborious, as dozens of neural architectures need to be designed, trained, and tested. The automation of this process is called *Neural Architecture Search* (NAS). NAS methods have outperformed manually designed architectures on various machine learning tasks[1].
NAS methods consist of three different dimensions: search space, search strategy, and performance estimation strategy. The search space this thesis will focus on are *cell-based neural architectures.* These architectures consist of one or more types of cells that are repeatedly stacked on top of each other. These types of cells all have the same structure, but different weights. Zoph et al. (2018)[4] used two types of convolutional cells in their NAS algorithm; the *Normal Cell* and the *Reduction Cell.* The Normal Cell preserves the feature map size of the input, while the Reduction Cell reduces the feature map size. Each cell takes the input of the previous two cells, and consists of an acyclic graph of nodes, where each edge is either a convolutional operation or a skip operation, which eliminates the edge.
Cell-based neural architectures have several advantages over other neural architectures used in NAS. Elsken et al. (2018)[1] discussed three of these advantages. Firstly, as cells usually consist of significantly less layers than whole architectures, the search space of the NAS algorithm is reduced. Secondly, these architectures can more easily be transferred or adapted to other data sets. Thirdly, architectures created by repeating building blocks have proven a useful design principle in general. Following from these advantages, cell-based neural architectures have been successfully employed in many works, which will be discussed in the following section.

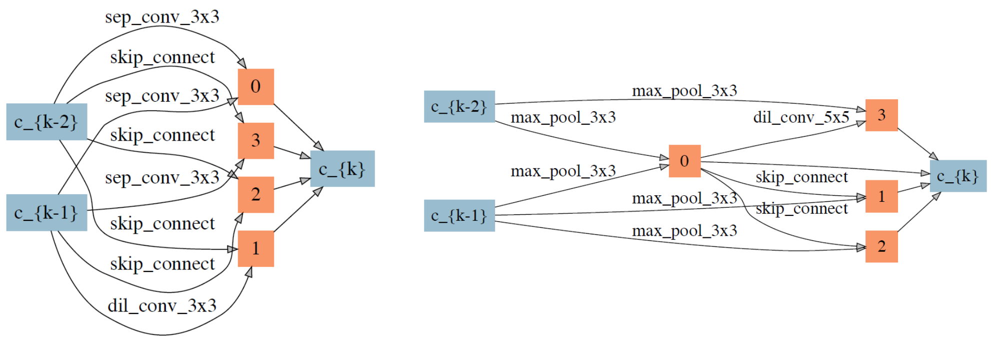
2.2 Stochastic Neural Architecture Search (SNAS)

Zoph et al. (2018) [4] were one of the first to employ such a cell-based neural architecture in a NAS algorithm. They called this search space the *NASNet search space*, and a convolutional architecture found in this search space a *NASNet.* One NASNet that was found by the method achieved a 2.4% error rate on the CIFAR-10[6] dataset. However, the search cost of this method is 1800 GPU days, which is significantly higher compared to later methods. This is because a reinforcement learning search method was employed, which is computationally expensive.
Liu et al. (2018) improved on this search cost significantly by employing a more efficient architecture search called *DARTS* (Differentiable ARchiTecture Search)[5].

Their method was orders of magnitude faster, taking only 1.5 and 4 days for the DARTS first order and second order architectures respectively. This increase in speed was caused by a different search method based on gradient descent. Instead of using a discrete search space like NASNet did, DARTS is based on the continuous relaxation of the architecture representation, thus allowing the use of gradient descent to explore the search space. However, this method did come with a slight increase in test error, achieving 3.00% and 2.76% respectively compared to 2.4% achieved by NASNet.
Xie et al. (2018)[3] used DARTS as the basis for *Stochastic Neural Architecture Search* (SNAS), building further on their differential search space. The search space is represented as a directed acyclic graph, also called a parent graph, consisting of a certain number of nodes. The edges of the graph are the possible convolutional operations between nodes. The skip operation is also included, which is enforced by having the nodes be ordered and having edges only point to nodes with higher indexes. Each cell is designed to take the output of the two previous cells, which in turn is the concatenation of all intermediate nodes in a cell. To make the search space differential, DARTS relaxed the categorical choice of a particular operation to a softmax over all possible operations, thus allowing for a gradient search over these operations. To derive the discrete architecture, the strength of each of the strongest operations is defined, and the operation with the highest strength is applied.
SNAS improves upon DARTS by assigning credits to structural decisions more efficiently, thereby improving the speed of the gradient descent. Resource constraints are also taken into account by limiting the parameter size of a cell. Three levels of resource constraint are used: mild, moderate, and aggressive. For each of these levels a normal cell and a reduction cell are found. These cells could have the same architecture, with the only diffference being the reduction cell using a stride of two for the cell's inputs. However, Zoph et al. (2018)[4] found that it is beneficial to learn two separate architectures. This research uses the mild constraint cells found by SNAS, as shown in Figure 1.

2.3 Embedded systems

An embedded system is a computer system which normally performs a single function within a larger system. It consists of at least three components: a processor to carry out computations, memory, and input and output devices. Embedded systems are seen in many aspects of everyday life; examples include electronic parking meters, home security systems, digital thermometers, and electronic safes. Often these systems are tightly constrained in terms of cost, size, power, and performance. In many cases they must be mass produced for a low cost, fit on a single chip, work efficiently on a battery, and carry out tasks in real-time. These



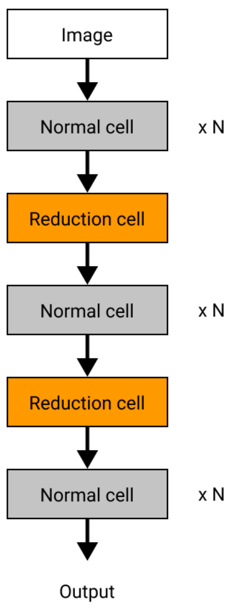
(a) Normal cell

(b) Reduction cell

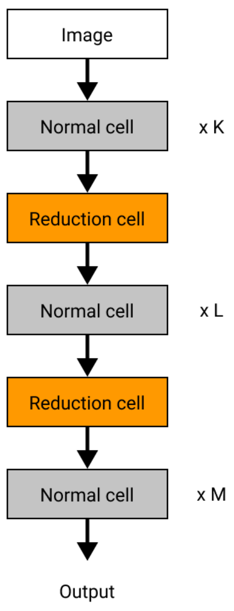
Figure 1: Cells found by SNAS (mild constraint) on CIFAR-10. Figure by Xie et al.[3]

.

constraints not only apply to the hardware of the system, but also in part to the software, which has a significant effect on the power usage and performance.
Image recognition performed by a convolutional neural network (CNN) is an example of such software which could be run on an embedded system. However, the majority of state-of-the-art CNNs are not designed with efficient power usage and fast performance taken into account [2]. Of the networks that do, even fewer are designed with performance on embedded or mobile systems in mind. Performance on these systems is different from those which these networks are usually tested on, like high-end GPUs and cloud-computing clusters, as the specifications of those non-embedded systems are designed around different goals and constraints.
While the focus of most CNNs is not on the performance on embedded systems, the demand for this will increase in the future as more embedded systems utilize deep learning image recognition techniques. In 2009, it was estimated that 98 percent of all microprocessors are manufactured for use in embedded systems[7], which shows that embedded systems should be a major focus in the design of CNNs. Bianco et al. [2018] have shown the performance of a majority of deep neural network architectures on both a workstation and an embedded system, including some architectures found with NAS algorithms. As the paper introducing the DARTS search algorithm was published after the aforementioned paper, neither DARTS nor later cell-based architectures based on it are included in the benchmark analysis. Therefore, research into the performance of these architectures on embedded systems, and changes in the meta-architecture which would lead to improvements in performance, could be valuable for other researchers and system designers.



(a) Balanced Architecture



(b) Non-Balanced Architecture

Figure 2: A representation of both a balanced and a non-balanced architecture consisting of repeated motifs of normal cells and reduction cells. NASNet, DARTS, and SNAS use a balanced architecture with $N = 6$ for CIFAR-10.

3 Method and Approach

3.1 Meta-architecture

As stated in the previous section, cell-based neural architectures are neural architectures which consist of one or more types of cells that are repeatedly stacked on top of each other. The way in which these cells are stacked and interact is called the *architecture*, while all the ways these cells can be stacked and interact is called the *meta-architecture*. NASNet employs a balanced architecture, with a total of 20 cells with reduction cells at $1/3$ and $2/3$ of the total depth of the network, as shown in Figure 2a. The reduction cells reduce the feature map size by a factor of two, while doubling the number of filters in the output. Both DARTS and SNAS base their architectures on this architecture.

Another parameter of the meta-architecture is the amount of initial channels (or depth) the network starts with. SNAS, like DARTS, initializes the network with 36 channels, which is doubled at both reduction cells. This value was chosen in

the DARTS paper, such that the total model size is comparable to other baselines in the literature. SNAS retains this initial channel value, resulting in a model size of 2.9 million parameters.
As mentioned before, the SNAS architecture consists of 20 cells with reduction cells at 1/3 and 2/3 of the total depth of the architecture. This results in three blocks of normal cells. In the SNAS architecture, these blocks all contain 6 normal cells. This research will explore different architectures within the meta-architecture. To indicate these architectures, we will use the following notation: $K-L-M$, where K is the amount of cells in the first block, L the amount in the second, etc. Thus, the original SNAS architecture is denoted as $6-6-6$.

3.2 Search space sampling

The chosen meta-architecture in which the architecture search is conducted consists of a total of 20 cells, with reduction cells placed at $K+1$ and $K+L+2$ depth of the network, for which $K \geq 2, L \geq 2, M \geq 2$, as illustrated in figure 2b. The amount of initial channels is sampled at an interval of 12, with a maximum of 36 channels.

3.3 Block search

To sample the search space, the distance measure between architectures is defined as a three-dimensional Manhattan distance. For meta-architectures a and b:

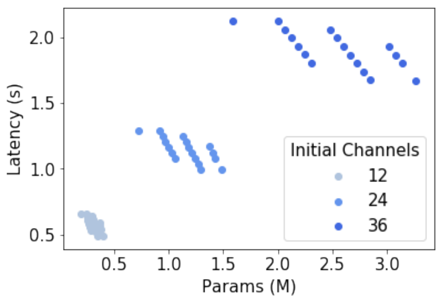
$$d = |K_a - K_b| + |L_a - L_b| + |M_a - M_b| \tag{1}$$

The architectures that were trained, were selected from the meta-architecture as follows; A list of all possible K-L-M meta-architectures is generated and sorted using radix sort, which prioritizes values in the order K, L, M.
In order to prevent architectures that are too unbalanced, K, L, M can not differ from each other individually more than a balance factor b. The value $b = 6$ was chosen for this research. Architectures in the list which do not adhere to this rule are removed.
Next, architectures that are too similar will be removed. Starting from the $6-6-6$ balanced architecture, and moving up and down the ordered list from there, architectures with a distance $d < 4$, as defined in Equation 1, between itself and the last valid architecture will be removed.
For the final step, architectures with a trainable parameter size of 3.3 million parameters or more are removed, as to be able to fit the networks on a single GPU and reduce training times. The final sample consists of 18 unique $K-L-M$

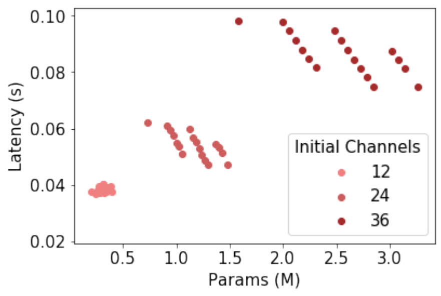
iterations, which will each be trained with 12, 24, and 36 initial channels, resulting in a total of 54 meta-architectures to be trained.

3.4 Architecture evaluation

For evaluation, all sampled meta-architectures are trained following the evaluation settings of SNAS; all networks are trained from scratch for 600 epochs with batch size 96 on the CIFAR-10 data set. Following existing works, cutout, path dropout of probability 0.2 and auxiliary towers with weight 0.4 are employed.
Latency is measured both on a NVIDIA Jetson Nano, a developer kit for embedded applications, and a NVIDIA T4, a high-end GPU, with batch size 64 (single batch results in severely under-utilized GPU for both systems). The Jetson Nano is an embedded system which is, as previously discussed, extremely constrained in resources compared to high-end GPUs. This has such a significant impact, that the power measurements must be done separately from the latency measurements, as they slow down the network by at least 33%. As we are only concerned with resource constraint of embedded systems, the power usage is only measured on the Jetson Nano with batch size 64. Both latency and power usage are measured over 50 individual runs of 50 batches. The first 20 runs are discarded as the measurements approach a constant mean. The first 5 batches of each run are also discarded, in order to remove any outliers.



(a) NVIDIA Jetson Nano



(b) NVIDIA T4

Figure 3: Latency of the architectures on the NIVDIA Jetson Nano and the NVIDIA T4, showing the latency difference between embedded systems and high-end GPUs.

4 Results

Figure 3 shows the enormous difference in latency between an embedded system and a high-end GPU , with latency on the embedded system ranging from 0.4 to 2.1 seconds, while the latency on the high-end GPU ranges from 0.03 to 0.10 seconds, which is an order of magnitude in difference. This illustrates the importance of efficient architectures on embedded systems.
The set of the best architectures found in the meta-architecture is determined by the Pareto front of the parameters of all architectures. A Pareto front is a set of solutions in which no parameter can be improved without sacrificing at least one other parameter[8]. In other words, for every architecture in the Pareto front, there should be no architecture which is equally good or better than it in all objectives. This set is useful, as it gives the ability to make trade-offs between different objectives, while having to consider a smaller set of points, which in this case are architectures. To provide better insight into the architectures, parameters should be at least better by a factor of 0.1.

In Table 1, the architectures of which the accuracy, number of parameters, latency, and power usage form a multi-dimensional Pareto front, are shown. For each of these architectures, the definition above holds, as no parameter can be improved without sacrificing at least one other parameter. All architectures in this set have a smaller number of parameters and a lower latency and power usage. The accuracy of the architectures are all comparable to previous SNAS architectures, with the $3-9-6$, $4-10-4$, $5-7-6$, $6-4-8$, and $7-7-4$ architectures achieving an accuracy higher than the margins given by the $6-6-6$ architecture, which was the balanced architecture employed by Xie et al (2019)[3]. The $6-4-8$ architecture with 36 initial channels performs the best in terms of accuracy, achieving

Architecture	Init Channels	Accuracy (%)	Params (M)	Latency (s)	Power Usage (W)
SNAS (6 - 6 - 6) (Xie et al., 2019)[3]	36	97.02	2.9	1.87	4.70
SNAS (3 - 9 - 6)	36	97.20	2.8	1.68	4.66
SNAS (4 - 10 - 4)	36	97.19	2.3	1.81	4.82
SNAS (4 - 10 - 4)	24	96.94	1.1	1.08	4.52
SNAS (4 - 10 - 4)	12	96.14	**0.3**	**0.53**	4.00
SNAS (5 - 7 - 6)	24	97.04	1.2	1.08	4.58
SNAS (6 - 4 - 8)	36	**97.49**	3.1	1.80	4.66
SNAS (7 - 7 - 4)	36	97.28	2.1	2.00	4.80
SNAS (8 - 8 - 2)	24	96.92	0.7	1.29	4.43
SNAS (9 - 3 - 6)	12	95.84	0.3	0.64	**3.88**

Table 1: Accuracy, number of parameters, inference time, and power usage of multi-dimensional Pareto front SNAS architectures on CIFAR-10. Latency and power usage are measured on the NVIDIA Jetson Nano with batch size 64. All architectures use the SNAS mild constraint cells with cutout.

97.49% compared to 97.02% achieved by the original SNAS architecture with a comparable parameter size. Furthermore, the $7-7-4$ architecture has a smaller parameter size, while still improving upon the accuracy, achieving 97.28%. However, the architectures which achieve a slightly lower accuracy, while improving on the latency and power usage significantly are also of importance. As mentioned before, embedded systems are highly resource constrained. Thus, sacrificing some precision for efficiency is often acceptable, as not all systems require perfect accuracy. The $4-10-4$ architecture with 12 initial channels, for example, is up to three times times faster and consumes about 15% less power, while still achieving an accuracy of 96.14%. The full data for all sampled architectures can be found in Appendix A.

An issue that should be brought up is that validation accuracy on the CIFAR-10 dataset is subject to high variance even with the same set-up (Liu et al., 2018[9]): DARTS and SNAS report the mean and standard deviation of 10 independent runs for their model. However, SNAS reports no standard deviation for their mild constraint architecture. Therefore, while a higher accuracy can be claimed, in future work this set-up should be utilized for a fair comparison.

Figure 4 shows the two-dimensional Pareto fronts of different sets of parameters, first comparing latency, power, and number of parameters to accuracy, and then latency and power to the number of parameters. In each graph, three clear clusters are clearly visible, which indicate the three different settings for the number of initial channels of the architectures. These graphs are useful for practical situations where only two factors are of importance. The best architecture for the application can then be selected based on these factors.

An interesting pattern emerges in Figure 4d, which can also be found in the com-

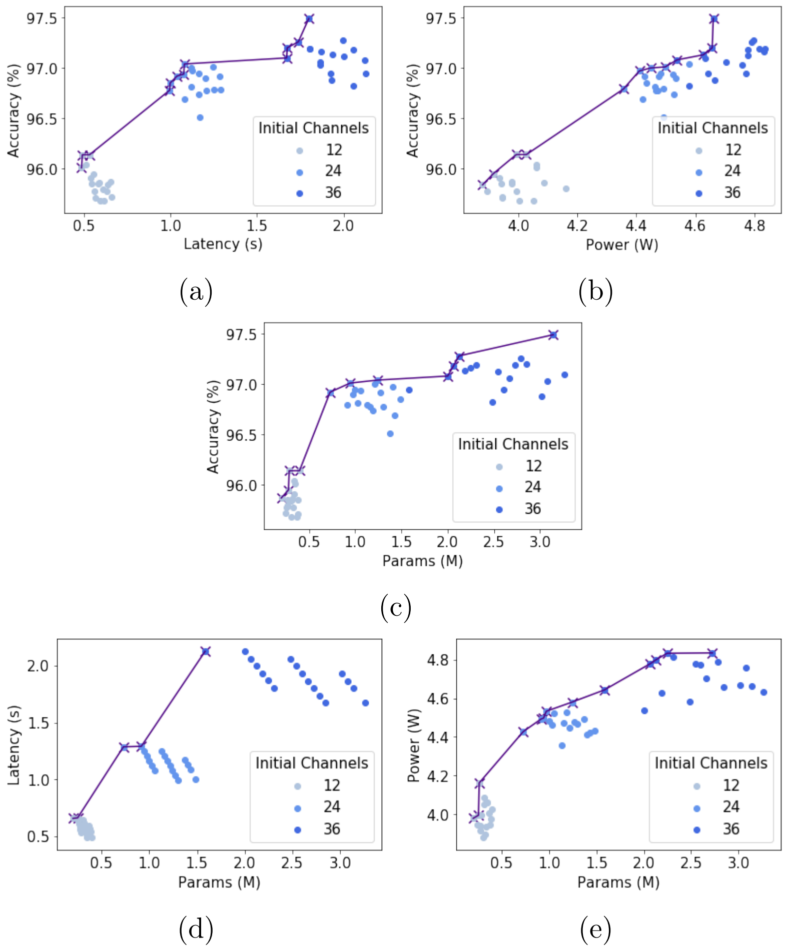


Figure 4: Pareto fronts of (a) latency, (b) power, and (c) parameter size compared to accuracy and of (d) latency, and (e) power compared to parameter size. Latency and power are averaged over several runs with batch size 64.

parison between latency on the NVIDIA Jetson Nano and the NVIDIA T4 as shown in Figure 3. This pattern, which shows that there is a direct correlation between latency on a high-end GPU and an embedded systems, can be used in future work to show that optimizing for both objectives is essentially equivalent.

5 Conclusion

The aim of this research was to explore the effect that different architectures within the meta-architecture of the cells found by SNAS have on the accuracy, and the latency and power usage on an embedded system. In order to do this, a meta-architecture was defined and a sample of this search space was trained and evaluated. A total of 54 architectures were evaluated. Among these architectures several improved upon the accuracy of the SNAS architecture. Of the architectures that are included in the Pareto set, all improved upon the SNAS architecture in terms of latency and power usage on the NVIDIA Jetson Nano embedded system. As mentioned in the Results section, in future work, a mean of 10 independent runs should be taken for an architecture, in order to make a fair comparison between their performance. Furthermore, increasing the scope of the meta-architecture to contain different amounts of total cells, varying reduction factors, and more should be explored as well as the transferability of the found architectures to new datasets, which is seen as an important advantage of cell-based neural architectures.

References

[1] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018.

[2] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[3] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

[4] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[5] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

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

[7] Michael Barr. Real men program in c. *Embedded systems design*, 22(7):3, 2009.

[8] M Janga Reddy and D Nagesh Kumar. Elitist-mutated multi-objective particle swarm optimization for engineering design. In *Encyclopedia of Information Science and Technology, Third Edition*, pages 3534–3545. IGI Global, 2015.

[9] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Appendix A

Architecture	Init Channels	Accuracy (%)	Params (M)	Latency (s)	Power Usage (W)
3 - 9 - 6	36	97.20	2.8	1.68	4.66
	24	96.78	1.3	1.00	4.47
	12	96.01	0.3	0.49	4.06
4 - 6 - 8	36	97.10	3.3	1.67	4.63
	24	96.85	1.5	1.00	4.43
	12	96.14	0.4	0.49	4.03
4 - 8 - 6	36	97.26	2.8	1.74	4.79
	24	96.92	1.3	1.04	4.48
	12	96.04	0.3	0.51	4.06
4 - 10 - 4	36	97.19	2.3	1.81	4.82
	24	96.94	1.1	1.08	4.52
	12	96.14	0.3	0.53	3.99
5 - 7 - 6	36	97.19	2.7	1.80	4.83
	24	97.04	1.2	1.08	4.58
	12	95.91	0.3	0.54	3.94
5 - 9 - 4	36	97.16	2.2	1.87	4.83
	24	96.81	1.0	1.12	4.46
	12	95.95	0.3	0.56	3.92
6 - 4 - 8	36	97.49	3.1	1.80	4.66
	24	96.69	1.4	1.08	4.42
	12	95.85	0.4	0.54	3.98
6 - 6 - 6	36	97.06	2.7	1.87	4.70
	24	97.00	1.2	1.12	4.45
	12	95.78	0.3	0.56	3.89
6 - 8 - 4	36	97.13	2.2	1.93	4.63
	24	96.95	1.0	1.16	4.48
	12	95.85	0.3	0.58	3.94
7 - 3 - 8	36	97.03	3.1	1.86	4.76
	24	96.97	1.4	1.13	4.41
	12	95.71	0.4	0.57	3.95
7 - 5 - 6	36	96.95	2.6	1.93	4.77
	24	96.74	1.2	1.16	4.53
	12	95.86	0.3	0.59	4.09
7 - 7 - 4	36	97.28	2.1	2.00	4.80
	24	96.90	1.0	1.20	4.53
	12	95.80	0.3	0.61	4.16
8 - 2 - 8	36	96.88	3.0	1.93	4.67
	24	96.51	1.4	1.17	4.49
	12	95.68	0.4	0.59	4.00
8 - 4 - 6	36	97.12	2.5	2.00	4.78
	24	96.78	1.2	1.21	4.47
	12	95.68	0.3	0.61	4.05
8 - 6 - 4	36	97.18	2.1	2.06	4.78
	24	97.01	0.9	1.25	4.50
	12	95.78	0.3	0.64	4.00
8 - 8 - 2	36	96.95	1.6	2.13	4.64
	24	96.92	0.7	1.29	4.43
	12	95.87	0.2	0.66	3.98
9 - 3 - 6	36	96.82	2.5	2.06	4.58
	24	96.79	1.1	1.25	4.36
	12	95.84	0.3	0.64	3.88
9 - 5 - 4	36	97.08	2.0	2.12	4.54
	24	96.79	0.9	1.29	4.49
	12	95.72	0.2	0.66	3.95

Figure 1: Accuracy, number of parameters, latency, and power usage of SNAS architectures on CIFAR-10. Latency and power usage are measured on the NVIDIA Jetson Nano with batch size 64. All architectures use the SNAS mild constraint cells with cutout.