# A. Appendix

## A.1. Comparison of different sparsification strategies and impact on pruning

We recap the three different sparsification strategies discussed in Section 4 and Figure 2:

1. vanilla DARTS [32], i.e., no sparsification (neither operations nor inputs)
2. sparsifying the operations only (e.g., as in SNAS [53])
3. sparsifying both operations and inputs (as proposed in our work).

Prior to the meta-learning setting, we evaluated these three strategies on the default single-task classification setting on CIFAR-10. We ran the search phase of DARTS with default hyperparameters (i.e., hyperparameters identical to Liu *et al*. [32]) on CIFAR-10 and evaluated the drop in accuracy after the search phase when going from the one-shot model to the pruned model. The one-shot model is pruned as proposed by Liu *et al*. [32]. The results can be found in Table 3. In the vanilla setting without any sparsification, the accuracy drops significantly, almost to chance level. When sparsifying the operations only, accuracy is much better but still clearly below the original performance. In contrast, with our proposed strategy, there is no significant drop in performance.

| Sparsification strategy | No sparsification (Fig. 2, left) | Sparsify operations only (Fig. 2, middle) | Sparsify operations & inputs (Fig. 2, right) |
|---|---|---|---|
| Accuracy before pruning | $87.9 \pm 0.2\%$ | $84.4 \pm 0.4\%$ | $84.8 \pm 0.3\%$ |
| Accuracy after pruning | $16.7 \pm 2.1\%$ | $60.2 \pm 4.0\%$ | $84.7 \pm 0.4\%$ |

Table 3: Comparing the different annealing strategies discussed in Section 4 and Figure 2: 1) vanilla DARTS (no annealing) 2) annealing the operations only, 3) annealing both operations and inputs (as proposed in our work). Mean $\pm$ SEM, on single-task NAS setting (CIFAR-10, DARTS' default hyperparameters).

## A.2. Detailed experimental setup and hyperparameters

Our implementation is based on the REPTILE [36][2] and DARTS [32][3] code. The data loaders and data splits are adopted from Torchmeta [10][4]. The overall evaluation setup is the same as in REPTILE.

[2] https://github.com/openai/supervised-reptile
[3] https://github.com/khanrc/pt.darts
[4] https://github.com/tristandeleu/pytorch-meta

Hyperparameters are listed in Table 4. The hyperparameters were determined by random search centered around default values from REPTILE on a validation split of the training data.

For the experiments in Section 5.1, all models were trained for 30,000 meta epochs. For METANAS, we did not adapt the architectural parameters for the first 15,000 meta epochs to warm-up the model. Such a warm-up phase is commonly employed as it helps avoiding unstable behaviour in gradient-based NAS [7, 43].

| Hyperparameter | Value |
|---|---|
| batch size | 20 |
| meta batch size | 10 |
| shots during meta training | 15 / 10 |
| task training steps (during meta training) | 5 |
| task training steps (during meta testing) | 50+50[5] |
| task learning rate (weights) | $10^{-3}$ |
| task learning rate (architecture) | $10^{-3}$ / $5 \cdot 10^{-4}$ |
| task optimizer (weights) | Adam |
| task optimizer (architecture) | Adam |
| meta learning rate (weights) | 1.0 |
| meta learning rate (architecture) | 0.6 |
| meta optimizer (weights) | SGD |
| meta optimizer (architecture) | SGD |
| weight decay ( weights) | 0.0 |
| weight decay (architecture) | $10^{-3}$ |

Table 4: Listing of hyperparameters for METANAS for the experiments of Section 5.1. Hyperparameters are the same across n-shot, k-way setting. Hyperparameters are the same for MiniImagenet and Omniglot except for rows with two values separated by a slash. In this case, the first value denotes the value for MiniImagenet while the latter one denotes the value for Omniglot.

For the experiment in Section 5.2, we make the following changes in contrast to Section 5.1: we meta-train for 100,000 meta epochs instead of 30,000 and increase the weight decay on the weights from 0.0 to $10^{-4}$ and use DropPath [58] with probability 0.2. We increase the number of channels per layer to 96 from 14 (30k parameter models) / 28 (100k parameter models) and use 5 cells instead of 4, whereas again the second and forth cell are reduction cells while all others are normal cells.

## A.3. Motivation of meta-learning algorithm $\Psi$

In Section 3.2, we proposed the two meta-learning updates

[5] By 50+50 we mean that for the first 50 steps, both the weights and architecture are adapted while for the later 50 steps only weights are adapted.

$$\begin{pmatrix} w_{meta}^{i+1} \\ \alpha_{meta}^{i+1} \end{pmatrix} = \Psi^{MAML}(\alpha_{meta}^{i}, w_{meta}^{i}, p^{train}, \Phi^{k})$$
$$= \begin{pmatrix} w_{meta}^{i} - \lambda_{meta} \nabla_{w} \mathcal{L}_{meta}(w_{meta}^{i}, \alpha_{meta}^{i}, p^{train}, \Phi^{k}) \\ \alpha_{meta}^{i} - \xi_{meta} \nabla_{\alpha} \mathcal{L}_{meta}(w_{meta}^{i}, \alpha_{meta}^{i}, p^{train}, \Phi^{k}) \end{pmatrix} \tag{8}$$

and

$$\begin{pmatrix} w_{meta}^{i+1} \\ \alpha_{meta}^{i+1} \end{pmatrix} = \Psi^{REPTILE}(\alpha_{meta}^{i}, w_{meta}^{i}, p^{train}, \Phi^{k})$$
$$= \begin{pmatrix} w_{meta}^{i} + \lambda_{meta} \sum_{\mathcal{T}_i} (w_{\mathcal{T}_i}^{*} - w_{meta}^{i}) \\ \alpha_{meta}^{i} + \xi_{meta} \sum_{\mathcal{T}_i} (a_{\mathcal{T}_i}^{*} - \alpha_{meta}^{i}) \end{pmatrix}. \tag{9}$$

Equation (8) extends the MAML update

$$w_{meta}^{i+1} = w_{meta}^{i} - \lambda_{meta} \nabla_{w} \mathcal{L}_{meta}(w_{meta}^{i}, p^{train}, \Phi^{k}),$$

which is simply one step of SGD on the meta-objective (Equation 2), while Equation (9) extends the REPTILE update

$$w_{meta}^{i+1} = w_{meta}^{i} + \lambda_{meta} \sum_{\mathcal{T}_i} (w_{\mathcal{T}_i}^{*} - w_{meta}^{i}),$$

which was shown to maximize the inner product between gradients of different batches for the same task, resulting in improved generalization[36]. Equations (8) and (9) constitute a simple heuristic that perform the same updates also on the architectural parameters.

### A.4. Additional plots

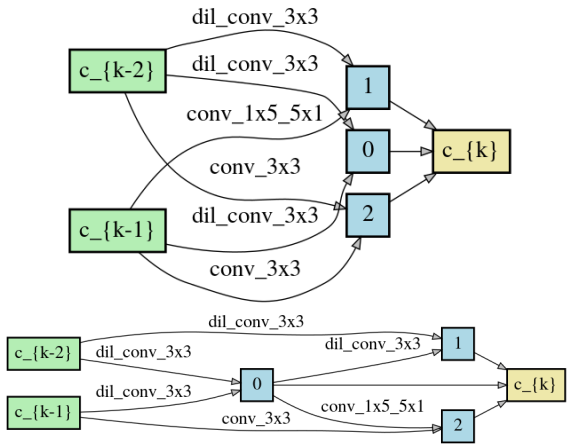


Figure 5: Two other commonly used reduction cells.