DEff-ARTS: Differentiable Efficient ARchiTecture Search

Sulaiman Sadiq[†]
ss2n18@soton.ac.uk

Partha Maji[‡]
partha.maji@arm.com

Jonathan Hare[†]
jsh2@ecs.soton.ac.uk

Geoff Merrett[†]
gvm@ecs.soton.ac.uk

[†] University of Southampton, United Kingdom
[‡] ARM Ltd, Cambridge, United Kingdom

Abstract

Manual design of efficient Deep Neural Networks (DNNs) for mobile and edge devices is an involved process which requires expert human knowledge to improve efficiency in different dimensions. In this paper, we present DEff-ARTS, a differentiable efficient architecture search method for automatically deriving CNN architectures for resource constrained devices. We frame the search as a multi-objective optimisation problem where we minimise the classification loss and the computational complexity of performing inference on the target hardware. Our formulation allows for easy trading-off between the sub-objectives depending on user requirements. Experimental results on CIFAR-10 classification showed that our approach achieved a highly competitive test error rate of 3.24% with 30% fewer parameters and multiply and accumulate (MAC) operations compared to Differentiable ARchiTecture Search (DARTS).

1 Introduction

With the recent success achieved by DNNs on a variety of tasks there has been significant interest in developing efficient network architectures that can be deployed on resource constrained edge devices. A common approach is to manually hand-craft architectures. This is done in a number of ways including reducing filter sizes (Iandola et al., 2016) or using separable convolutions (Howard et al., 2017) amongst other appoaches. However, exploring the large space manually requires significant human effort. Other approaches involve quantisation of network weights/activations (Mishra and Marr, 2018; Courbariaux et al., 2015) or pruning (Han et al., 2015), but these techniques involve altering a pre-designed model. This has led to interest in automated architecture search to derive high-performing network architectures without the need for manual effort in the design.

Recent evolutionary optimisation (Real et al., 2019) and reinforcement learning (Zoph and Le, 2017; Tan et al., 2019) based approaches have produced state of the art network architectures by minimising classification loss on a task. However, these search methods have prohibitive search costs in the range of 10^3 GPU hours. In this paper, we utilise the machinery of DARTS (Liu et al., 2018) to derive architectures with a search cost three orders of magnitude smaller than other approaches. In order to search for efficient architectures, we formulate a differentiable closed form of the computational complexity which is minimised simultaneously with the cross-entropy loss. To measure computational complexity, other approaches have used inaccurate proxies such as FLOPS or physically measured inference latency which requires implementation of the architectures Tan et al. (2019) or its building

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Workshop on Machine Learning for Systems, Vancouver, Canada.

blocks Wu et al. (2019) on the target hardware. Our metric, the compute cost, involves analytically calculating the number of CPU cycles required to perform inference on the target hardware to account for varying characteristics of different computing devices.

DEff-ARTS combines the two sub-objectives through a linear combination with the trade-off being controlled through a single hyper-parameter enabling the discovery of architectures satisfying varying user requirements. By jointly optimising the sub-objectives, DEff-ARTS is able to derive high-performing novel network architectures with significantly lower computational complexity with no extra overhead in the search cost.

2 Differentiable Efficient Architecture Search

Similar to DARTS, DEff-ARTS employs a continuous relaxation over the search space to search for smaller computational blocks called *cells* which are stacked together to form more complex networks as required. In the continuous search space, all candidate operations are applied to representations in the cells and weighted by architecture weights, α to produce output representations through *mixed operations*. After some epochs of search, the candidate operations with the largest architecture weights are selected as the best operations to minimise the objective function.

Performance Loss, $\mathcal{L}_{per}$: In DEff-ARTS, the sub-objectives of cross-entropy loss, $\mathcal{L}_{ce}$ and *compute cost*, $\mathcal{L}_{com}$ are combined in the multi-objective function, the *performance loss*, $\mathcal{L}_{per}$. This function is shown in Eq. 1 where w are the network weights, Γ is the cost weightage hyperparameter used to control the trade-off between the sub-objectives, and β is the modulation parameter used to modulate the gradient of the compute cost and linearise the relationship between the sub-objectives.

$$\mathcal{L}_{per}(w, \alpha) = \mathcal{L}_{ce}(w, \alpha) + \Gamma \times \mathcal{L}_{com}(\alpha)^{\beta} \tag{1}$$

In the search process we seek to find the architecture, α that minimises the performance loss on the validation data-set with the optimal weights. The search problem can then be formulated as a multi-objective optimisation problem where the performance loss is minimised based on bi-level optimisation and approximate architecture gradients (Liu et al., 2018). This is shown in Eq. 3 where $\mathcal{L}_{per,val/train}$ are the validation and training performance loss respectively and w' are the network weights after one step of training used to approximate the optimal weights for the architecture encoded by α.

$$\min_{\alpha} \quad \mathcal{L}_{per,val}(w'(w, \alpha), \alpha) \tag{2}$$

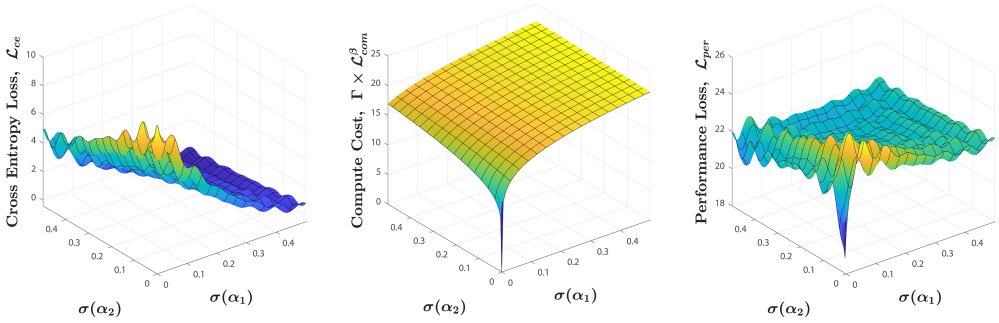
$$\text{s.t.} \quad w'(w, \alpha) = w - \xi \nabla_w \mathcal{L}_{per,train}(w, \alpha) \tag{3}$$

Compute Cost, $\mathcal{L}_{com}$: Our differentiable expression of the compute cost, $\mathcal{L}_{com}$ is shown in Eq. 4 where p is the number of mixed operations within a cell, q is the number of candidate operations in a mixed operation and $Cost_j$ is the compute cost of the jth candidate operation. The function $\sigma(\alpha)$, denotes the softmax operation on the architecture weights in the normal and reduction cells which is used as a proxy for a candidate operations strength and importance.

$$\mathcal{L}_{com} = \Sigma_{i=1}^{p} \Sigma_{j=1}^{q} \sigma(\alpha_{i,j}^{normal}) \times Cost_j + \Sigma_{i=1}^{p} \Sigma_{j=1}^{q} \sigma(\alpha_{i,j}^{reduce}) \times Cost_j \tag{4}$$

The cost, $Cost_j$ of candidate operations was calculated as the number of CPU cycles required to apply the candidate operation to a representation. Other approaches have used FLOPs to measure complexity (Gordon et al., 2018; He et al., 2018), however, these approaches assume all CPU operations have equal cycle requirements. For every candidate operation, we analytically calculate the number of cycles required according to the number of different CPU operations needed (e.g. MPY, ADD) to apply the operations and the cycle requirements of the CPU operations. The costs for every candidate operation are calculated offline and substituted in the compute cost to make it differentiable and enable gradient based optimisation.

Non-Linear Transformation of Compute Cost In composing the multi-objective metric, we applied a non-linear transformation through exponentiation of the compute cost similar to other approaches (Tan et al., 2019; Wu et al., 2019). We do this in order to sufficiently linearise the relationship between the sub-objectives. This enabled the trade-off between the sub-objectives, to be controlled through a simple cost-weightage hyperparameter, Γ. Secondly, without the transformation,



(a) Cross-Entropy Loss, $\mathcal{L}_{ce}$ (b) $\Gamma \times \mathcal{L}_{com}^{\beta}$, $\Gamma = 13$, $\beta = 0.14$ (c) Performance Loss, $\mathcal{L}_{per}$

Figure 1: **Optimisation Landscape Sketches** The $\sigma(\alpha_1)$ and $\sigma(\alpha_2)$ axes represent strengths for an expensive, expressive operation (e.g. convolution) and a cheaper operation (e.g. max pooling) respectively. **(1a)** The minima in the cross-entropy loss landscape would lie in the region where $\sigma(\alpha_1)$ was stronger. **(1b)** Due to the non-linear transformation, the compute cost measures complexity of operations on a comparable scale. **(1c)** Since the performance loss considers both sub-objectives, the minima moves to where the cheaper operation associated with $\sigma(\alpha_2)$ is stronger. The minima near the origin is ignored since the softmax on the architecure weights would sum to unity.

gradients back-propagated for an architecture weight from the monotonically increasing compute cost objective depend only on the particular weight itself. In reality, the strength of any operation would depend on the rest of the network structure and topology as well (Howard et al., 2017; He et al., 2016). Exponentiating the compute cost introduced a dependence on the rest of the cells architecture weights in the back-propagated gradients. Such techniques have also been used in multi-task learning works (Liang and Zhang, 2020). We further demonstrate these effects analytically in Appendix A. The transformation also modulated the magnitude of the the gradients, and compute cost to bring the sub-objectives to a comparable scale thus preventing any one from dominating the other in the optimisation landscape. This idea is demonstrated in Figure 1 through sketches of the landscapes where it can be observed that no sub-objective overrides the other and features of the sub-objectives are preserved in the performance loss landscape.

3 Experiments and Results

Search Space: For fair comparison of results, we used the cell structure and search space of Liu et al. (2018). The considered CPU operations and their cycle requirements are shown in Table 1. The candidate operations in the search space and their costs calculated as the CPU cycles required are shown in Table 2 where the intermediate representations sizes were assumed to be fixed and memory characteristics of the target hardware were not considered. Similar to previous NAS work (Liu et al., 2018; Zoph and Le, 2017; Real et al., 2019), separable convolutions were applied twice. Note in Table 2 that the max and average pooling have different cycle requirements due to the division operation executed in average pooling needing more CPU cycles. Analytical expressions used to calculate the CPU cycles for candidate operations are given in Appendix B.

Table 1: CPU cycles cost for considered low-level CPU operations on a Texas Instruments C64x+ 90nm CMOS processor. 32 bit precision was considered for all operations

CPU Operation	CPU Cycles
Comparison	1
Addition	1
Multiplication	4
Division	4

Table 2: CPU cycles cost for candidate operations

Candidate Operation	CPU Cycles
Max Pool 3×3	27,648
Avg Pool 3×3	30,720
(Sep Conv 3×3)$\times 2$	368,340
(Sep Conv 5×5)$\times 2$	860,160
Dil Conv 3×3	184,320
Dil Conv 5×5	430,080
Skip Connect	0

Table 3: Comparison with state-of-the-art image classifiers on CIFAR-10. Values besides DEff-ARTS were taken from Liu et al. (2018). All evaluated architectures are visualised in Appendix C.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method	GMACs	Compute Cost (CPU Cycles)
NASNet-A + cutout	**2.83**	3.1	2000	13	RL	0.624	5,861,376
AmoebaNet-A + cutout	**3.12**	3.1	3150	19	evolution	0.506	6,100,992
DARTS + cutout, ($\Gamma = 0.0$)	**2.76 ± 0.09**	3.3	**4**	7	gradient-based	0.547	1,244,160
DEff-ARTS + cutout, $\Gamma = 0.01$	**3.24 ± 0.26**	**2.3**	**4**	7	gradient-based	**0.376**	**642,048**
DEff-ARTS + cutout, $\Gamma = 0.02$	**4.42 ± 0.07**	**1.6**	**4**	7	gradient-based	**0.262**	**276,480**
DEff-ARTS + cutout, $\Gamma = 0.04$	16.01 ± 0.41	1.45	4	7	gradient-based	0.242	0

Experimental Setup: We used the same setup as (Zoph and Le, 2017; Liu et al., 2018; Real et al., 2019) where networks were formed by stacking cells together with architecture weight sharing. Normal and reduction cells were derived to control representation sizes with reduction cells placed at 1/3rd and 2/3rd of the depth of the network. Aditionally, hard-coded preprocessing consisting of RELU-Conv-BN operations was applied to the input representations of every cell. The modulation parameter, β was set to a value of 0.27 and kept constant through all experiments. Cells were derived on CIFAR-10 (Hinton, 2007) with 8-cell networks by searching for 50 epochs. The test error results reported for DEff-ARTS were obtained from 20-cell networks and are averages of 3 runs. We include a second metric of MAC operations to measure complexity of the networks, since our compute cost metric didn't consider preprocessing in cells. The search was carried out four times each for the different values of the cost weightage hyperparameter. The set of cells used to report test error results were selected for their compute cost and were all derived from a common random initialisation.

Results: A comparison with state-of-the-art architectures derived through other approaches is shown in Table 3. The trade-off between the test error and complexity can be observed in Figure 2 and 3. For $\Gamma = 0.01$ and 0.02, the networks achieved highly competitive error rates with $1.4\times$ to $2\times$ smaller model sizes and $1.9\times$ to $4.5\times$ lower compute costs. The drop in MACs required was not as large as the compute cost due to the metric not accounting for the different cycle requirements of the CPU operations. For $\Gamma = 0.04$, the network achieved an error rate of 16.01% with a compute cost of 0. This was due to the exclusion of preprocessing blocks from the optimisation which had a fixed contribution to the networks performance. Comparing the search cost, the differentiable approaches were three orders of magnitude faster than the other approaches and DEff-ARTS incurred no extra cost to the search compared to DARTS.

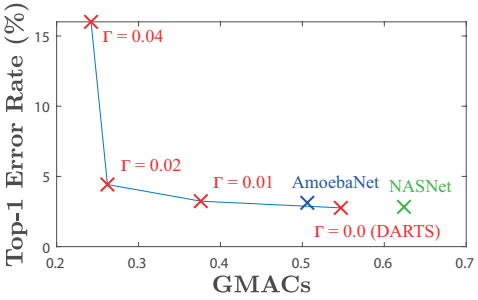


Figure 2: GMACS vs. Test Error

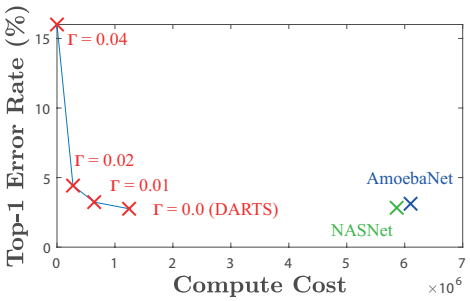


Figure 3: Compute Cost vs. Test Error

4 Conclusion and Discussion

Designing efficient neural network architectures can be quite challenging due to human expert knowledge and effort required in the process. We used gradient based optimisation to search for efficient CNN architectures for target hardware which bypassed the need for human effort in designing novel efficient architectures. We derived multiple architectures with DEff-ARTS that produced highly competitive results with the state-of-the-art on CIFAR-10 classification with significantly lower compute requirements and no need for hardware implementation. Expanding the search space and further improving the metric used to measure computational complexity could lead to discovery of even better performing models.

Acknowledgements

This work was supported by the UK Research and Innovation (UKRI) Centre for Doctoral Training in Machine Intelligence for Nano-electronic Devices and Systems [EP/S024298/1] and the Engineering and Physical Sciences Research Council (EPSRC) International Centre for Spatial Computational Learning [EP/S030069/1]. The authors thank Christopher Culley and Jack Dymond for helpful discussions and also acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton, in the completion of this work.

References

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131.

Gordon, A., Eban, E., Nachum, O., Chen, B., Wu, H., Yang, T.-J., and Choi, E. (2018). Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1586–1595.

Han, S., Mao, H., and Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. (2018). Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800.

Hinton, G. E. (2007). Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

Liang, S. and Zhang, Y. (2020). A simple general approach to balance task difficulty in multi-task learning. *arXiv preprint arXiv:2002.04792*.

Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

Mishra, A. and Marr, D. (2018). Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. In *International Conference on Learning Representations*.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823. IEEE.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. (2019). Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

A Multi-Objective Optimisation

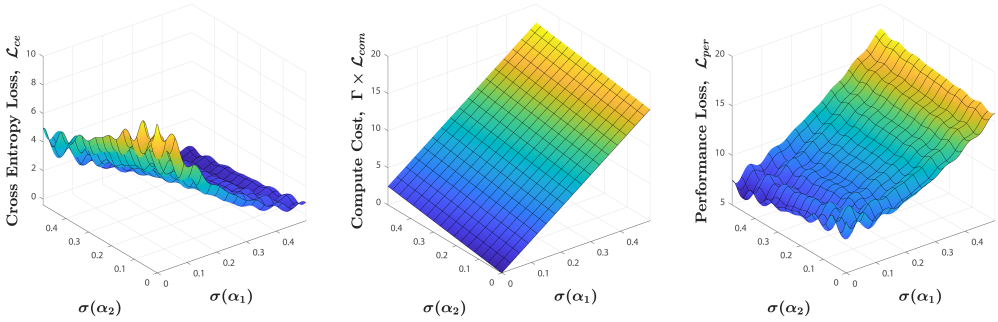
A.1 Limitations of Linear Combination of Sub-Objectives

$$\mathcal{L}_{com} = \Sigma_{i=1}^{p}\Sigma_{j=1}^{q}\sigma(\alpha_{i,j}^{normal}) \times Cost_j + \Sigma_{i=1}^{p}\Sigma_{j=1}^{q}\sigma(\alpha_{i,j}^{reduce}) \times Cost_j$$

For our expression of the compute cost function as given above, the gradient with respect to any particular architecture weight could be computed as shown below. The first limitation we observed was that the back-propagated gradient only depended on the architecture weight itself as shown below.

$$\frac{\partial}{\partial \alpha_{i,j}^{normal}}\mathcal{L}_{com}(w,\alpha) = \exp(\alpha_{i,j}^{normal})\left(1 - \exp(\alpha_{i,j}^{normal})\right) \times Cost_j$$

Secondly, without transforming the compute cost, we observed that dimensions associated with expensive operations overrided other dimensions for cheaper operations as shown in Figure 4b where $\sigma(\alpha_1)$ and $\sigma(\alpha_2)$ are associated with an expensive and cheaper operation respectively. It can be observed that the $\sigma(\alpha_1)$ dimension dominated the second in terms of the gradient and the magnitude. Further, due to the difference in scale of the sub-objectives in Figure 4a and 4b, the features of the cross-entropy loss landscape are lost in the multi-objective optimisation landscape in Figure 4c. In our experiments, the difference in scale was 4 orders of magnitude larger than demonstrated, so the problem was even more pronounced



(a) Cross-Entropy Loss, $\mathcal{L}_{ce}$ (b) $\Gamma \times \mathcal{L}_{com}$, $\Gamma = 1$ (c) Performance Loss, $\mathcal{L}_{per}$

Figure 4: Optimisation Landscape Sketches

A.2 Non-Linear Transformation Analysis

Applying a non-linear transformation to the compute cost, with a small value for the modulation parameter, β brought the different dimensions of the compute cost to a comparable scale as previously shown in Figure 1b. Further, the back-propagated gradients were also controlled as shown below. The extra dependence on the rest of the cell structure that appears modulated the gradients such that they were on a similar scale.

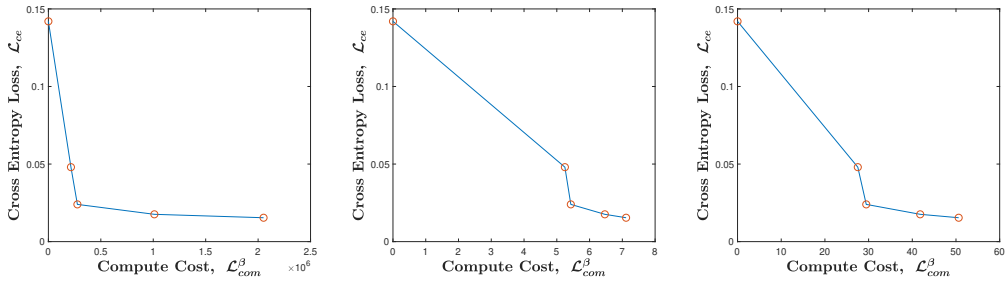
$$\frac{\partial}{\partial \alpha_{i,j}^{normal}}\mathcal{L}_{com}(w,\alpha)^{\beta} = \beta(\sum_{i=1}^{n}\sigma(\alpha_{i,j}^{normal})\exp(\alpha_{i,j}^{normal})\left(1 - \exp(\alpha_{i,j}^{normal})\right) \times Cost_j$$

The effect of the cost weightage parameter, Γ is analytically shown below where it simply scales the gradients to over or underemphasise the importance of the compute cost in the optimisation.

$$\begin{aligned}\frac{\partial}{\partial \alpha_{i,j}^{normal}}\mathcal{L}_{per}(w,\alpha) &= \frac{\partial}{\partial \alpha_{i,j}^{normal}}\left(\mathcal{L}_{per}(w,\alpha) + \Gamma\mathcal{L}_{com}(w,\alpha)^{\beta}\right) \\ &= \frac{\partial}{\partial \alpha_{i,j}^{normal}}\mathcal{L}_{per}(w,\alpha) + \Gamma \times \frac{\partial}{\partial \alpha_{i,j}^{normal}}\mathcal{L}_{com}(w,\alpha)^{\beta}\end{aligned}$$

Another reason for the transformation was that a simple linear combination would assume that the sub-objectives had a linear relationship that could be traded off. This was not true as shown below

in Figure 5a which was obtained by training networks with varying compute costs and noting the cross-entropy loss at the end of training. By exponantiating, the relationship was made sufficiently linear as shown in Figure 5b and 5c, so that the cross-entropy loss could be traded off with the transformed compute cost through the cost weightage hyperparameter, Γ.



(a) $\beta = 1$ (b) $\beta = 0.135$ (c) $\beta = 0.27$

Figure 5: Cross-Entropy Loss, $\mathcal{L}_{ce}$ vs. Compute Cost, $\mathcal{L}_{com}$

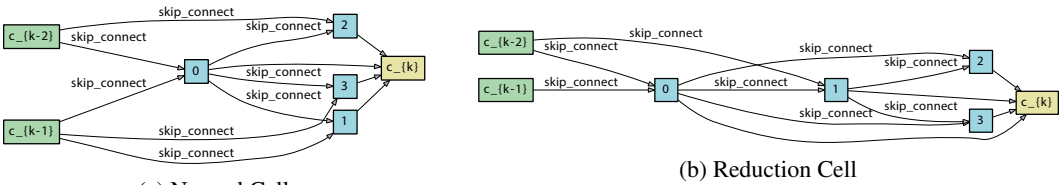
B Candidate Operation CPU Cycles

The cycles required by the considered candidate operations were calculated manually using the expressions below where H, W and C_{in} are the size of the intermediate representations and k, S and C_{out} are the kernel size, stride and output channels of the operation. We assumed intermediate representations to be of fixed sizes and unit strides.

$$
\begin{aligned}
Cost_{sep} = \Big[&\big[\left(k \times k \times H \times W \times C_{in} \div S^2\right) + \left(C_{in} \times H \times W \times C_{out}\right) \big] \times Cost_{mul} \\
&+ \big[\left(k \times k \times H \times W \times C_{in} \div S^2\right) + \left(C_{in} \times H \times W \times C_{out}\right) \big] \times Cost_{add} \Big] \times 2 \\
Cost_{dil} = &\big[\left(k \times k \times H \times W \times C_{in} \div S^2\right) + \left(C_{in} \times H \times W \times C_{out}\right) \big] \times Cost_{mul} \\
&+ \big[\left(k \times k \times H \times W \times C_{in} \div S^2\right) + \left(C_{in} \times H \times W \times C_{out}\right) \big] \times Cost_{add} \\
Cost_{avg} = &\left(k \times k \times H \times W \times C_{in}\right) \times Cost_{mul} + \left(H \times W \times C_{in}\right) \times Cost_{div} \\
Cost_{max} = &\left(k \times k \times H \times W \times C_{in}\right) \times Cost_{comp} \\
Cost_{skip} = &\ 0
\end{aligned}
$$

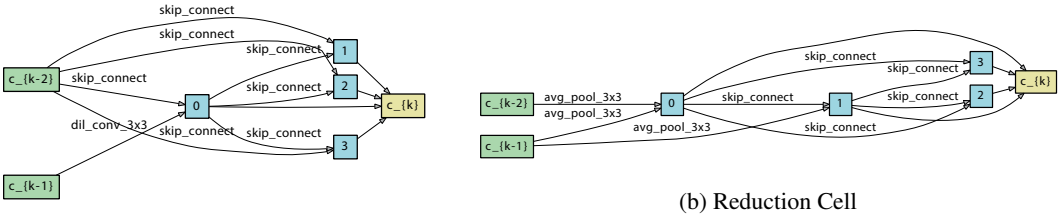
C Derived and Evaluated Cells

In this section, we visualise the cells that were evaluated for test error in Section 3. Cells besides DEff-ARTS were taken from Liu et al. (2018).



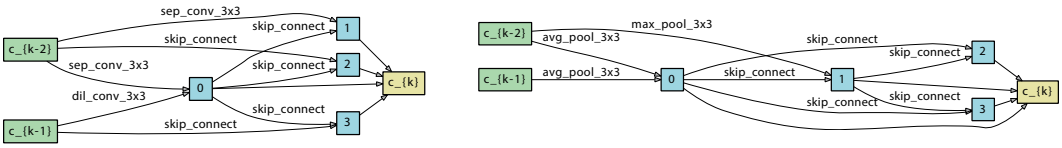
(a) Normal Cell (b) Reduction Cell

Figure 6: Derived Cells, $\Gamma = 0.04$, $\mathcal{L}_{com} = 0$



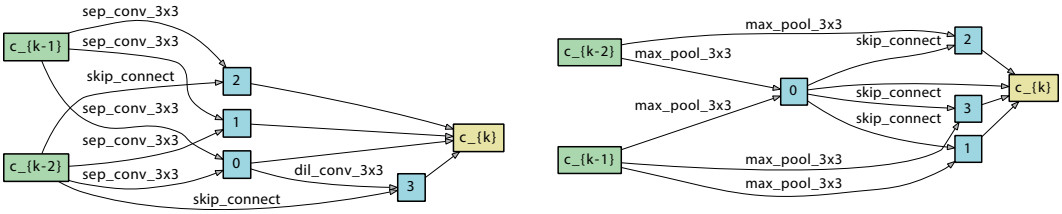
(a) Normal Cell (b) Reduction Cell

Figure 7: Derived Cells, $\Gamma = 0.02$, $\mathcal{L}_{com} = 276,480$



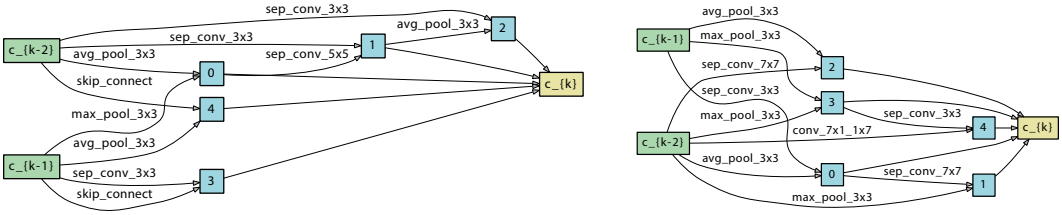
(a) Normal Cell (b) Reduction Cell

Figure 8: Derived Cells, $\Gamma = 0.01$, $\mathcal{L}_{com} = 642,048$



(a) Normal Cell (b) Reduction Cell

Figure 9: DARTS Cells ($\Gamma = 0$), $\mathcal{L}_{com} = 1,244,160$



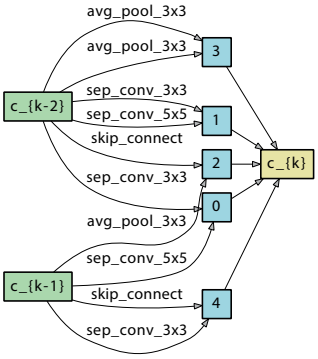
(a) Normal Cells (b) Reduction Cells

Figure 10: AmoebaNet Cells, $\mathcal{L}_{com} = 6,100,992$

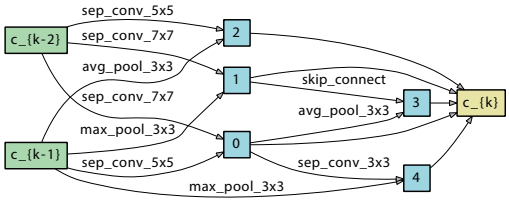
D Experimental Details

D.1 Architecture Search

- Number of Cells: 8
- Cell Step Size: 4
- Epochs: 50
- Batch Size: 64
- Initial Stem Block Output Channels: 16
- Networks Weights Optimiser: SGD with momentum



(a) Normal Cells



(b) Reduction Cells

Figure 11: NASNet Cells, $\mathcal{L}_{com} = 5,861,376$

  - Learning Rate: 0.025 annealed through cosine schedule
  - Momentum: 0.9
  - Weight Decay: 3×10^{-4}
  - Gradient Norm Clipping: 5.0
- Architecture Weights Optimiser: Adam
  - Learning Rate: 3×10^{-4}
  - Momentum β: $(0.5, 0.999)$
  - Weight Decay: 10^{-3}
- Modulation Parameter, $\beta = 0.27$
- Cost Weightage Parameter, $\Gamma = 0.01, 0.02, 0.04$
- $Cost_{zero} = 10^6 - 1$, Liu et al. (2018) declared a zero candidate operation in the search which was not considered when discretising a cell. The given value was used to focus selection pressure on non-zero candidate operations considered in discretisation.

D.2 Architecture Evaluation

Results reported for test error rate were an average of 3 runs.

- Number of Cells: 20
- Epochs: 600
- Batch Size: 96
- Initial Stem Block Output Channels: 36
- Optimiser: SGD with momentum
  - Learning Rate: 0.025 annealed through cosine schedule
  - Momentum: 0.9
  - Weight Decay: 3×10^{-4}
  - Gradient Norm Clipping: 5.0
- Auxiliary Tower Weight: 0.4
- Auxiliary Depth: $2/3$rd of network depth
- Cutout Length: 16
- Drop Path Probability: 0.2

E Data

E.1 Architecture Evaluation

Table 4: Evaluation data across 3 runs for results shown in Table 3

Cost Weightage, Γ	**Run 1**	**Run 2**	**Run 3**
0.01	3.02	3.6	3.09
0.02	4.37	4.52	4.38
0.04	16.34	16.24	15.43

E.2 Compute Cost vs. Cross-Entropy Loss

Table 5: Empirically obtained data used to plot the relationship between the sub-objectives in Fig 5

Cross-Entropy Loss, $\mathcal{L}_{ce}$	0.142	0.048	0.024	0.0176	0.0154
Compute Cost, $\mathcal{L}_{com}$	0	215,040	276,480	1,010,688	2,052,096