Exploring Cell-Based Neural Architectures for Embedded Systems

van Ipenburg, I.; Sapra, D.; Pimentel, A.D.

DOI
10.1007/978-3-030-93736-2_28
Publication date
2021
Document Version
Final published version
Published in
Machine Learning and Principles and Practice of Knowledge Discovery in Databases
License
Article 25fa Dutch Copyright Act

Exploring Cell-Based Neural Architectures for Embedded Systems

Ilja van Ipenburg, Dolly Sapra(✉), and Andy D. Pimentel

University of Amsterdam, Amsterdam, Netherlands
i.vanipenburg@student.uva.nl, {d.sapra,a.d.pimentel}@uva.nl

Abstract. *Neural Architectures Search* (NAS) methodologies, which automatically discover state-of-the-art neural networks, have seen a growing interest in recent years. One particular group of NAS methodologies searches for small sub-networks called cells, which are then linearly connected to form the complete neural network. The composition of the final neural network, established through the width of the cells and the depth of the connections, is manually designed while being influenced by the available GPU memory. Typically, the hardware architectures targeted in NAS research are powerful, high-end GPUs. Hence, the attention is on creation of a large neural network that will still fit in the GPU, in turn leading to a very high accuracy for the given task. In direct contrast, we exploit the inherent flexibility of cells to create smaller neural networks, with the intention to study their behaviour on resource-constrained embedded systems. We use the cells discovered from *Stochastic Neural Architecture Search* (SNAS), to explore the effect that the composition of the cell has on various metrics, namely, the number of parameters, accuracy, latency and power usage. The last two metrics are measured on NVIDIA Jetson Nano, an embedded AI computing platform with a small GPU with mere 4GB on-chip memory. When comparing results of our exploration to the original SNAS architecture's with 97.02% accuracy for the CIFAR-10 dataset, one particular architecture, with only a tenth of original parameters, achieved an accuracy of 96.14%, notably with 15% lower power consumption and ≈3x faster inference time. Furthermore, this model outperforms other architectures, which are designed for edge devices, specifically to reduce the model size. Thus demonstrating that cell-based architectures, with adequate composition, provide efficient models to be deployed on resource-constrained edge devices.

1 Introduction

Recently, there has been a growing interest in *Neural Architecture Search* (NAS), the automation of architecture engineering for efficient neural networks. The automatically discovered neural architectures have routinely outperformed the hand-designed ones in a variety of domains such as language processing and image classification tasks [1]. There is an important and increasingly popular

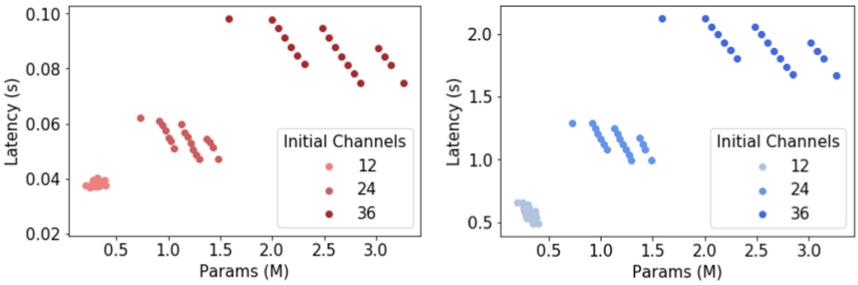
M. Kamp et al. (Eds.): ECML PKDD 2021 Workshops, CCIS 1524, pp. 363–374, 2021.
https://doi.org/10.1007/978-3-030-93736-2_28

subgroup of NAS methodologies, consisting of algorithms focused on cell-based neural architectures. As discussed in [2], cell-based NAS has several advantages over other methodologies. Firstly, the search space of the NAS algorithm is reduced as the algorithm only searches for a small sub-network called *cell*, which is a small part of a complete neural architecture. Secondly, these cells can be transferred and re-used in different datasets and domains. Thirdly and most importantly in this work, the architectures created by repeating building blocks are a useful design principle in general.

The cell-based NAS typically discovers two types of cells, namely, a normal cell and a reduction cell [3]. The normal cell is designed to maintain the feature map size of the input, whereas the reduction cell reduces the feature map size. The complete neural architecture is generated by forming a linear connection of the normal cells, interrupted by a few reduction cells at regular intervals. The neural architectures created by repeating the same cell possess an inherent flexibility to be able to form neural networks of different sizes. Individual cells can be wide or narrow (depending on the number of filters it has), and the variable frequency of cell repetition in the neural network further adds to their flexible nature.

The GPUs targeted in NAS research typically are state-of-the-art hardware, usually resulting in the creation of a large neural network, containing a stack of many wide cells. However, most NAS methodologies do not consider hardware limitations during engineering. The neural networks designed for a high-end GPU may not be a viable option for an edge device. Oftentimes, the edge devices are cheap, able to fit in small spaces, and run on an internal battery. Consequentially, they have constraints on memory, processing power, speed, and energy. Figure 1 illustrate the enormous difference between the performance of neural networks on a high-end GPU (NVIDIA Tesla T4) and a resource-constrained GPU (NVIDIA Jetson Nano). The graphs are drawn for latency vs parameters for various cell-based architectures. In a glance they do look similar, however the scale of the latency for these two devices are in stark contrast. An architecture designed for a high-end GPU, providing inference in 0.1 s can take more than 2 s on an embedded device, driving the point that the best model for one device can not always be the optimal choice for another device. This also demonstrates the importance of designing efficient neural network architectures for embedded systems.

Taking the flexibility of the cell-based neural architectures into consideration, the question arises if it is possible to alter and organize the cells, such that it is feasible to be deployed on resource-constrained edge devices. This question motivates the current work, where we first analyze the performance of the original architecture created by a popular NAS cell on our target hardware. Subsequently, this information is utilized to design a grid search, which evaluates various architectures created from the chosen cell. The objective of this search is to discover good architecture(s), composed from the same cell, which have better performance in terms of power and latency on an embedded system while retaining an acceptable accuracy. In other words, the aim is to investigate the



(a) NVIDIA T4 (b) NVIDIA Jetson Nano

Fig. 1. Latency of various architectures on the NVIDIA T4 and the NVIDIA Jetson Nano, illustrating the latency difference between embedded systems and high-end GPUs.

cell-based architecture's suitability to an embedded system. As far as we know, this is the first experimental and exploratory study that systematically analyses cell-based architectures for deployment on embedded devices.

Specifically, in this work, the cells chosen were those discovered from a NAS methodology called *Stochastic Neural Architecture Search* (SNAS). We studied the effects of the composition of the cells with a variable number of parameters, on its accuracy along with latency, and power usage on NVIDIA Jetson Nano, which is an embedded AI computing platform with a small GPU with mere 4 GB on-chip memory. The results of this exploration illustrate that the architectures, composed of the same cells that were originally designed for high efficiency and a large memory footprint, are able to achieve competitive performance on another hardware with resource limitations.

Since the search is based on multiple objectives, selection of the best candidates is concluded through *Pareto optimization*, where any objective cannot be improved without worsening some of the other objectives. The set of candidates selected in such a fashion are collectively called as a *Pareto Front*. The Pareto Front obtained upon convergence, presents the various possible architectures that can be deployed on the edge device. It allows the designer to be aware of the trade-offs that exist between different evaluation parameters. For instance, a highly efficient model generally has many convolutional kernels, thereby having a large memory footprint and a longer processing time. On the other hand, a smaller model with fewer parameters is highly likely to have less prediction accuracy, but might be the best option for a resource-constrained device.

The remainder of this paper is structured as follows. Firstly, in Sect. 2, related works in the domain of neural architectures for embedded systems is discussed. Secondly, in Sect. 3, we discuss the tradition composition of cell based architecture and its behaviour on resource-constrained Jetson-Nano. Subsequently, Sect. 4 presents our search methodology and the evaluation objectives. Next, the results from experiments are described in Sect. 5. Lastly, Sect. 6 concludes the paper.

2 Related Work

Neural networks are increasingly being used in resource-constrained edge devices for various tasks and domains [4]. This has led to research on novel neural architectures, which specifically cater to the resource limitations on the target hardware. New architectures have been proposed that are manually designed, in addition to NAS methodologies, that have been defined to automatically discover the hardware aware architectures.

In the past few years, various manually-designed architectures for embedded systems have been proposed, such as, MobileNets [5], ShuffleNet [6], DenseNet [7] and CodenseNet [8]. These neural architectures are generally designed to reduce the resource usage, however, they do not evaluate the performance on a specific target hardware. They require significant design time, in addition to the human expertise. Moreover, they are not always optimal for a new application or hardware, and may require further manual effort to fine-tune.

Early NAS methodologies [2] were focused only on improving the accuracy and took many days to converge [9,10]. With the introduction of faster differentiable search algorithms, which only takes a few hours to converge on a cell-based search space, the cell-based NAS methodologies have become mainstream [11–13]. Consequently, there are many efficient known cells, however, once a cell is discovered, all architectures are constructed in a similar manner (as first proposed in [9]).

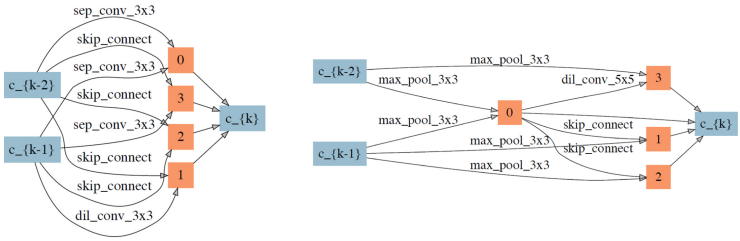
Specific hardware aware NAS methodologies, such as MnasNet [14], PPP-Net [15], are efficient in searching for neural networks that achieve high accuracy, low computation cost, and low latency on a specific device. However, most of them do not generate flexible architectures (or cells) that can be adapted after the NAS algorithm has finished execution. For example, PPP-Net and MnasNet, both generate architectures with pre-defined number of blocks.

The inherent flexibility of a cell however, allows it to be used in various configurations to construct the whole architecture. Motivated by this thought, the current work is an exploratory study that investigates the architectures constructed from one such cell (SNAS [13]). To the best of our knowledge, no other study has been proposed to systematically analyse the effects of cell-based architecture composition for embedded systems.

3 Cell-Based Architecture Analysis

In this section, we first explain the baseline neural architecture, originally constructed by the cells discovered during the SNAS (Stochastic Neural Architecture Search) [13] work. Next, we analyse the baseline model for the CIFAR-10 dataset, on NVIDIA Jetson Nano and further utilize the details from this analysis to search for architectures that are suitable to resource-constrained devices.

The SNAS methodology discovers 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 by half. For the CIFAR-10 dataset, the original baseline



(a) Normal cell (b) Reduction cell

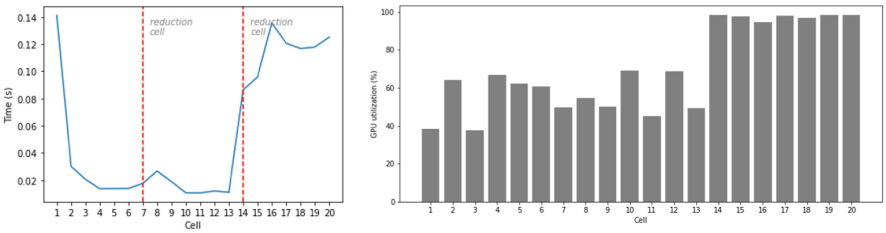
Fig. 2. Cells found by SNAS (mild constraint) [13].

model has linearly connected 18 normal cells, partitioned into three blocks by two additional reduction cells at 1/3 and 2/3 of the total depth of the network.

The cell itself is designed to take the output of the two previous cells, and as depicted in Fig. 2, consists of an acyclic graph of various nodes. In the figure, blue nodes are input/output nodes of the cell and orange nodes are the intermediate nodes. Each edge between an input node and an intermediate node is either a convolutional operation or a skip connection. The output of all the intermediate nodes is then concatenated to produce the final cell output.

In the SNAS work of [13], resource constraints were weakly considered during the search, by placing limits on the size of the cell. Three levels of resource constraints were used: mild, moderate, and aggressive. For each of these levels, a normal cell and a reduction cell were discovered. The mild constraint allows for relatively large cells to be discovered, whereas, the aggressive mode leads to discovery of cells with fewer parameters. However, the number of parameters in the aggressive-cell were still too high, the final model has ≈80% of the size of the model constructed from the mild-cell. This still may not be considered to be of suitable size to be deployed on many edge devices with memory limitations.

Furthermore, most NAS approaches construct the neural network in the same standard manner. The classic architecture was first introduced by [9] and has been deployed by many other cell-based NAS works [11–13]. These works spend considerable time and effort to discover the cells and subsequently construct the architecture as originally suggested. In the classic architecture, blocks of the normal cells are partitioned by the reduction cells at regular intervals. The standard composition for the eventual neural network is a balanced architecture, with an equal number of cells in each block. Figure 5a illustrates the baseline architecture composition commonly used for the CIFAR-10 dataset. In the figure, FxN to the right of each block represents the number of channels per cell in the block × number of cells per block. In the classic architectures, the number of cells per block is the same for all blocks, however, the number of channels of the cell gets doubled in subsequent blocks. The number of channels of a cell refers to the number of kernels in every convolutional operator edge in the cell.



(a) Average latency per cell per epoch

(b) Average GPU utilization per cell per epoch

Fig. 3. Analysis of cell behaviour on Jetson-Nano. The last block of the architecture is compute intensive, relying heavily on GPU based computation and is taking longer time to complete all its operations.

We analysed the behaviour of the baseline SNAS architecture on Jetson-Nano and one of the chief observations was, that the last block consumes most computational resources. It is unsurprising though, considering the fact that the last block has very wide cells, with 4x as many channels as compared to the cells in the first block. Figure 3 shows the behaviour of individual cells during an inference cycle. Cells 1 to 20 are consecutive cells in the order of execution during the inference. Figure 3a shows execution time of each cell and Fig. 3b shows GPU utilization during execution of each cell. It is evident from these graphs that the cells in the last block are utilizing the GPU to the maximum, as a direct consequence of the large number of computational operations required. Additionally, it takes between 5x–10x longer for each cell (in the last block) to finish execution as compared to cells in first and second block.

From this analysis, it is obvious that to achieve a suitable neural architecture composition for an embedded device, utilization of narrow cells should be ensured. Certain design principles from the classic architecture may still be retained, such as, the pattern of doubling the number of channels after every reduction cell can be maintained, by reducing the number of initial channels in the first block. Another alternate approach may be to reduce the number of cells only in the last block. The total number of cells can also be retained by varying the numbers of cells in different blocks and thereby creating an unbalanced architecture. We eventually employ these strategies in a grid search methodology to explore the architecture composition for embedded systems.

4 Architecture Search

In this section, we explain the methodical neural architecture search approach using the SNAS cells and their evaluation process on the target hardware. We expect a cell from any other NAS methodology will also provide a similar exploration result on an embedded device, since all existing NAS algorithms favor a similar cell with wide and shallow structure [3].

4.1 Search Space

To indicate different architectures, we use the following notation: $(C@K-L-M)$, where C is the number of channels in the first block and K,L,M are the number of cells in the first, second and third block respectively. Thus, the original SNAS architecture in Fig. 5a is denoted as $(36@6-6-6)$.

The neural architectures generated by our search algorithm maintain a constant depth of 20 cells, with reduction cells placed at $K+1$ and $K+L+2$ depth of the network, for which $K \geq 2, L \geq 2, M \geq 2$. The amount of initial channels is sampled at an interval of 12, with a maximum of 36 channels.

4.2 Grid Search

To evaluate various neural architecture compositions, 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. The distance measure between architectures is defined as a three-dimensional Manhattan distance. For meta-architectures NN_1 and NN_2:

$$d = |K_1 - K_2| + |L_1 - L_2| + |M_1 - M_2| \tag{1}$$

In order to prevent architectures that are too unbalanced, the first stipulation on architectures is that the distance between baseline $(6-6-6)$ architecture and K, L, M can not differ from each other by more than a balance factor b. The value $b = 6$ was chosen for this research. Architectures in the list that do not adhere to this rule were removed.

Next, architectures that are similar to each other were removed after it was observed that architectures that were close to each other had comparable evaluated metrics, including accuracy. Moving through the sorted list, architectures with a distance $d < 4$, between itself and the last valid architecture were removed. This step was performed to reduce the number of architectures that would need to undergo a resource expensive training process.

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 were each trained with 12, 24, and 36 initial channels, resulting in a total of 54 architectures that were trained and evaluated.

4.3 Architecture Evaluation

For evaluation, all 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 for image classification. CIFAR-10 consists of $60,000$ labeled images of dimensions $32 \times 32 \times 3$, comprising of $50,000$ training and $10,000$ testing images. The images are divided into 10 classes. Standard data augmentation techniques [16] with small translations, cropping, rotations and horizontal flips along with cutout [17] were utilized during the training.

Latency is measured on the NVIDIA Jetson Nano developer kit for embedded applications. This hardware, as previously discussed, is extremely constrained in resources, when 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%. Both latency and power usage are measured and averaged over 50 individual runs of 50 batches and a batch size of 64 images.

5 Exploration Results

In this section, we present the evaluations and results of the cell-based architecture exploration. All the generated architectures are first trained and then the performance is measured on the target hardware specific metrics.

All training settings were the same as followed in [13], where every neural network was trained using stochastic gradient descent with initial learning rate 0.1, weight decay 3×10^{-5} and batch size 128. The learning rate was decayed by a factor of 0.97 after each epoch and auxiliary towers with weight 0.4 were used as additional enhancements. All neural networks were trained on NVIDIA Tesla T4 and on an average took 1.5 days to train completely.

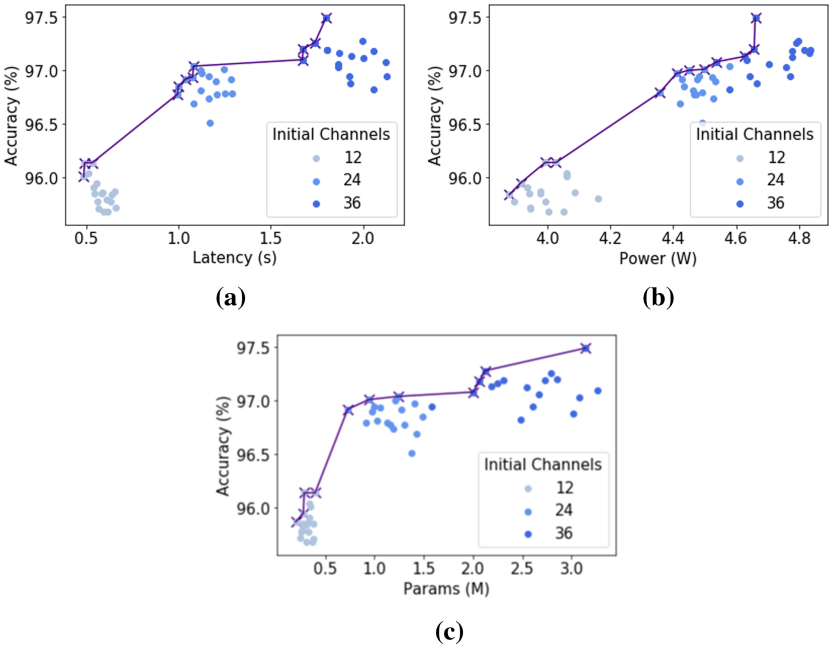


Fig. 4. Pareto fronts for accuracy of different architectures w.r.t. (a) latency, (b) power, and (c) parameter size.

Table 1. The multi-dimensional Pareto front for cell-based architectures on CIFAR-10 with accuracy, number of parameters, inference time, and power usage on Jetson-Nano as evaluation objectives. All architectures use the SNAS mild constraint cells with cutout. SNAS^{orig} refers to the original baseline architecture. SNAS^{c} refers to other architectures composed using the SNAS-cell.

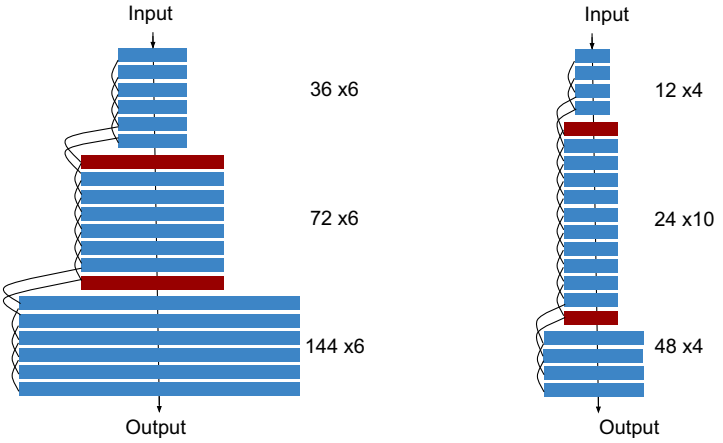
Architecture	Init-Channels	Accuracy (%)	Params (M)	Latency (s)	Power-usage (W)
SNAS^{orig} (6 - 6 - 6)	36	97.02	2.9	1.87	4.70
SNAS^{c} (6 - 4 - 8)	36	**97.49**	3.14	1.80	4.66
SNAS^{c} (7 - 7 - 4)	36	97.28	2.12	2.00	4.80
SNAS^{c} (3 - 9 - 6)	36	97.20	2.85	1.68	4.66
SNAS^{c} (4 - 10 - 4)	36	97.19	2.31	1.81	4.82
SNAS^{c} (5 - 7 - 6)	24	97.04	1.24	1.08	4.58
SNAS^{c} (4 - 10 - 4)	24	96.94	1.05	1.08	4.52
SNAS^{c} (8 - 8 - 2)	24	96.92	0.73	1.29	4.43
SNAS^{c} (4 - 10 - 4)	12	96.14	0.28	**0.53**	4.00
SNAS^{c} (9 - 3 - 6)	12	95.84	0.30	0.64	**3.88**
SNAS^{c} (9 - 5 - 4)	12	95.72	**0.25**	0.66	3.95

Once all the neural networks were trained, they were evaluated on four metrics, namely, accuracy, number of parameters, latency and power usage. The last two metrics were measured on the NVIDIA Jetson Nano developer board. Next, the Pareto Front was selected based on all evaluated metrics, which is presented in Table 1. All the models in the Pareto set are considered to be equally adequate to be marked as a good model. This set is a handy tool for system designers, as it provides the quantitative trade-offs between different objectives.

Considering it is not easy to draw and understand four-dimensional plots, Fig. 4 shows the two-dimensional Pareto fronts of different sets of parameters, comparing latency, power, and number of parameters to the accuracy of an architecture. Beside the visualization, these graphs also provide insight into the impact of initial channels on different evaluation metrics. In each graph, three clusters are clearly visible, which indicate the three different settings for the number of initial channels of the architectures.

Looking at the whole pareto front (Table 1), surprisingly, the $(36@6-4-8)$ architecture performs the best in terms of accuracy, achieving 97.49% compared to 97.02% achieved by the original balanced SNAS architecture with a comparable parameter size. This difference is more than what can be attributed to statistical difference. This architecture strongly suggests that the composition of the cells in a neural network plays an important role in the eventual performance of the model. Beyond the search for a cell, there is little manual effort required in training of some of the architecture compositions, and this can be a recommended segment of the underlying NAS methodology.

For the resource-constrained devices, sacrificing some precision for efficiency is often acceptable, since not all systems require perfect accuracy. From the exploration results, it is evident that the cell based architectures are still very



(a) SNAS original architecture (36@6-6-6) (2.9M parameters, 97.02% accuracy)

(b) SNAS^c Architecture (12@4-10-4) (0.3M parameters, 96.14% accuracy)

Fig. 5. Cell-based architectures for CIFAR-10 with three blocks of normal cells partitioned by two reduction cells (in red). FxN to the right of each block represents the number of channels per cell in the block × number of cells per block. (Color figure online)

efficient with fewer parameters. For example, the architecture $(24@5-7-6)$ with less than half the parameters of the baseline model, has a similar accuracy.

As expected, the models with the fewest parameters, fastest inference time and lowest power usage were all the architectures with 12 initial channels. Most research only takes the number of parameters into account when optimizing for an edge device. However, latency and power do not always have a linear relationship with the model size. One notable architecture, $(12@4-10-4)$, as shown in Fig. 5b, is up to three times faster and consumes about 15% less power, while still achieving an accuracy of 96.14%. This architecture has narrow cells, as well as fewer cells in the last block.

We compared the models on pareto front that had less than 1M parameters, to other state of the art neural architectures that were designed specifically for embedded systems. We use the number of parameters for the comparison, since latency and power usage cannot be fairly compared as either the target hardware is different or not included in the search. The results are presented in Table 2.

The neural networks explored in this work outperform all other small architectures when it comes to prediction accuracy. This result strongly indicates the advantages of constructing neural architectures for an edge device by utilizing cells that were discovered for high-end GPUs. Not only are they flexible and highly efficient, but can also expedite the design process of an application for a specific target hardware, specially when fast turn-around time is desired.

Table 2. Accuracy of various architectures for CIFAR-10 with <1M parameters.

Architecture	Type	Accuracy (%)	Params (M)
DenseNet-BC (k = 12) [7]	Non-cell	95.49	0.8
LEMONADE Cell 9 [18]	Cell	95.43	0.5
CondenseNet-86 [8]	Non-cell	95.0	0.52
CondenseNet^{light}-94 [8]	Non-cell	95.0	0.33
PPP-Net-A [15]	Non-cell	94.72	0.45
PPP-Net-B [15]	Non-cell	95.42	0.52
One-Shot Top (F = 16) [19]	Cell	94.6	0.7
One-Shot Small (F = 16) [19]	Cell	94.6	0.4
SNAS^c (24@8-8-2)	Cell	**96.92**	0.73
SNAS^c (12@4-10-4)	Cell	96.14	0.28
SNAS^c (12@9-3-6)	Cell	95.84	0.3
SNAS^c (12@9-5-4)	Cell	95.72	0.25

6 Conclusion

The aim of this research was to explore cell-based architectures and their composition strategies for an embedded system. In order to do this, a grid search was defined and a carefully truncated list of architectures was trained and evaluated. A Pareto Front was presented, describing the trade-offs between different evaluation metrics, namely, accuracy, number of parameters, along with the latency and power usage on the NVIDIA Jetson Nano developer board. Among these architectures several noteworthy architectures were found, highlighting the importance of the composition of cells, geared towards different situations.

As an extension to this work, we aim to explore more connection patterns to create the architectures from a cell. In the current work, there were many restrictions put on the search space, such as, the connection pattern of doubling the number of channels after every reduction cell was maintained. In the future, the aim is to create a search space where highly unbalanced architectures as well as different connection patterns can be explored through a search algorithm. We further aim to utilise a population based meta-heuristic algorithm [20] to cover this large search space to achieve a better pareto front.

Acknowledgements. This project has received funding from the EU Horizon 2020 Research and Innovation programme under grant agreement No. 780788.

References

1. He, X., Zhao, K., Chu, X.: AutoML: a survey of the state-of-the-art. Knowl.-Based Syst. **212**, 106622 (2021)
2. Elsken, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. arXiv preprint arXiv:1808.05377 (2018)

3. Shu, Y., Wang, W., Cai, S.: Understanding architectures learnt by cell-based neural architecture search. In: International Conference on Learning Representations (2020)
4. Wang, X., Han, Y., Leung, V.C., Niyato, D., Yan, X., Chen, X.: Convergence of edge computing and deep learning: a comprehensive survey. IEEE Commun. Surv. Tutor. **22**, 869–904 (2020)
5. Howard, A.G., et al.: MobileNets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
6. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)
7. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2017)
8. Huang, G., Liu, S., Van der Maaten, L., Weinberger, K.Q.: CondenseNet: an efficient DenseNet using learned group convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)
9. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2018)
10. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence (2019)
11. Pham, H., Guan, M., Zoph, B., Le, Q., Dean, J.: Efficient neural architecture search via parameters sharing. In: International Conference on Machine Learning (2018)
12. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: International Conference on Learning Representations (2019)
13. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: International Conference on Learning Representations (2019)
14. Tan, M., et al.: MnasNet: platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2019)
15. Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., Sun, M.: PPP-Net: platform-aware progressive search for pareto-optimal neural architectures (2018)
16. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net. arXiv preprint arXiv:1412.6806 (2014)
17. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)
18. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via Lamarckian evolution. In: International Conference on Learning Representations (2019)
19. Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning. PMLR (2018)
20. Beheshti, Z., Shamsuddin, S.M.H.: A review of population-based meta-heuristic algorithms. Int. J. Adv. Soft Comput. Appl. **5**(1) (2013)