Understanding deep meta-learning

Huisman, M.

Chapter 6

Subspace Adaptation Prior for Few-Shot Learning

Chapter overview

Gradient-based meta-learning techniques aim to distill useful prior knowledge from a set of training tasks such that new tasks can be learned more efficiently with gradient descent. While these methods have achieved successes in various scenarios, they commonly adapt *all* parameters of trainable layers when learning new tasks. This neglects potentially more efficient learning strategies for a given task distribution and may be susceptible to overfitting, especially in few-shot learning where tasks must be learned from a limited number of examples. To address these issues, we propose *Subspace Adaptation Prior* (SAP)[1], a novel gradient-based meta-learning algorithm that jointly learns good initialization parameters (prior knowledge) and layer-wise *parameter subspaces* in the form of operation subsets that should be adaptable. In this way, SAP can learn which operation subsets to adjust with gradient descent based on the underlying task distribution, simultaneously decreasing the risk of overfitting when learning new tasks. We demonstrate that this ability is helpful as SAP yields superior or competitive performance in few-shot image classification settings (gains between 0.1% and 3.9% in accuracy). Analysis of the learned subspaces demonstrates that low-dimensional operations often yield high activation strengths, indicating that they may be important for achieving good few-shot learning performance. For reproducibility purposes, we publish all our research code publicly.

6.1 Introduction

In this chapter, we revisit the popular deep meta-learning technique model-agnostic meta-learning (MAML) (Finn et al., 2017) that learns a prior in the form of the initialization parameters of the neural network. Learning a new task is then done by performing gradient descent starting from this meta-learned initialization. This approach, which is also widely used by techniques that are based on MAML (Lee and Choi, 2018; Flennerhag et al., 2020; Park and Oliva, 2019b; Yoon et al., 2018;

[1]This chapter is based on the following research article: *Huisman, M., Plaat, A., & van Rijn, J. N. (2023). Subspace Adaptation Prior for Few-Shot Learning. Machine Learning. Springer.*

Nichol et al., 2018), updates all of the parameters of every *trainable* layer with gradient descent when learning new tasks, which may be suboptimal for a given task distribution and may lead to overfitting since there are more degrees of freedom to fit the noise in the data. Especially in few-shot learning, where tasks are noisy due to the fact that only limited examples are available, these issues could hinder performance.

To address these issues and investigate the research question of whether the few-shot learning performance of deep neural networks can be improved by meta-learning which subsets of parameters to adjust, we propose a new gradient-based meta-learning technique called *Subspace Adaptation Prior* (SAP) that jointly learns good initialization parameters as well as layer-wise subspaces in which to perform gradient descent when learning new tasks. More specifically, SAP is given access to a candidate pool of *operations* for every layer that transforms the hidden representations, and it learns which of these subsets to adjust in order to learn new tasks quickly, similar to DARTS (Liu et al., 2019). Here, every operation corresponds to a parameter subspace. Note that this method serves as a form of regularization and allows SAP to find more efficient adaptation strategies than adjusting all parameters of trainable layers. In addition, it utilizes implicit gradient modulation to warp (Lee and Choi, 2018; Flennerhag et al., 2020) these subspaces per layer such that gradient descent can quickly adapt to new tasks, if they share a common structure.

We empirically demonstrate that SAP is able to find efficient parameter subspaces, or operation subsets, that match the underlying task structure in simple synthetic settings and yield good few-shot learning. Moreover, SAP outperforms gradient-based meta-learning techniques—that do not have the ability to learn in which structured subspaces to perform gradient descent—on few-shot sine wave regression and performs on-par or favorably in various few-shot image classification settings. In short, the contributions of this chapter are the following:

- We propose SAP, a new meta-learning algorithm for few-shot learning that jointly learns good initialization parameters and parameter subspaces in the form of operation subsets in which to perform gradient descent.
- We demonstrate the advantage of learning parameter subspaces as SAP outperforms existing methods by at least 18% on few-shot sine wave regression and yields competitive or superior performance on popular few-shot image classification benchmarks (improvements in classification accuracy scores range from 0.1% to 3.9%).
- We investigate the learned layer-wise parameter subspaces on synthetic few-shot sine wave regression and image classification problems and find that small subsets of adjustable parameters (simple parameter subspaces), including feature transformations such as element-wise scaling and shifting are assigned large weights, suggesting that they play an important role in achieving good performance with SAP.
- For reproducibility and verifyability purposes, we make all our research code publicly available.[2]

6.2 Related work

We review related work on optimization-based meta-learning, neural architecture search and gradient modulation.

[2]See: `https://github.com/mikehuisman/subspace-adaptation-prior`

Optimization-based meta-learning Our proposed technique belongs to the category of optimization-based meta-learning (Vinyals, 2017) (also see Chapter 2), which employs optimization methods to learn new tasks (Yoon et al., 2018; Bertinetto et al., 2019; Lee et al., 2019). These methods aim to meta-learn good settings for various hyperparameters, such as the initialization parameters, such that new tasks can be learned quickly using optimization methods. These methods vary from regular stochastic gradient descent, as used in MAML (Finn et al., 2017) and Reptile (Nichol et al., 2018), to meta-learned procedures where a network updates the weights of a base-learner (Ravi and Larochelle, 2017; Andrychowicz et al., 2016; Li et al., 2017; Rusu et al., 2019; Li and Malik, 2018) (also see Chapter 3). SAP aims to learn good initialization parameters such that new tasks can be learned quickly with regular gradient descent, similar to MAML.

This is a form of transfer learning (Taylor and Stone, 2009; Pan and Yang, 2009) where we transfer knowledge—in this case the initialization parameters—obtained on a set of source tasks to a new target task that we are confronted with. The idea is also related to the idea of domain adaptation (DA) (Daumé III, 2009; Farahani et al., 2021), although in DA it is often assumed that we have a single task but two different data distributions (a source distribution and a target distribution). Note that in deep meta-learning (Hospedales et al., 2021) (also see Chapter 2), we have set of various different training tasks and aim to transfer knowledge to a new target task, different from the ones seen at training time.

Neural architecture search (NAS) for meta-learning The techniques mentioned above assume a pre-specified network architecture. Recently, there has been some work on combining meta-learning with neural architecture search, where the architecture can also be learned. Kim et al. (2018) performs meta-learning as a subroutine to NAS, meaning that meta-training is performed for every candidate architecture, which can be computationally expensive. This problem can be overcome by combining gradient-based meta-learning with gradient-based neural architecture search such that the architecture and initialization parameters can be optimized jointly instead of separately. A popular gradient-based meta-learning algorithm is DARTS (Liu et al., 2019) which starts with a candidate pool of operations (as in SAP) and learns which of them to use, thereby learning an appropriate architecture. Learning which subspaces or subsets of operations to use per layer, as done in SAP, can be seen as applying DARTS over the candidate operation sets. A difference between DARTS and SAP is that we fix the base-learner parameters when adapting to new tasks, which can then serve to warp, or transform, the gradients such that gradient descent can quickly move to a good solution for new tasks (see below). Moreover, SAP updates the initialization parameters of all meta-trainable parameters with a MAML-like update (to maximize post-adaptation performance), while DARTS uses a Reptile-like update (to maximize multi-step performance). We describe DARTS in full detail in Section 6.3.

Lian et al. (2019) were the first to combine DARTS (Liu et al., 2019) with gradient-based meta-learning in order to learn a base-learner architecture that can be quickly adapted to new tasks. They perform hard-pruning, which requires re-running the meta-training phase for every new task, which is computationally expensive. In parallel to this work, Elsken et al. (2020) proposed a similar approach (MetaNAS) that does not perform hard-pruning and thus side-steps these expensive re-running procedures. In contrast to these works, which learn and adapt the base-learner network architecture as well as all of the parameters to every new task, SAP assumes a fixed base-learner architecture as a starting point and aims to learn a set of operations that are inserted per layer (see Section 6.4) that are responsible for quickly adapting to new tasks. In SAP, the architecture of the network is frozen at test time, in contrast to, for example, the architecture of the networks learned by MetaNAS (Elsken et al., 2020).

Gradient modulation in gradient-based meta-learning Recent works that build upon MAML have shown that gradient modulation can improve the generalization of optimization-based techniques (Sun et al., 2019). Explicit gradient modulation techniques directly transform the gradient updates when learning new tasks (Simon et al., 2020) through, for example, diagonal matrix multiplication (Li et al., 2017), or block-diagonal preconditioning (Park and Oliva, 2019b). Implicit gradient modulation techniques do not directly operate on the gradients but rely on indirect transformations. CAVIA (Zintgraf et al., 2019) separates shared parameters from context parameters. The latter serve as additional inputs to one or more layers of the neural network and are adjusted when learning a new task, whilst the shared parameters are kept fixed. Other examples of implicit gradient modulation methods are T-Net (Lee and Choi, 2018) and Warp-MAML (Flennerhag et al., 2020). SAP also performs implicit gradient modulation in a similar fashion to these two techniques.

T-Net inserts linear projection transformations directly after every matrix multiplication in the base-learner. The weights of these transformations are frozen when learning new tasks, and only the base-learner weights are adjusted. The goal is to meta-learn good initialization parameters of the base-learner weights as well as the transformation weights, such that new tasks can be learned more quickly. These transformation layers serve to implicitly modulate the gradients of the base-learner parameters so that gradient descent can quickly move to good solutions for new tasks. MT-Net is an extension to T-Net, which also learns to mask certain features, preventing them from being adapted when learning new tasks. We also investigated whether feature masking was useful for SAP, but found that it decreased performance. Warp-MAML is a generalization of T-Net as it does not require that the inserted transformation layers are linear, that is, the theoretical framework allows these transformation layers to be non-linear and consist of multiple layers (arbitrary neural networks).

Both T-Net and Warp-MAML adjust all parameters of trainable layers, as is common in gradient-based meta-learning. However, this may be suboptimal for a given task distribution and lead to overfitting due to the large degree of freedom to fit the noise in the data. MT-Net, on the other hand, freezes certain features, which, in turn, also requires certain weights to be frozen but this is rather inflexible as that does not allow us to perform simple operations such as element-wise scaling of all features, which may be helpful for a given task distribution. To overcome these issues, we propose SAP, which learns per trainable layer which operations from a pre-defined candidate pool to use and adapt when learning new tasks, instead of resorting to regular matrix multiplications in which all weights are adjusted when learning new tasks (as done by other methods). While the expressivity of SAP is equivalent to T-Net and Warp-MAML (when using linear warp layers), the candidate pool of operations allows SAP to learn which operations are important for the given task distribution, thereby structuring the weight updates.

SAP is similar to T-Net and Warp-MAML in the sense that the linear base layers $\mathbf{W}^\ell$ (see Section 6.4) of the network in SAP can be seen as the warp layers or transformation layers that are used in T-Net and Warp-MAML, which act as implicit preconditioning layers that warp the loss surface to aid gradient descent in finding a good solution. Due to the similarities between T-Net, Warp-MAML, and SAP, they serve as excellent baselines to investigate whether the ability of SAP to learn which operation subsets to adapt when learning new tasks is helpful for few-shot learning. Concurrently to our work, Jiang et al. (2022) have proposed a subspace meta-learning algorithm. Whilst the title is similar, they explicitly meta-learn the bases for K subspaces. Then, when learning a new task, they aim to find linear combinations of the basis vectors of each of the subspaces that give rise to the best parameters for the given task in the subspaces. The subset containing the parameters with the lowest training loss is then used to obtain predictions for the query/test set. Note that their work is

different in that we do not learn basis vectors for different subspaces, but instead insert candidate operations that act to transform intermediate representations in the base-learner network to allow for faster learning and modulating the gradients.

6.3 Background on DARTS

In this section, we briefly describe DARTS (Liu et al., 2019), which is a gradient-based neural architecture search method that we build upon in this chapter. The goal of DARTS is to find a suitable neural architecture for a given problem. To do this, DARTS assumes a set of candidate operations that can be used to transform an input into an output. These candidate operations form a weighted graph as shown in Figure 6.1. In the figure, every node $o_i(\mathbf{x})$ corresponds to a candidate operation and the weights of the edges correspond to the activation strengths of the different operations. These weights are initially unknown and DARTS aims to learn them jointly with the initial parameters of every operation. The output of the layer in the figure is given by

$$\mathcal{O}(\mathbf{x}) = \sum_{i=1}^{n} w_i o_i(\mathbf{x}), \tag{6.1}$$

where w_i is the weight of operation i and $\sum_{i=1}^{n} w_i = 1$ (e.g., by using a softmax). For our purposes, we only consider DARTS for searching over operations for a single layer, but it can be used for multi-layer architectures as well.

In addition to learning the weights w_i, DARTS simultaneously learns good parameters for every operation o_i. We denote the group of all activation weights as $\lambda = \{w_1, w_2, \ldots, w_n\}$ and all operation parameters as θ. DARTS adopts a method similar to MAML for learning λ and θ. That is, given a training task $\mathcal{T}_j = (D^{tr}_{\mathcal{T}_j}, D^{te}_{\mathcal{T}_j})$, DARTS performs a gradient update step on the operation weights θ as follows

$$\theta'_j = \theta - \alpha \nabla_{\theta,\lambda} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\theta, \lambda). \tag{6.2}$$

Note that this is similar to Equation 2.20 in Chapter 2 with the exception that we now have activation strength parameters λ, which are kept constant during this inner-loop adaptation step. After updating the operation parameters θ, DARTS computes the loss of the new model on the query set, i.e., $\mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta'_j, \lambda)$ and updates the activation strengths using gradient descent on this loss

$$\lambda = \lambda - \beta \nabla_{\lambda} \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\theta'_j, \lambda). \tag{6.3}$$

Similarly to MAML (see Chapter 2 and Chapter 3), this update also contains second-order gradients, but first-order approximations can be made. In DARTS, the weights of the operations θ are simply updated to their new values, that is, $\theta = \theta'_j$, i.e., after every task in the meta-train set, we update the initialization parameters θ to the parameters that were obtained after training on task $\mathcal{T}_j$.

6.4 Subspace Adaptation Prior

In this section, we motivate and present our proposed technique called *Subspace Adaptation Prior* (SAP).

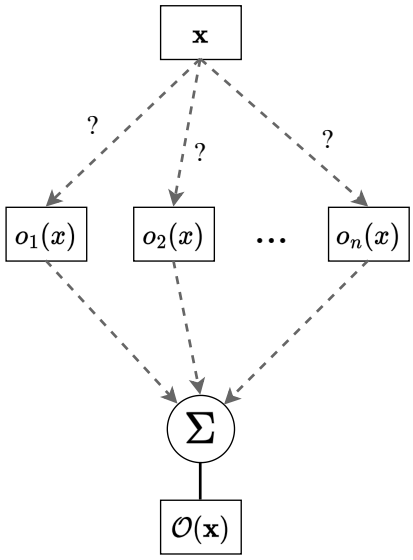
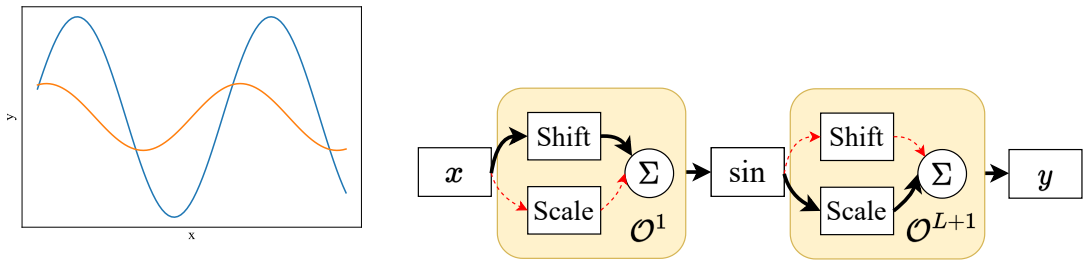


Figure 6.1: Intuitive visualization of DARTS. It is given a set of candidate operations $\mathcal{O}$ and aims to learn the weights of the edges (indicated as ?), corresponding to the strengths of the different operations $o_i(\mathbf{x})$. The output of the weighted graph is a convex combination of the different operations $\mathcal{O}(\mathbf{x}) = \sum_{i=1}^{n} w_i o_i(\mathbf{x})$.

6.4.1 Intuition and operations

Our method (SAP) builds on MAML as we also aim to learn good initialization parameters such that good performance can be achieved after a small number of gradient updates. However, MAML adapts all of its network parameters when presented with a new task, which may be suboptimal for the given task distribution and lead to overfitting. Our method, SAP, is given a pool of candidate operations per layer (described below) and it learns per layer which subset of operations should be adjusted to adapt to a new task. Since all of the operations that SAP can choose from per layer are subsumed in terms of expressivity by a full-rank matrix multiplication (or convolution in the case of image data), this can be understood as learning in which *parameter subspaces* to perform gradient descent so that new tasks can be learned more efficiently.

This is a form of regularization and can help the network to exploit structures in problems. For example, take the distribution of tasks $\mathcal{T}_j$ corresponding to different sine waves $g_j(x) = A_j \cdot \sin(x - p_j)$, where A_j is the amplitude and p_j the phase. There exists a common structure amongst these tasks: a given sine wave can be transformed into any other sine wave by simply shifting the input and scaling the output. This has been visualized in Figure 6.2. Techniques that adapt all parameters may overwrite the sine function and overfit to the noise, whereas theoretically, SAP could learn to keep these parameters fixed and that shifting the input and scaling the output are the most important operations and consequently, that gradient descent should be performed in the parameter subspaces corresponding to these operations. Sine waves form a simplistic example to demonstrate the idea of SAP, however, we note that also for image classification tasks, simple operations such as scaling and shifting feature maps can be useful too (Sun et al., 2019; Perez et al., 2018; Requeima et al., 2019). SAP can discover such underlying structures and use them to enhance its few-shot learning abilities.



(a) Sine wave tasks (b) Learned subspaces

Figure 6.2: SAP can learn the activation strengths of candidate operations $\mathcal{O}^\ell$ (corresponding to parameter subspaces) that match the problem structure. Suppose we are given a sine wave task distribution, where every task $\mathcal{T}_j$ is a sine wave $g_j(x) = A_j \cdot \sin(x - p_j)$, where p_j is the phase and A_j the amplitude. Instead of adapting all parameters of the network on a new task, SAP can learn to keep the sine network parameters (sin) frozen and that the input shift (shift in $\mathcal{O}^1$) and output scale (scale in $\mathcal{O}^{L+1}$) are the most important operations to adjust (bold and dark-colored arrows), matching the role of the phase and amplitude, respectively.

Candidate operations The candidate operations that SAP uses are specified by hand before meta-training. In order to preserve the original expressivity of the base-learner network, the operations are elementary linear algebra operations that are subsumed by full-rank matrix multiplication.

Table 6.1 displays all the operations that we use for both fully-connected and convolutional layers. The MTL scale operation was proposed by Sun et al. (2019). By construction, we require that the output of an operation set must have the same dimensionality as the input. Recall that in the case of fully-connected layers, all candidate operations can be expressed by a single matrix multiplication where only a subset of the entries is used. For example, an element-wise scale can be performed by multiplying the input with a diagonal matrix where the diagonal entries correspond to the element-wise scalars, and the non-diagonal entries are zero. In this way, every candidate operation occupies a *part* of the full operation set matrix. This also holds for convolutions, which can be seen as a stack of matrices.

We also include singular value (SVD) decomposition operations, where three v-rank matrices $A = U\Sigma V^T$ are multiplied to obtain a transformation matrix $A \in \gtrdot \times \ltimes$ with the same dimensionality as a full-rank transform $T \in \mathbb{R}^{m \times n}$ (although with a lower rank). Here, $U \in \mathbb{R}^{m \times v}$, $\Sigma \in \mathbb{R}^{v \times v}$, and $V^T \in \mathbb{R}^{v \times n}$. The obtained transformation A is then applied to the input.

Below, we describe how these operations are interleaved with the base-learner network and how SAP learns which subsets to adjust.

6.4.2 The algorithm

Architecture Let f_θ be a neural network with parameters θ, where the output, or prediction, is given by

$$f_\theta(\mathbf{x}) = \mathbf{W}^L \sigma(\dots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x}))). \tag{6.4}$$

Fully-connected		Convolutional	
Operation	Dimensionality	Operation	Dimensionality
Identity	N.A.	Identity	N.A.
Matrix multiplication	$d \times d$	Convolution	$C \times C \times k \times k$
SVD-matrix multiplication	$d \times v$	SVD convolution	$C \times C \times k \times v$
Element-wise scale	d	1x1 convolution	$C \times C$
Scalar scale	1	MTL scale	$C \times C$
Vector shift	d	Channel-wise scale	C
Scalar shift	1	Channel-wise shift	C
		Scalar shift	1

Table 6.1: The candidate operations for fully-connected and convolutional network layers and the corresponding dimensionality of the subspace in which gradient will be performed. Here, d is the dimensionality of the input in the case of a fully-connected layer and C is the number of input and output dimensions of candidate operations in the case of convolutional layers. k is the kernel size of convolutions and $v < k$ is a variable dimension for SVD matrices.

Here, L is the number of layers of the network, σ is a non-linear activation function, and $\mathbf{W}^\ell$ is the weight matrix for layer $\ell \in \{1, 2, \ldots, L\}$ (which can also include the bias by concatenating a 1 at the top of the input vector). Note that $\theta = \{\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^L\}$ is the set of all base-learner weight matrix parameters. In SAP, we insert sets of candidate operations $\mathcal{O}^\ell = \{o_1^\ell, \ldots, o_{n_\ell}^\ell\}$ before the application of weight matrices $\mathbf{W}^\ell$ and after computing the final output, as shown in Figure 6.3. Here, n_ℓ is the number of operations in the candidate set $\mathcal{O}^\ell$ in layer ℓ. Each of these operations $o_i^\ell \in \mathcal{O}^\ell$ act on the input, giving rise to partial outputs $o_i^\ell(\mathbf{z}^\ell)$ of the same dimensionality of the inputs, where $\mathbf{z}^\ell$ is the input to the ℓ-th operation layer. The final output of applying the candidate operations is a convex combination of the partial outputs, that is,

$$\mathcal{O}^\ell \mathbf{z}^\ell = \sum_{i=1}^{n_\ell} w_i^\ell o_i^\ell(\mathbf{z}^\ell), \tag{6.5}$$

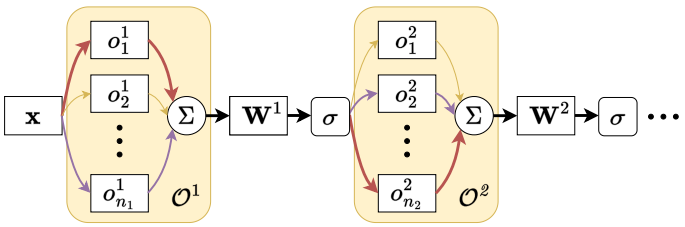


Figure 6.3: A diagram of a feed-forward pass in SAP. Sets of operations $\mathcal{O}^\ell$ are interleaved with base-learner weights $\mathbf{W}^\ell$. The operation sets perform a convex combination of a number of operations $\{o_1^\ell, \ldots, o_{n_\ell}^\ell\}$. SAP learns the strengths of each of the candidate operations and thereby learns in which parameter subspaces gradient descent can effectively adapt the network to learn new tasks. The operation strengths and the weight matrices $\mathbf{W}^\ell$ are frozen when adapting to new tasks. Only the operation parameters are adjusted at test time.

where $\mathbf{z}^1 = \mathbf{x}$ and w_i^ℓ is the activation strength of operation o_i^ℓ. We require that $\sum_{i=1}^{n_\ell} w_i^\ell = 1$ and $0 \leq w_i \leq 1$. Learning these activation strengths can be seen as neural architecture search. Thus, the output of the neural network in SAP is given by

$$f_{\mathbf{\Theta}}(\mathbf{x}) = \mathcal{O}^{L+1}\mathbf{W}^L\mathcal{O}^L\sigma(\dots\sigma(\mathbf{W}^2\mathcal{O}^2\sigma(\mathbf{W}^1\mathcal{O}^1\mathbf{x}))), \tag{6.6}$$

where $\mathbf{\Theta} = \{\theta, \phi, \lambda\}$ is the set of the initial hyperparameter values for the base-learner weights (θ), the operation weights (ϕ), and the activation weights (λ). Note that $\phi = \{\mathcal{O}^1, \mathcal{O}^2, \dots, \mathcal{O}^{L+1}\}$ are the parameters corresponding to the operations in all layers (see Section 6.4.1), and λ is the set containing all w_i^ℓ for all layers $\ell \in \{1, 2, \dots, L+1\}$.

Importantly, each of these candidate operations o_i^ℓ are *subsumed* or equivalent in terms of expressivity with full-rank matrix multiplication. For example, candidate operations can include element-wise shifting or multiplication of the input by a fixed scalar or by a vector, which can also be done by weight matrix multiplication. Since the application of a set of operations $\mathcal{O}^\ell$ of such expressivity can be seen as a single matrix multiplication (hence the suggestive notation), *the expressivity of an SAP network is equivalent to that of the original network.* To see this, note that the application of two weight matrices to an input can be written as the application of a single weight matrix to the input $\mathbf{x}$, that is, $\mathbf{W}(\mathcal{O}\mathbf{x}) = (\mathbf{W}\mathcal{O})\mathbf{x} = \mathbf{W}'\mathbf{x}$, where $\mathbf{W}$ and $\mathcal{O}$ are weight matrices, and $\mathbf{W}' = \mathbf{W}\mathcal{O}$. For the sake of another example, suppose that we have a set of two operations in $\mathcal{O}$: scalar multiplication $s \cdot \mathbf{z}$ and matrix multiplication $\mathbf{M}\mathbf{z}$ (preserving the dimensionality of $\mathbf{z}$). Furthermore, suppose that the two operations are applied with activation strengths w_1 and w_2, granting us the output $\mathbf{z}' = w_1 s \cdot \mathbf{z} + w_2\mathbf{M}\mathbf{z}$. We can rewrite this as $\mathbf{z}' = w_1 sI\mathbf{z} + w_2\mathbf{M}\mathbf{z} = (w_1 sI + w_2\mathbf{M})\mathbf{z} = \mathcal{O}\mathbf{z}$, where $\mathcal{O} = (w_1 sI + w_2\mathbf{M})$ and I is the identity matrix. For a more intuitive example, suppose that a base-layer is a fully-connected layer, and we add a fully-connected operation to alter the resulting representation, maintaining the original dimensionality. The composition of the two fully-connected layers is effectively linear and equally expressive as a single fully-connected layer. **Thus, introducing the operations used by SAP does not alter the expressivity of the original base-learner network.**

Crucially, this insight that we can write the weighted combination of different operations as a single weight matrix multiplication $\mathcal{O}\mathbf{x}$, where $\mathcal{O}$ is a weighted combination of different structured matrices, reveals that SAP effectively learns what subset of parameters of this weight matrix $\mathcal{O}$ and thus of $\mathbf{W}\mathcal{O}$ to adjust by learning the activation strengths λ. In this work, we use the expressions "learning which subsets of parameters to adjust" and "learning in what subspaces to perform gradient descent" synonymously.

Meta-learning The activation strengths w_i^ℓ are meta-learned by SAP in addition to the initialization parameters of the operations $\mathcal{O}^\ell$ and the base-learner weights $\mathbf{W}^\ell$. Note that learning the w_i^ℓ corresponds to learning in which parameter subspaces gradient descent is performed when learning new tasks, which can be done through the layer-wise application of the gradient-based neural architecture search technique DARTS (Liu et al., 2019). Let θ denote the initial parameters of the weight matrices $\mathbf{W}^\ell$, ϕ the parameters of all candidate operations $\mathcal{O}^\ell$, and λ the activation strengths w_i^ℓ of all individual candidate operation. Recall that $\mathbf{\Theta} = \{\theta, \phi, \lambda\}$.

When presented with a new task $\mathcal{T}_j$, the candidate operation activation strengths λ and the base-learner parameters θ are frozen, and only the candidate operation parameters ϕ are updated using

gradient descent for T steps

$$\phi_j^{(t+1)} \leftarrow \phi^{(t)} - \alpha \nabla_{\phi^{(t)}} \mathcal{L}_{\mathcal{T}_j}(\theta, \phi^{(t)}, \lambda), \tag{6.7}$$

where $\phi^{(0)}$ is initialized with ϕ. At the meta-level, the goal is to find good initial parameter settings for all involved parameters such that the task-specific performance is maximized. Thus, we wish to find

$$\underset{\boldsymbol{\Theta}=\{\theta,\phi,\lambda\}}{\arg\min} \underset{\mathcal{T}_j \backsim p(\mathcal{T})}{\mathbb{E}} \mathcal{L}_{\mathcal{T}_j}(\theta, \phi_j^{(T)}, \lambda), \tag{6.8}$$

where $\phi_j^{(T)}$ denotes the task-specific parameters obtained through one or more gradient update steps on task $\mathcal{T}_j$. In other words, we wish to find good initial values for the parameters θ, ϕ, and λ such that new tasks can be learned quickly by updating the operation parameters ϕ. This meta-objective can also be optimized through gradient descent by updating

$$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \beta \nabla_{\boldsymbol{\Theta}} \sum_{\mathcal{T}_j \in \mathcal{B}} \mathcal{L}_{\mathcal{T}_j}(\theta, \phi_j^{(T)}, \lambda). \tag{6.9}$$

The full algorithm for application to few-shot learning is shown in Algorithm 12. At the start (line 1), we initialize the parameters of the base-learner θ randomly. The candidate operation parameters ϕ are initialized to leave the input unaffected (for example, scalars are initialized to 1 and biases to 0). The layer-wise activation strengths w_i^ℓ of the candidate operations are initialized to the uniform distribution. After this initialization, we repeat the following steps until a stopping criterion is met, such as having sampled a certain number of task batches, or observing decreasing performance on held-out validation tasks. We randomly sample batches of tasks (line 3), initialize the task-specific parameters $\phi^{(0)} = \phi$, and make T gradient update steps on the support set of every task (lines 6–8), and perform meta-updates to the initialization parameters $\boldsymbol{\Theta}$ (line 11) using the query sets of the tasks. Note that the meta-update requires the computation of second-order gradients as we have to compute the gradient of the inner-level gradients. The complexity of this is quadratic in the number of parameters, but can be avoided by using the first-order assumption $\nabla_\phi \phi_j^{(T)} = I$.

Algorithm 12 Subspace Adaptation Prior (SAP)

Require: $p(\mathcal{T})$
Require: α, β
1: initialize θ, ϕ, λ
2: **while** not converged **do**
3: sample batch of tasks $\mathcal{B} = \{\mathcal{T}_j = (D_{\mathcal{T}_j}^{tr}, D_{\mathcal{T}_j}^{te}) \backsim p(\mathcal{T})\}_{j=1}^{M}$
4: **for** task $\mathcal{T}_j = (D_{\mathcal{T}_j}^{tr}, D_{\mathcal{T}_j}^{te}) \in \mathcal{B}$ **do**
5: initialize task-specific parameters $\phi_j^{(0)} = \phi$
6: **for** $t = 0, \ldots, T-1$ **do**
7: compute gradient update $\phi_j^{(t+1)}$ using Equation 6.7 on $D_{\mathcal{T}_j}^{tr}$
8: **end for**
9: **end for**
10: update initial parameters $\boldsymbol{\Theta} = \{\theta, \phi, \lambda\}$ using Equation 6.9
11: **end while**

Pruning The scores w_i^ℓ represent the *activation strengths* of the different candidate operations/subspaces, and can also be used for pruning the operations, for example, in a layer-wise or regular top-K fashion. By default, we do not hard-prune operations and maintain a convex combination of different candidate operations unless explicitly mentioned otherwise. Note that we cannot simply drop low activation strength operations from the network as that changes the composite features and layerwise activation statistics. Hard-pruning requires re-training the network with only the selected (non-pruned) subspaces/operations.

6.4.3 Analysis

One may wonder what the role is of inserting operation sets $\mathcal{O}^\ell$ in the base-learner network since they have the same expressivity as weight matrices. In other words, why do we have two consecutive matrix multiplications $\mathbf{W}^\ell\mathcal{O}^\ell\mathbf{x}$ if that is equivalent to having one matrix multiplication $\mathcal{U}\mathbf{x}$, where $\mathbf{U} = \mathbf{W}^\ell\mathcal{O}^\ell$. There are two reasons for maintaining two separate matrices, which we describe below.

Regularization First, having a set of operations $\mathcal{O}^\ell$ allows SAP to learn which sets, corresponding to weight subspaces of a full-rank matrix, are relevant for a given task distribution. Choosing lower-dimensional subspaces is a form of regularization, as fewer parameters can be adjusted to fit the noise in the data.

Gradient modulation Second, when computing gradient updates for the operation parameters ϕ^ℓ of a given layer ℓ, the frozen base-layers $\mathbf{W}^\ell$ implicitly modulate the gradients since the error signal traverses backward through $\mathbf{W}^\ell$ to $\mathcal{O}^\ell$. This method of gradient modulation was proposed by Lee and Choi (2018). Below, we borrow the analysis performed in that paper to illustrate the modulation.

Suppose we are presented with a task $\mathcal{T}_j$ and that the output for a given layer in the network is given $\mathbf{v} = \mathbf{W}\mathcal{O}\mathbf{x}$, where $\mathbf{x}$ is the input to the layer. Moreover, assume that the loss of the network on task $\mathcal{T}_j$ is given by $\mathcal{L}_{\mathcal{T}_j}$ Then, the parameters of the operations $\mathcal{O}$ are updated using a gradient update step, and we obtain the new output

$$\mathbf{v}^{new} = \mathbf{W}(\mathcal{O} - \alpha\nabla_{\mathcal{O}}\mathcal{L}_{\mathcal{T}_j})\mathbf{x} \tag{6.10}$$

$$= \mathbf{v} - \alpha(\mathbf{W}\nabla_{\mathcal{O}}\mathcal{L}_{\mathcal{T}_j})\mathbf{x}. \tag{6.11}$$

Note that we slightly abuse notation here since the parameters of the operations are denoted as ϕ. As we can see, the change in the layer's output $\Delta(\mathbf{v}^{new}, \mathbf{v})$ is negatively proportional to the $(\mathbf{W}\nabla_{\mathcal{O}}\mathcal{L}_{\mathcal{T}_j})$. Here, $\mathbf{W}$ *warps* the gradients with respect to the operation parameters. The warping of these gradients is meta-learned across tasks such that within a few gradient updates in warped space, a good performance can be achieved (Flennerhag et al., 2020; Lee and Choi, 2018; Park and Oliva, 2019b).

As a consequence, *SAP can learn both in which parameter subspaces to perform gradient descent by learning appropriate subsets of operations, as well as learn how to warp these subspaces so that few gradient updates yield good performance.*

6.5 Experiments

In this section, we aim to answer the following research questions:

- Does learning suitable layer-wise operations/subspaces improve meta-learning performance on sine wave regression? (Section 6.5.1)
- Do the learned strengths of subspaces/operations match the task structure in a simple synthetic setting? (Section 6.5.2, Section 6.5.3)
- How well does SAP perform at few-shot image classification? (Section 6.5.4)
- How well does SAP perform at cross-domain few-shot image classification? (Section 6.5.5)
- Is hard subspace pruning beneficial for the performance of SAP? (Section 6.5.6)
- What is the influence of second-order gradients on the performance of SAP? (Section 6.5.7)
- What operations are important for few-shot image classification? (Section 6.5.8)
- How does SAP compare in terms of the running time and and number of trainable parameters compared to the baselines? (Section 6.5.9)

6.5.1 Sine wave regression

First, we study the few-shot learning performance of SAP on few-shot sine wave regression, which is commonly used in the meta-learning community (Finn et al., 2017; Li et al., 2017; Park and Oliva, 2019b). Here, the goal is to learn sine wave regression tasks $\mathcal{T}_j$ corresponding to sine curves $g_j(x) = A_j \cdot \sin(x - p_j)$ from a limited set of k examples. The amplitudes A_j and phases p_j of these sine curves are randomly sampled from the intervals $[0.1, 5.0]$ and $[0, \pi]$, respectively. While the results on sine-wave regression are not our main contribution, the structure of these problems were a motivation for the development of this method, and therefore this is a good test-case on which we expect SAP to perform well. Of course, SAP can only be considered a valuable contribution when it also works on more relevant problem types, which we explore in the following sections.

We use the same base-learner architecture, a fully-connected neural network with 2 hidden ReLU layers of 40 nodes each, as in (Finn et al., 2017). For the SVD operations (see candidate operations in Section 6.4.2), we use ranks 5, 10, and 15 in the candidate pools. All candidate operations were initialized to have no effect on the network predictions at the start (transformation matrices were initialized to identity matrices, biases to 0, and scale operations to 1). All techniques are meta-trained on 70 000 tasks using one update step per task and a meta-batch size of 4. We perform validation every 2 500 tasks to select the best performing model, which will be tested after 1 and 10 gradient update steps on 2 000 meta-test tasks consisting of k support examples and 50 query data points.

As baselines, we compare against MAML, T-Net, and MT-Net (Lee and Choi, 2018) as well as Warp-MAML (Flennerhag et al., 2020) as are highly similar to SAP, which allows us to investigate the advantage of SAP's ability to learn which subsets of operations to adjust. We refrain from comparing against MetaNAS (Elsken et al., 2020), as this technique also adjusts the architecture at meta-test time and is orthogonal to SAP and the methods we compare against. For all methods, we use the same hyperparameters as reported in (Finn et al., 2017; Lee and Choi, 2018). In this case, however, Warp-MAML is equivalent to T-Net as both use insert linear "transformation" or "warp" layers in the base-learner network. The results of the experiments are displayed in Table 6.2. In this

		5-shot		10-shot	
	params	T=1	T=10	T=1	T=10
MAML	1 761	0.73 ± 0.016	0.42 ± 0.011	0.49 ± 0.011	0.15 ± 0.005
T-Net	4 962	0.53 ± 0.014	0.24 ± 0.009	0.33 ± 0.009	0.09 ± 0.004
MT-Net	5 043	0.55 ± 0.013	0.19 ± 0.005	0.34 ± 0.008	0.06 ± 0.002
SAP (ours)	10 013	**0.47** ± 0.012	**0.10** ± 0.003	**0.28** ± 0.008	**0.04** ± 0.001

Table 6.2: The mean MSE meta-test loss on 5- and 10-shot sine wave regression after $T = 1$ and $T = 10$ update steps. The results are averaged over 5 runs with different random seeds and the 95% confidence intervals are displayed as ± x. The number of parameters is shown in the column "params", even though the used backbones are equally expressive.

table, we can see that SAP consistently outperforms all tested baselines, supporting the hypothesis that it is indeed beneficial to learn in which subspaces to perform gradient descent. We have also performed experiments with SAP and the feature masking method used in MT-Net, where some features are frozen based on learned feature masking probabilities, but found that it decreases the performance, which may be due to the low-dimensional operations present in the architecture, which are more susceptible to being completely frozen as soon as a single feature is masked.

6.5.2 The learned subspaces for sine regression

Next, we investigate (in the same setting as above) the importance of the different candidate operations for quick adaptation to new tasks to see whether the operations match the task structure. We hypothesize that shifting the input and scaling the output are important operations as they are inherent in the definition of a sine wave $g_j(x) = A_j \cdot \sin(x - p_j)$. To investigate this, we inspect the activation strengths w_i^ℓ of the operations of the best models across 5 different runs with different random seeds. The operations that were used are were introduced in Table 6.1 (left side). The results for SAP with $T = 1$ are displayed in Figure 6.4 (similar results are obtained when making $T = 10$ updates and therefore omitted for brevity). As we can see, the most important transformations on the input and output are a scalar shift and multiplication, respectively. In other words, SAP has learned that shifting the input and scaling the output are effective operations to learn new tasks. Note that these operations match the structure of sine waves. While this confirms our hypothesis, SAP also assigns relatively large importance to operations that are not directly observable in the mathematical definition of sine curves such as an output shift and intermediate shifts.

6.5.3 Matching the problem structure

To further investigate the ability of SAP to match the learned candidate operation strengths to the structure of the problem, we investigate whether changes in the problem structure amount to changes in the learned activation strengths by SAP for the different operations. For this, we consider a synthetic sine wave regression problem that generalizes the settings studied by Finn et al. (2017) and Li et al. (2017). In this setting, we create different *task families* (task distributions) that are characterized by the mathematical operations inherent in the ground-truth function. All task families share the following template for the ground-truth function $g(x) = A \cdot \sin(f \cdot x - p) + \beta$, where A is the amplitude, f the frequency, p the phase, and β the output offset. What distinguishes task families is which of these parameters they include in the functional description. For example, task family A

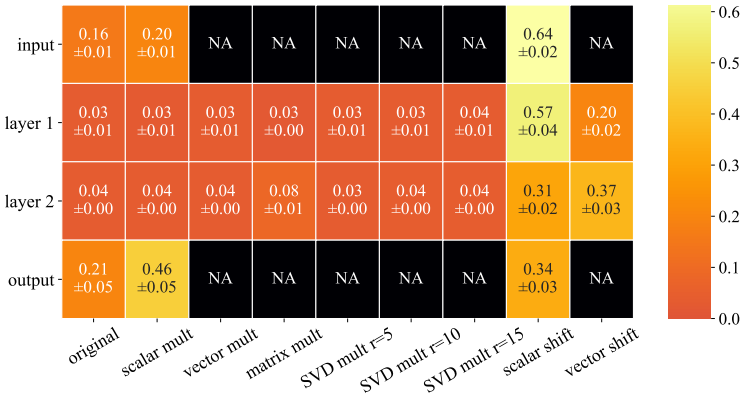


Figure 6.4: The importance of the different operations in SAP for 5-shot sine wave regression. The results are averaged across 5 runs with different random seeds and the standard deviations are shown as $\pm x$. NA entries indicate that these operations were not in the candidate pool for that layer, and "mul" means multiplication. The y-axis indicates the layer on which the operations act, and the x-axis displays the different candidate operations. Simple scalar multiplication and shifting, and vector shifts obtain high activation strengths in all layers. The input shift and output scale (inherently present in the definition of a sine wave) obtain high activation strengths.

may fix the amplitude and vary the frequency, phase, and output offset, whereas task family A may vary the amplitude and fix the rest. Each task family is thus defined by which of these parameters are varied among tasks from that family and which are kept constant. If a given parameter is not varied, we fix it to a value that leaves the function unaltered (i.e., $A, f, p = 1$ and $\beta = 0$). In total, there are $2^4 = 16$ task families that can be constructed by varying or fixing these parameters.

We perform meta-training on each of these task families separately and investigate whether SAP discovers the operations that are inherently present in the task structure. The experimental details follow those used in Section 6.5.1 with the exception that only operations were included that could be present in the task families to be able to measure whether SAP correctly detects and uses them. We use 20-shots per task and set the number of inner updates to $T = 1$. The results of this experiment are displayed in Figure 6.5. As we can see, SAP assigns higher activation strengths to operations that are inherently present in the task families in three out of four cases, i.e., input scale (frequency), input shift (phase), and output shift. A statistical T-test shows that these differences in mean activation strengths are statistically significant, using a threshold of 0.05. For the input scale, however, we observe that SAP assigns similar activation strength to the input scale activation, regardless of whether such an operation was present in the task family. This may indicate that SAP uses other operations to compensate for this, such as vector multiplications or matrix multiplications in later layers. Overall, these results suggest in this simple synthetic setting, SAP is capable of learning to use operations that appear in the problem structure in 75% of the scenarios.

6.5.4 Few-shot image classification

Next, we investigate the performance of SAP in few-shot image classification settings, where the goal is to learn new image classification tasks from a few examples. For this, we use the popular

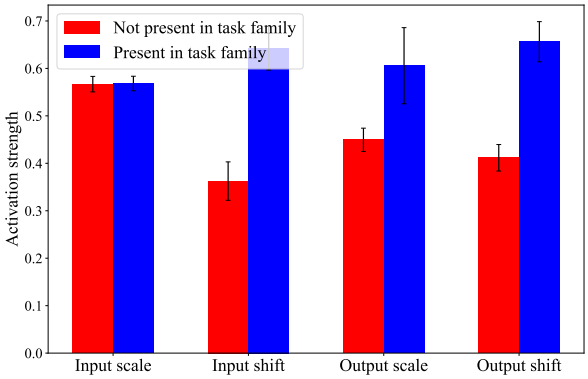


Figure 6.5: The mean activation strengths of the different operations corresponding to the intrinsic parameters that were varied within task families. The vertical bars display 95% confidence intervals over 5 runs with different random seeds. Each task family has the following template $g(x) = A \cdot \sin(f \cdot x - p) + \beta$ and differs in which of these operations are varied among tasks. If operations are inherently present in a task family, SAP assigns higher activation strengths to them than if they are not present in 3 out of 4 cases, indicating that the operations often match the problem structure.

Hyperparameter	Range
Inner learning rate	LogUniform(1e-3, 6e-1)
Inner update steps (training)	Uniform(1,10)
Inner update steps (testing)	Uniform(inner steps training, 15)
Meta-batch size	Uniform(1,10)
Gradient masking	Uniform({False, True})

Table 6.3: The ranges and sampling types for the hyperparameters, which were tuned with random search. The bounds are inclusive.

N-way k-shot classification setup (see Chapter 2) on miniImageNet (Vinyals et al., 2016; Ravi and Larochelle, 2017) and tieredImageNet (Ren et al., 2018). We use the frequently used Conv-4 backbone (Finn et al., 2017; Lee and Choi, 2018; Flennerhag et al., 2020), consisting of four blocks, where each block contains 3×3 convolutions, a max pooling layer, 2D BatchNorm, and a ReLU nonlinearity. In the literature, this backbone has been used with 64 channels for every convolutional block (Snell et al., 2017; Vinyals et al., 2016) as well as 32 channels (Finn et al., 2017; Nichol et al., 2018). For this reason, we present the results for SAP on both variants. The final feature representations are flattened and fed into a softmax output layer. All techniques were trained for 60 000 episodes and were validated after every 2 500 tasks and we use the best-reported hyperparameters by the original authors.

We tuned a subset of the hyperparameters for SAP on the *meta-validation* tasks using random search with a function evaluation budget of 30 runs. Each run was restricted to finish within 7 days on a single PNY GeForce RTX 2080TI GPU. Runs that took longer (e.g., because of a large meta-batch size) were discarded from the hyperparameter search. The used hyperparameter ranges and sampling

types that were used for the random search are displayed in Table 6.3. Due to computational constraints, we adopted the best reported hyperparameters of the baseline methods as reported in their respective papers. As such, the comparison against these baselines is in this experiment only for illustrative purposes, as the hyperparameter optimization procedure on these methods has not been executed under the same conditions.

	1-shot		5-shot	
	32 channels	64 channels	32 channels	64 channels
MAML	48.0 ± 0.8	46.7 ± 0.8	64.4 ± 0.4	63.6 ± 0.4
T-Net	48.9 ± 0.8	48.7 ± 0.8	65.3 ± 0.4	-
MT-Net	48.5 ± 0.8	49.3 ± 0.8	63.0 ± 0.4	-
Warp-MAML	49.5 ± 0.8	49.8 ± 0.8	63.9 ± 0.4	64.6 ± 0.4
SAP (ours)	**51.6** ± 0.8	**52.8** ± 0.8	**65.9** ± 0.4	**67.4** ± 0.4

Table 6.4: Meta-test accuracy scores on 5-way miniImageNet classification over 5 runs with two variants of the Conv-4 backbone, that is, with 32 or 64 channels per block. The 95% confidence intervals are displayed as ± x. "-" indicates that the experiments required more GPU VRAM than available.

The results for the experiments on 5-way miniImageNet and tieredImageNet classification are displayed in Table 6.4 and Table 6.5. Note that the results for 5-shot T-Net and MT-Net are missing as they were unable to run on our GPU with 12GB of VRAM. As we can see, the performance of the techniques improves when using 64 channels compared with 32, with the exception of MAML on miniImageNet and T-Net in the 1-shot setting on miniImageNet. As we can see, SAP consistently outperforms all tested baselines in all tested settings (with gains between 1.1% to 3.3% accuracy), indicating that it is beneficial to learn subsets of operations on which gradient descent is performed in the case of few-shot image classification.

	1-shot		5-shot	
	32 channels	64 channels	32 channels	64 channels
MAML	50.7 ± 0.8	51.5 ± 0.8	65.2 ± 0.4	66.6 ± 0.4
T-Net	49.4 ± 0.8	51.7 ± 0.8	64.6 ± 0.4	-
MT-Net	49.8 ± 0.9	51.5 ± 0.8	64.6 ± 0.4	-
Warp-MAML	51.8 ± 0.8	53.3 ± 0.8	66.0 ± 0.4	68.2 ± 0.4
SAP (ours)	**52.9** ± 0.8	**54.5** ± 0.8	**69.3** ± 0.3	**71.3** ± 0.4

Table 6.5: Meta-test accuracy scores on 5-way tieredImageNet classification over 5 runs with two variants of the Conv-4 backbone, that is, with 32 or 64 channels per block. The 95% confidence intervals are displayed as ± x. "-" indicates that the experiments required more GPU VRAM than available.

6.5.5 Cross-domain few-shot image classification

Next, we study the performance of SAP in a more challenging *cross-domain* few-shot image classification setting. In this setting, techniques are trained on tasks from dataset A and evaluated on tasks

from another dataset B, in contrast to the setting used above, where the techniques were evaluated on *unseen* tasks from the same dataset used for training. We use the same setting as Chen et al. (2019), in which we train on miniImageNet and evaluate on CUB (Wah et al., 2011). In addition, we also train on tieredImageNet (Ren et al., 2018) and test on CUB. All other experimental details are the same as above.

The results of this experiment are shown in Table 6.6. As we can see, SAP performs on par with Warp-MAML in the 1-shot setting for MIN → CUB. Both outperform the other tested baselines in that scenario. In other cases, however, SAP yields performance improvements ranging from 0.5% to 3.9% accuracy. This supports the hypothesis that it is beneficial to learn which subsets of operations to adjust when learning new tasks.

	MIN → CUB		Tiered → CUB	
	1-shot	5-shot	1-shot	5-shot
MAML	37.3 ± 0.3	54.7 ± 0.3	38.1 ± 0.3	55.1 ± 0.3
T-Net	38.0 ± 0.3	55.6 ± 0.3	37.5 ± 0.3	54.8 ± 0.3
MT-Net	37.1 ± 0.3	53.1 ± 0.3	38.0 ± 0.3	55.5 ± 0.3
Warp-MAML	**41.0** ± 0.3	55.3 ± 0.3	40.9 ± 0.3	56.8 ± 0.3
SAP (ours)	40.9 ± 0.3	**55.8** ± 0.3	**41.1** ± 0.3	**60.7** ± 0.3

Table 6.6: Average cross-domain meta-test accuracy scores over 5 runs a 32-channel Conv-4 backbone. Techniques trained on tasks from one data set were evaluated on tasks from another data set. The 95% confidence intervals are displayed as ± x.

6.5.6 Effect of hard pruning

Next, we investigate the effect of hard pruning the number of operations per layer, which is a common feature of DARTS (Liu et al., 2019), and therefore also inherited by SAP. For this, we compare the performance of SAP without hard pruning and SAP where we only retain the top-K operations as indicated by their strength scores. The hard-pruned SAP is re-trained using only the candidate operations which were not pruned. The results of this experiment with a 32-channel Conv-4 backbone are displayed in Table 6.7 (for the 64-channel variant, please see Table 5.4 in the appendix). As we can see, hard pruning can have a mild positive effect on the meta-learning performance, whilst reducing computational costs due to the fact that fewer parameters have to be trained. This also implies that some operations may indeed be suboptimal for a given task distribution, which soft-pruning is not able to completely filter out, and that a model which fully excludes these, can achieve better performance. We note, however, that the 95% confidence intervals are overlapping, suggesting that these performance increases are not significant.

6.5.7 The effect of the gradient order

All tested techniques require the computation of second-order gradients by default. Here, we investigate how the performance of SAP is affected by making a first-order approximation. We compare this first-order variant with the regular second-order variant, using the same experimental settings as used in Section 6.5.4. The results of this experiment are shown in Table 6.8. As we can see, the first-order approximation is consistently outperformed by the regular variant, with differences between 0.2% and 7.3 % accuracy, indicating that second-order gradients play an important role in

	miniImageNet		tieredImageNet	
	1-shot	5-shot	1-shot	5-shot
No pruning	51.6 ± 0.8	65.9 ± 0.4	52.9 ± 0.8	69.3 ± 0.3
Top-1	51.4 ± 0.8	65.8 ± 0.4	52.8 ± 0.8	69.4 ± 0.4
Top-2	**51.8** ± 0.8	**66.3** ± 0.4	**53.4** ± 0.8	69.4 ± 0.4
Top-3	**51.8** ± 0.8	**66.3** ± 0.4	53.0 ± 0.9	69.9 ± 0.4

Table 6.7: Mean meta-test accuracy scores on miniImageNet and tieredImageNet with 95% confidence intervals over 5 different runs. We used a Conv-4 backbone with 32 channels for these results.

achieving good performance.

	miniImageNet		tieredImageNet	
	1-shot	5-shot	1-shot	5-shot
SAP (first-order)	51.4 ± 0.8	63.7 ± 0.4	47.2 ± 0.8	62.0 ± 0.4
SAP (second-order)	**51.6** ± 0.8	**65.9** ± 0.4	**52.9** ± 0.8	**69.3** ± 0.3

Table 6.8: Meta-test accuracy scores on miniImageNet and tieredImageNet classification over 5 runs using the Conv-4 backbone with 32 channels. The 95% confidence intervals are displayed as ± x.

6.5.8 The learned subspaces for image classification

In order to gain insight into what operations are important for achieving good few-shot learning performance in SAP, we investigate the learned activation strengths for the different candidate operations. The operations that were used are were introduced in Table 6.1 (right side). In Figure 6.6, we can see these learned strengths in SAP on 1-shot 5-way miniImageNet using the Conv-4 backbone with 32 channels (similar patterns are seen for the backbone with 64 channels as can be seen in Figure C.1 in the appendix). As we can see, high-dimensional convolutional operations (conv1x1, conv3x3, convSVD) obtain low activation strengths, while lower-dimensional subspaces/operations such as shifts (scalar and vector) and MTL scale yield larger strengths. The greatest strength is assigned to the former throughout all layers. This may indicate that the higher-dimensional operations lead to overfitting, while the lower-dimensional operations are more suited for adapting to tasks when only limited data is available. Consequently, this implies that it is indeed beneficial to adapt subsets of operations when learning new tasks.

6.5.9 Number of parameters and running time

Lastly, we compare the running times and the number of parameters used by the different methods on few-shot image classification. These statistics were measured whilst performing the experiments in Section 6.5.4 and the results are displayed in Table 6.9. As we can see, SAP has the largest number of parameters, even though the backbone is equally expressive as that used by others. The running time of SAP, however, is often less than that of the baselines. This is caused by the fact that all methods use different hyperparameter settings in order to optimize the performance, which relates to the running time. For example, a larger meta-batch size or number of updates per task leads to an increase in running time. SAP uses the smallest meta-batch size and number of updates and

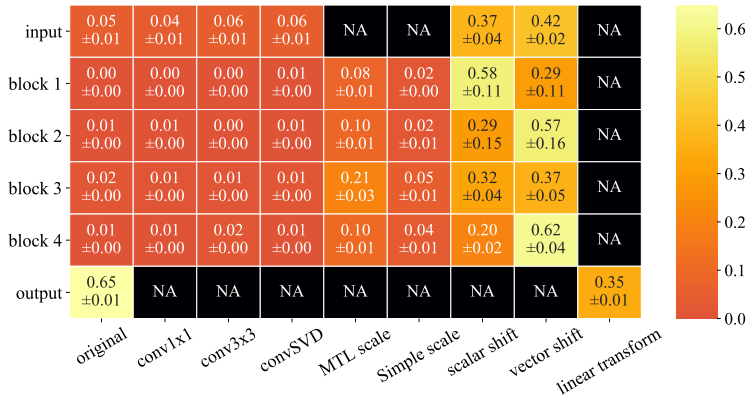


Figure 6.6: The importance of the different subspaces/operations in SAP on 5-way 1-shot miniImageNet using Conv-4 with 32 channels. The results are averaged across 5 runs with different random seeds and the standard deviations are shown as $\pm x$. NA entries indicate that these operations were not in the candidate pool for that layer. Simple scalar shift and vector shift operations obtain the highest activation strengths throughout the convolutional network.

hence yields the quickest running time. Note that these runtimes do not include the hyperparameter optimization that was performed, which adds a factor to the runtimes.

		miniImageNet		tieredImageNet	
	params	1-shot	5-shot	1-shot	5-shot
MAML	32 901	11h36min ± 7min	8h20min ± 1min	11h34min ± 5min	8h25min ± 4min
T-Net	37 022	33h05min ± 19min	30h20min ± 7min	33h25min ± 23min	30h30min ± 17min
MT-Net	37 150	33h33min ± 8min	30h22min ± 14min	33h49min ± 40min	30h47min ± 18min
Warp-MAML	60 645	7h19min ± 6min	7h17min ± 8min	7h35min ± 13min	7h19min ± 5min
SAP (first-order)	106 196	1h26min ± 2min	1h4min ± 0min	1h51min ± 6min	2h12min ± 4min
SAP	106 196	3h59min ± 0min	6h09min ± 0min	1h51min ± 6min	9h24min ± 19min

Table 6.9: The number of trainable parameters ("params") and mean running times on miniImageNet and tieredImageNet classification over 5 runs using the Conv-4 backbone with 32 channels. The standard deviations are displayed as ± x min. In spite of the differences in the number of parameters, the backbones are equally expressive. SAP was found to work best with a small meta-batch size and number of updates per task compared with the other approaches and hence yields the quickest running time.

6.6 Conclusions

In this chapter, we introduced, *Subspace Adaptation Prior* (SAP), a novel meta-learning algorithm that jointly learns a good neural network initialization and good parameter subspaces (or subsets of operations) in which new tasks can be learned within a few gradient descent updates from a few data. SAP overcomes the limitations of current state-of-the-art gradient-based meta-learning techniques

which perform gradient descent in *full parameter space* as they adjust all parameters (Finn et al., 2017; Lee and Choi, 2018; Flennerhag et al., 2020), which may be suboptimal, and may lead to overfitting during few-shot learning. Note, however, that our goal is not to yield state-of-the-art performance. Instead, we investigate the question of whether the few-shot learning performance of deep neural networks can be improved by meta-learning which subsets of parameters to adjust.

Our experiments show that SAP outperforms similar existing gradient-based meta-learners in few-shot sine wave regression, yields better performance in single-domain few-shot image classification settings, and yields competitive or superior performance in cross-domain few-shot image classification. This highlights the advantage of learning suitable subspaces in which to perform gradient descent when learning new tasks. This could be due to the regularization effect of not having to adjust all parameters as well as due to the ability to match structures inherently present in task families. Our experiments in Section 6.5.3 on synthetic task families demonstrate that the SAP is able to learn operations that match the task structure in simple settings in 75% of the cases. In other cases, it may compensate by using other operations that are not inherently present in the task structure.

Inspection of the subspace activation strengths in few-shot image classification reveals that simple and low-dimensional operations, such as shifting features by a single scalar or element-wise by a vector, are important. This is in line with recent work and findings (Triantafillou et al., 2021; Requeima et al., 2019; Bateni et al., 2020) which show that adapting pre-trained embeddings by means of such low-dimensional transformations, such as FiLM layers (Perez et al., 2018), can yield excellent performance. Furthermore, we found that hard-pruning the subspaces in SAP, or operations, such that only a discrete subset is used instead of a convex combination, was slightly beneficial, although no statistically significant differences were found.

Future work One limitation of SAP is that it requires the computation of second-order gradients by default during meta-training in order to update the initialization parameters, in a similar fashion as other gradient-based meta-learners such as MAML (Finn et al., 2017), (M)T-Net (Lee and Choi, 2018), and Warp-MAML (Flennerhag et al., 2020). These second-order gradients require $O(N^2)$ storage, where N is the number of total network parameters, which is prohibitive for deep networks. This limitation can be bypassed by using a first-order approximation, which comes at the cost of a performance penalty (between 0.2% and 7.3% accuracy in our experiments).

Gradient-based meta-learning methods struggle to scale well to deep networks as recent work suggests that simple pre-training and fine-tuning of the output layer (Tian et al., 2020; Chen et al., 2021) (also see Chapter 4) can yield superior performance on common few-shot image classification benchmarks. This is also the reason, besides searching for energy-efficient few-shot learners, that in our experiments we focus on relatively shallow backbones that adapt all layers when learning new tasks, instead of only the output layer.

Other limitations are that SAP introduces more parameters and that the candidate pools of operations are selected by hand, despite the fact that these operations are general. One direction for future work could be to design a method to discover such subspaces from scratch, instead of relying on a candidate set of operations, perhaps using an auto-encoder that generates the weights of a layer based on latent codes as used by Rusu et al. (2019). Masking the adaptation of these latent codes using Gumbel-softmax (Jang et al., 2017; Maddison et al., 2017) as done by MT-Net (Lee and Choi, 2018) would amount to adjusting only a subset of the parameters when performing gradient descent. This can reduce the number of parameters and may also help to scale gradient-based meta-learners, including SAP, to deep networks and make them competitive with approaches relying on pre-trained features, which is an open challenge.

Finally, orthogonal work has proposed a method that can also adjust the architecture during the meta-test phase (Elsken et al., 2020). Since this showed great potential, it would be worthwhile to combine this with SAP. Moreover, it would be interesting to investigate the sensitivity of SAP related methods such as MetaNAS to the chosen operations or blocks that these methods can select to use. We leave these ideas for future work, which has the potential to further advance the state-of-the-art.

This chapter is the final chapter that presents research results. In the next chapter, we take a step back and return to the big picture, revisiting and answering our original research questions, and proposing directions for future work.