Meta-Learning Algorithms and Applications

Ondrej Bohdal

Doctor of Philosophy

Institute of Perception, Action and Behaviour

School of Informatics

University of Edinburgh

2023

Abstract

Meta-learning in the broader context concerns how an agent learns about their own learning, allowing them to improve their learning process. Learning how to learn is not only beneficial for humans, but it has also shown vast benefits for improving how machines learn. In the context of machine learning, meta-learning enables models to improve their learning process by selecting suitable meta-parameters that influence the learning. For deep learning specifically, the meta-parameters typically describe details of the training of the model but can also include description of the model itself – the architecture. Meta-learning is usually done with specific goals in mind, for example trying to improve ability to generalize or learn new concepts from only a few examples.

Meta-learning can be powerful, but it comes with a key downside: it is often computationally costly. If the costs would be alleviated, meta-learning could be more accessible to developers of new artificial intelligence models, allowing them to achieve greater goals or save resources. As a result, one key focus of our research is on significantly improving the efficiency of meta-learning. We develop two approaches: EvoGrad and PASHA, both of which significantly improve meta-learning efficiency in two common scenarios. EvoGrad allows us to efficiently optimize the value of a large number of differentiable meta-parameters, while PASHA enables us to efficiently optimize any type of meta-parameters but fewer in number.

Meta-learning is a tool that can be applied to solve various problems. Most commonly it is applied for learning new concepts from only a small number of examples (few-shot learning), but other applications exist too. To showcase the practical impact that meta-learning can make in the context of neural networks, we use meta-learning as a novel solution for two selected problems: more accurate uncertainty quantification (calibration) and general-purpose few-shot learning. Both are practically important problems and using meta-learning approaches we can obtain better solutions than the ones obtained using existing approaches. Calibration is important for safety-critical applications of neural networks, while general-purpose few-shot learning tests model's ability to generalize few-shot learning abilities across diverse tasks such as recognition, segmentation and keypoint estimation.

More efficient algorithms as well as novel applications enable the field of meta-learning to make more significant impact on the broader area of deep learning and potentially solve problems that were too challenging before. Ultimately both of them allow us to better utilize the opportunities that artificial intelligence presents.

Lay Summary

In machine learning we use data to teach machines how to solve given tasks, for example recognizing objects in images or translating texts to other languages. Such approach can be contrasted with programming, where we explicitly specify what actions the machine should perform. A very popular and successful approach for machine learning utilizes the concept of neural networks. Neural networks can be understood as a series of mathematical operations that are applied in a sequence to the input data. These operations depend on parameters, also known as weights, that need to be chosen appropriately. Selecting them is not trivial, so mathematical techniques have been developed for finding suitable values. Initially random values are used and then they are iteratively updated based on how well they perform on data – also known as training. The values are updated in the direction that minimizes the amount of mistakes made.

How the training is performed depends on various choices and can be summarized as a set of parameters. These are typically known as hyperparameters or meta-parameters, and could be chosen manually by trial and error or based on intuition. However, selecting them in a more principled way is likely to lead to significantly better results, especially if there are many of them. The task of automatically selecting or learning the values of these parameters is known as meta-learning and is the topic of our thesis.

A key challenge with meta-learning is that it is often computationally costly. Hence one focus of our thesis is on improving the computational efficiency of meta-learning. We develop two methods that significantly decrease the computational costs in two common scenarios and make meta-learning more scalable and accessible.

Meta-learning is a general-purpose technique that can improve performance in many applications. To showcase the impact that meta-learning can make, another focus of our thesis is on using meta-learning as a new solution to selected problems related to machine learning. We present two new use-cases. The first use-case is about trying to obtain a more accurate quantification of uncertainty, which is important in safety-critical applications as we should not rely on predictions that have a large uncertainty. The second use-case concerns learning new concepts from a small number of examples, which is typically challenging for neural networks but very useful in practice. Compared to earlier work, we extend these data-efficient learning abilities to situations where one neural network can adapt to diverse computer vision tasks.

Acknowledgements

I would first like to thank my supervisor Timothy Hospedales for all his support during my PhD studies. I have learned a lot from Tim and really enjoyed working with him on the many projects that we have worked on together. I have grown significantly as a researcher under his supervision, and I am very grateful for all his guidance and help.

I would also like to thank my friends from the Centre for Doctoral Training in Data Science, Machine Intelligence Group, our office, the School of Informatics and more widely. I have had a fantastic time with them and enjoyed our daily lunches, many events and celebrations, and our conversations, which were both inspiring and fun.

My thanks also go to my collaborators, with whom we have accomplished to write a number of papers, several of which are included in this thesis. Special thanks go to Da Li from Samsung AI Center and Giovanni Zappella from Amazon Web Services for hosting me as an intern. It was a pleasure to collaborate with them all, and I look forward to more collaborations with them in the future.

Further I thank Frank Hutter and Siddharth Narayanaswamy for examining my PhD thesis, a friendly discussion on the topic of meta-learning and research more broadly during the PhD viva, as well as valuable suggestions for improving the thesis.

Finally I thank my family, especially my parents for raising me and all their support and love during the years. I am also grateful to my brother for providing great company when growing up. All of the achievements would not be possible without them.

My research was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh. I was also supported by the Alan Turing Institute as part of the Enrichment scheme, and Samsung AI Center and Amazon Web Services during my internships.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Ondrej Bohdal*)

Table of Contents

Chapter 1

Introduction

1.1 Motivation

One of the intriguing human abilities is the ability to learn how to improve one's learning process. Such higher-level learning can occur on the conscious level but also on the subconscious level. On the conscious level, humans can reflect on how they have learned to solve given problems or tasks and develop strategies that can help them solve new problems more quickly and more successfully. This occurs in various settings, including in education (Biggs, 1985) and can include students preparing for exams in a new subject, by making use of strategies that help them succeed. For example, their strategy could include first writing short summaries of lectures from the given subject and then solving many similar exams from the previous years.

On the subconscious level the strategies can be more subtle as in such case there is no formal reasoning that takes place to improve the learning process. However, it has been demonstrated that this form of learning also occurs in humans. Such learning allows us to learn to solve new tasks more quickly and more accurately after we have been solving similar tasks earlier (Harlow, 1949). Neuroscientists have studied this phenomenon in more depth and identified that prefrontal cortex can act as a meta-reinforcement learning system (Wang et al., 2018a). In fact, the described meta-learning is not only limited to humans, but also happens in other living organisms, for example monkeys (Harlow, 1949) and mice (Zhao et al., 2023b). It has been therefore observed that both humans and animals are able to learn how to learn, suggesting they are able to improve their own learning processes. The improvements in learning can result in being able to learn to solve related tasks more quickly, but they can also take the form of finding better and more creative solutions to the tasks considered.

Biological meta-learning also takes place across evolutionary time-scales, in addition to individual lifetimes. Humans and animals benefit from learning how to adapt and improve themselves, and those that learn to do it better have a better chance of survival. This relates to the broader learning that takes place across generations and is reflected within the genome of the species. More specifically it occurs as part of the Baldwin effect that means organisms with a better ability to learn new behaviours are more successful in reproduction, hence more likely to pass their genes to future generations (Baldwin, 1896). The improvements to the genome that take place via evolution can be seen as a form of meta-learning, although on a yet higher level than discussed earlier.

Meta-learning is prevalent in the nature and brings important benefits for living organisms (Wang, 2021), which suggests these abilities could be highly valuable also for machines. Machines that can learn how to improve their learning process could therefore benefit tremendously and improve their chances of success for the tasks to which they are deployed.

Meta-learning and automated ML (AutoML) (Yao et al., 2018; Hutter et al., 2019) more broadly have the potential to transform how we do deep learning. They could automatically find suitable solutions to how we train deep learning models for the given problem rather than having to define the details manually (Hospedales et al., 2021). The mentioned paradigm shift is related to the advent of deep learning in areas such as computer vision (CV) or natural language processing (NLP), where learned feature extractors replaced and later significantly outperformed manually designed feature extractors (Krizhevsky et al., 2012; He et al., 2016; Devlin et al., 2019). We illustrate the increasing levels of automation in Figure 1.1, starting with machine learning, continuing with deep learning and then meta-learning. In this sense meta-learning has the ability to unlock the next wave of progress in deep learning, machine learning or artificial intelligence more broadly. Furthermore, meta-learning has even been seen as a key component towards achieving artificial general intelligence (AGI) (Clune, 2019).

The possibility of neural networks being able to automatically improve themselves has fascinated researchers for a long time (Schmidhuber, 1987; Naik and Mammone, 1992; Bengio et al., 1997; Pratt, 1997). However, it is only relatively recently that the field of meta-learning has attracted significant attention (Hospedales et al., 2021). The resurgence of interest has started when meta-learning approaches were applied with great success to the challenging problem of few-shot learning (Wang et al., 2020). Few-shot learning refers to learning new concepts from only a very small number of examples, which is considered particularly challenging for neural networks as they are

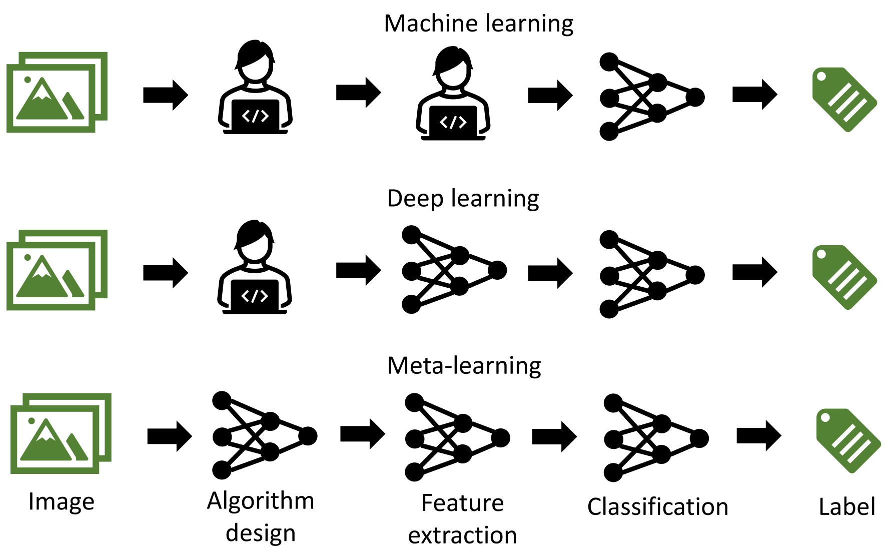


Figure 1.1: Comparison of the level of automation in machine learning, deep learning and meta-learning. Deep learning has improved upon standard machine learning by automatically learning how to extract features, which was previously done manually. Meta-learning (or AutoML more broadly) iterates on this further by also automating to a significant extent the algorithm design that guides training of deep-learning models.

widely believed to require large amounts of data for learning new concepts (Marcus, 2018). The ability to learn new concepts from only a small amount of data would be highly desirable and is also something that humans can do easily (Brown and Kane, 1988; Lake et al., 2015).

The seminal meta-learning approach that has achieved remarkable performance on the few-shot learning problem is called Model-Agnostic Meta-Learning (MAML) (Finn et al., 2017). We briefly discuss it as part of the introduction to provide an initial overview of how meta-learning methods operate. MAML is intuitive and elegant, and includes meta-training across a large number of learning tasks, also known as episodes. Such episodic meta-learning has been widely used also in many other later works, including (Snell et al., 2017; Sung et al., 2018; Lee et al., 2019). The learning consists of an inner and outer loop. The inner loop is about learning the model parameters that can solve the given task. The outer loop is about meta-learning meta-parameters that enable us to successfully learn across tasks. The described process can be seen as a form of fast learning on the current task and slow learning that takes place across tasks. In

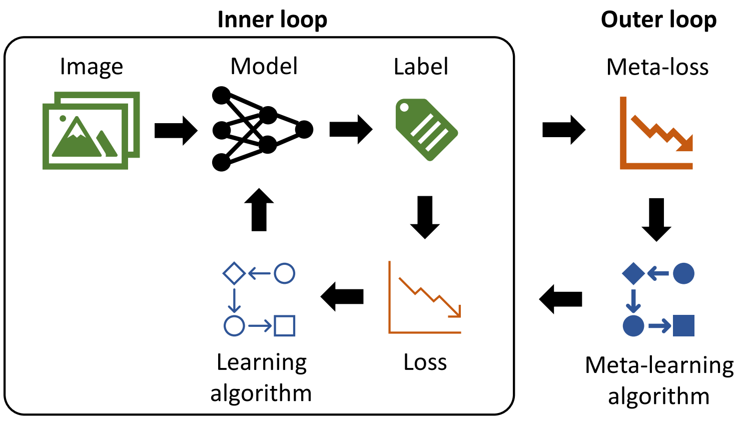


Figure 1.2: Illustration of how meta-learning operates as part of the outer loop wrapped around conventional learning done within the inner loop. The outer loop takes the inner loop training as input, evaluates it on novel data and updates the meta-parameters that influence the training in the inner loop.

the case of MAML, it is the initialization of the neural network that is meta-learned and reused across the different learning episodes. Both inner and outer loop are gradient-based in MAML, so meta-learning is achieved by backpropagating through the learning process. We illustrate the inner and outer loop in Figure 1.2.

Few-shot learning is closely related to in-context learning in large language models (LLMs) (Brown et al., 2020; Chowdhery et al., 2023; Touvron et al., 2023), an area which has recently become very popular (Dong et al., 2023). As part of in-context learning, we give the LLM one or a few examples as context, and the model then adapts its response accordingly, via analogy. The context is included as part of the prompt with our query and the adaptation is done without updating the model parameters. The main benefit of in-context learning is that it helps the model give significantly more personalized responses to our queries, making them more helpful.

While few-shot learning (Wang et al., 2020) is the most famous application of meta-learning, meta-learning has been useful for many other problems too (Hospedales et al., 2021). These include learning new optimizers (Andrychowicz et al., 2016; Wichrowska et al., 2017), beneficial loss functions (Bechtle et al., 2021; Gao et al., 2022a), neural architecture search (NAS) (Liu et al., 2019a; Zoph and Le, 2017), making the models more robust to domain shift (domain generalization) (Balaji et al., 2018; Li et al., 2018a), and improved ability to deal with label noise present in the training data (Li et al., 2019a; Gao et al., 2021). As we can see, meta-learning has a wide range of applications within deep learning. Improvements to how we perform meta-learning can therefore have a wide impact, including in real-world applications such as prediction of clinical risk (Zhang et al., 2019), identifying fraudulent transactions (Zheng et al., 2020b), or giving feedback to students based on a few annotated examples (Wu et al., 2021).

Having discussed meta-learning more broadly, we now introduce the specific topics we focus on as part of this thesis. One of the key challenges associated with meta-learning is that meta-learning approaches are often computationally expensive (Hospedales et al., 2021). The costs come from performing optimization on two levels, one to train the model (inner loop) and the other to select the meta-parameters (outer loop). In particular, naive implementation of such optimization can include full training of the model for each update of the meta-parameters, resulting in extreme costs. Various simplifications have been developed, but overall the computational requirements of meta-learning make it challenging to use it for larger-scale problems or if given only limited resources. This leads to the need for improving the computational efficiency of meta-learning so that it can be scalable to the modern architectures needed for excellent performance. Hence one focus of our work is on studying how to make meta-learning algorithms more efficient.

Meta-learning is a general-purpose technique that can be used in a wide variety of settings and to achieve various goals (Hospedales et al., 2021). While it has already been used for many problems, for example few-shot learning, domain generalization or neural architecture search that we mentioned earlier, there are many other problems where meta-learning could make a difference. Consequently, another focus of our work is on using meta-learning as a novel solution to selected deep-learning problems where it is has not yet been applied.

1.2 Contributions

Overall as part of this thesis we explore *how* to make meta-learning algorithms *more efficient* and *what new problems* can be suitably addressed using meta-learning. More efficient meta-learning algorithms enable us to use more advanced deep-learning architectures as well as solve larger-scale problems. New meta-learning applications allow us to expand the practical impact of meta-learning and also provide improved solutions to the given problems.

The specific research questions we try to answer in the direction of efficiency are:

- *How to improve the time and memory efficiency of gradient-based meta-learning approaches?*

- *How to improve the speed of gradient-free meta-learning approaches?*

In terms of novel meta-learning applications we study the following questions:

- *How can we use meta-learning to improve uncertainty calibration of neural networks?*
- *How to extend the existing few-shot meta-learning approaches to situations where tasks have a greater amount of diversity?*

We answer these research questions and contribute to the area of meta-learning and machine learning more broadly in the following ways:

- **More efficient meta-learning algorithms:** We develop two efficient meta-learning algorithms, namely EvoGrad and PASHA. Together these two approaches enable us to efficiently optimize meta-parameters in a wide range of settings.
  - EvoGrad is an efficient gradient-based algorithm that enables us to optimize millions of meta-parameters, under the assumption that these are differentiable and directly influence the loss function (a very common scenario – e.g. regularization, but not learning rate). A key bottleneck in meta-learning is the need to backpropagate through backpropagation-based inner loop, leading to expensive higher-order gradients. EvoGrad replaces the inner loop with an evolutionary update for efficiency, while keeping the outer loop gradient-based to enable precise learning of the meta-parameters. It matches or even improves the performance of the models trained with the learned meta-parameters, yet it requires significantly less time or memory.
  - PASHA is an efficient gradient-free approach suitable for optimizing a smaller number of meta-parameters. It does not require us to be able to differentiate with respect to the meta-parameters and can also optimize meta-parameters that do not directly influence the loss function. In hyperparameter optimization, the amount of resources for training various sets of hyperparameters (meta-parameters) has direct impact on the total costs, yet it is typically overestimated and strong configurations can be found using far fewer resources. The key idea of PASHA is to progressively increase the amount of maximum resources used for training models with various sets of meta-parameters, and stop increasing the maximum resources when performance rankings of the sampled configurations stabilize. In practice the method leads to large speedups.

- **New meta-learning applications:** We utilize meta-learning as a novel solution to two challenging problems: calibration of neural networks and general-purpose few-shot learning.
  - Our proposed Meta-Calibration approach uses meta-learning and a new differentiable calibration objective to directly optimize for well-calibrated models, without any need for post-processing. The new calibration objective is used as part of the outer loop to learn meta-parameters that lead to well-calibrated models. We have obtained the best calibration by meta-learning non-uniform label smoothing, but the objective works well also for optimizing other types of meta-parameters.
  - Within our new Meta Omnium benchmark we extend existing few-shot learning approaches and evaluate their abilities to adapt to diverse task types within computer vision, including image recognition, keypoint localization and semantic segmentation. Existing computer vision benchmarks only evaluated few-shot learners within single-task settings, but we challenge them to a significantly larger extent and identify which few-shot learners are the best in multi-task settings. Our evaluation tests also the ability to adapt to completely new held-out task types.

Our contributions to the field enable us to scale meta-learning to larger settings (or solve existing ones significantly more efficiently) and also obtain better solutions to the given practically-useful problems.

1.3 Thesis Outline

Our aim is to make meta-learning algorithms more practical by improving their efficiency, and also to use them to obtain new better solutions to various challenges within the area of deep learning.

We provide the wider context and background needed to understand our work as part of Chapter 2. We first describe conventional machine learning and introduce how meta-learning extends it. We discuss the main meta-learning algorithms, covering gradient-based meta-learning as well as gradient-free approaches, most notably ones typically known as hyperparameter optimization (HPO). We then describe the various challenges that can be solved with meta-learning.

Part I: *Meta-Learning Algorithms* covers Chapters 3 and 4 and introduces our two newly-proposed efficient meta-learning algorithms: EvoGrad and PASHA. Chapters 3 and 4 correspond to the following papers (Bohdal et al., 2021, 2023a):

- **EvoGrad: Efficient Gradient-Based Meta-Learning and Hyperparameter Optimization**
  Ondrej Bohdal, Yongxin Yang, Timothy Hospedales
  Advances in Neural Information Processing Systems (NeurIPS), 2021
- **PASHA: Efficient HPO and NAS with Progressive Resource Allocation**
  Ondrej Bohdal, Lukas Balles, Martin Wistuba, Beyza Ermis, Cedric Archambeau, Giovanni Zappella
  The Eleventh International Conference on Learning Representations (ICLR), 2023

Part II: *Meta-Learning Applications* covers Chapters 5 and 6 and describes how we use meta-learning to develop new better solutions to two challenging problems: calibration of neural networks and general-purpose few-shot learning. Chapters 5 and 6 are based on these papers (Bohdal et al., 2023c,b):

- **Meta-Calibration: Learning of Model Calibration Using Differentiable Expected Calibration Error**
  Ondrej Bohdal, Yongxin Yang, Timothy Hospedales
  Transactions on Machine Learning Research (TMLR), 2023
- **Meta Omnium: A Benchmark for General-Purpose Learning-To-Learn**
  Ondrej Bohdal, Yinbing Tian, Yongshuo Zong, Ruchika Chavhan, Da Li, Henry Gouk, Li Guo, Timothy Hospedales
  Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023

Chapter 7 summarizes our work, gives conclusions and also discusses opportunities for future work in the area of meta-learning.

We include an appendix chapter for each of the four main chapters, to provide more details or give additional analyses. The appendix chapters correspond to the appendices of the individual papers.

1.4 Additional Publications

As part of the PhD studies we have published also additional papers that are not included in the thesis. We provide a list of them as part of this section. Several of the papers relate to applications of meta-learning (Bohdal et al., 2020; Li et al., 2021a; Bohdal et al., 2024), and we refer to them in Section 2.5 that summarizes existing meta-learning applications as background.

Our additional publications are as follows:

- **Flexible Dataset Distillation: Learn Labels Instead of Images**
  Ondrej Bohdal, Yongxin Yang, Timothy Hospedales
  Neural Information Processing Systems (NeurIPS) Workshop on Meta-Learning, 2020

- **A Channel Coding Benchmark for Meta-Learning**
  Rui Li, Ondrej Bohdal, Rajesh Mishra, Hyeji Kim, Da Li, Nicholas Lane, Timothy Hospedales
  Proceedings of the Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks, 2021

- **Feed-Forward Source-Free Domain Adaptation via Class Prototypes**
  Ondrej Bohdal, Da Li, Timothy Hospedales
  European Conference on Computer Vision (ECCV) Workshop on Out of Distribution Generalization in Computer Vision, 2022

- **Fairness in AI and Its Long-Term Implications on Society**
  Ondrej Bohdal, Timothy Hospedales, Philip H.S. Torr, Fazl Barez
  Proceedings of the Stanford Existential Risks Conference, 2023

- **Label Calibration for Semantic Segmentation Under Domain Shift**
  Ondrej Bohdal, Da Li, Timothy Hospedales
  International Conference on Learning Representations (ICLR) Workshop on Pitfalls of Limited Data and Computation for Trustworthy ML, 2023

- **Impact of Noise on Calibration and Generalisation of Neural Networks**
  Martin Ferianc, Ondrej Bohdal, Timothy Hospedales, Miguel Rodrigues
  International Conference on Machine Learning (ICML) Workshop on Spurious Correlations, Invariance, and Stability, 2023

- **Feed-Forward Latent Domain Adaptation**
  Ondrej Bohdal, Da Li, Shell Xu Hu, Timothy Hospedales
  Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2024

Chapter 2

Background

Both meta-learning algorithms and applications are the focus of our work, and as part of this chapter we provide the background useful for understanding our contributions. We start by summarizing conventional machine learning and introducing how meta-learning extends it. We formalize meta-learning as a bilevel optimization problem, mentioning the diverse forms it can take. We then discuss gradient-based and gradient-free approaches for meta-learning, describing how some of the most common methods operate and what are their limitations. We also cover the different applications where meta-learning has been used, including few-shot learning, domain generalization, neural architecture search and others.

2.1 Conventional Machine Learning

We start by a brief summary of how standard machine learning models are trained and evaluated. The training is done on a dataset $\mathcal{D}^{train} = \{(x_i, y_i)\}_{i=1}^{i=N}$ that includes N datapoints, with (x_i, y_i) representing the i-th example and its label. A predictive model $\hat{y} = f_\theta(x)$ is parameterized by θ and trained by solving the following optimization problem:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}^{train}; \theta, \omega),$$

where $\mathcal{L}$ is the loss function measuring the error between the predicted and true label, and ω is the set of hyperparameters that influence how the model is trained. Validation set $\mathcal{D}^{val}$ is commonly used for selection of the best checkpoint and hyperparameters, and is separate from the training set. Evaluation of how well the model generalizes is done on a further separate set of examples $\mathcal{D}^{test}$.

Conventional machine learning is characterised by using a pre-specified set of hyperparameters ω that define how the model is trained. Their choice is critical as they have a large influence on how successful the trained model is for the given problem. However, selecting their values manually by trial and error is possible only in the most limited cases and is not scalable, especially to settings that can include thousands or millions such parameters.

Meta-learning does not assume meta-parameters ω are fixed, and instead it aims to learn them so that the learning process can be improved and be more successful. Meta-parameters ω are often meta-learned by learning across many tasks, and then used for the new task that we want to solve. However, single-task cases also exist and are relatively common.

2.2 Meta-Learning as a Bilevel Optimization Problem

Meta-parameters (also known as hyperparameters or meta-knowledge) are parameters that specify how the neural network learns. When learning them, there are two levels of learning happening. There is an inner-level (base) learning that updates the parameters of the main (base) model. This part corresponds to standard training of machine learning models. There is also an outer-level (meta) learning that updates the meta-parameters based on how they influence the main model training. This gives rise to the bilevel optimization problem where we perform optimization on two levels: inner and outer level. We have illustrated this process earlier in Figure 1.2.

Meta-learning is done by training across episodes, which often represent various tasks sampled from a task distribution $\mathcal{T} \sim p(\mathcal{T})$. In such case, the inner loop corresponds to learning to solve a new task $\mathcal{T}$ that has data $\mathcal{D}_{\mathcal{T}}$, with the goal of learning meta-knowledge ω that enables us to learn to solve new tasks better:

$$\min_{\omega} \mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})} \mathcal{L}(\mathcal{D}_{\mathcal{T}}, \omega).$$

The training $\mathcal{D}_{\mathcal{T}}^{train}$ data in an episode are called the *support* set, while the evaluation data $\mathcal{D}_{\mathcal{T}}^{val}$ are called the *query* set. For meta-learning across tasks, there are three stages: *meta-training*, *meta-validation* and *meta-testing*. During meta-training we learn the meta-knowledge by training across tasks. Meta-validation tasks are used for selecting the meta-knowledge checkpoint that generalizes the best and also for selection of parameters that influence the meta-training. During meta-testing, we sample new unseen tasks to evaluate the quality of the learned meta-knowledge by using it to train new models. Disjoint sets of examples are used in the different meta-learning stages (in some common cases using separate classes). Analogous to conventional learning, meta-learning has the concepts of *meta-underfitting* and *meta-overfitting*. Meta-overfitting in particular occurs when the meta-parameters do not generalize to new tasks, for example due to meta-training on insufficient number of tasks. Note that it is possible to have episodes sampled from only one task, but then evaluation is done on the same task.

Meta-training is usually solved as bilevel optimization. Before providing a formal definition, we give a taxonomy of meta-learning components (Hospedales et al., 2021):

- Meta-representation that defines *what* the meta-parameters ω are that we meta-learn (e.g. regularization parameters).
- Meta-optimizer that defines *how* we learn the values of these meta-parameters (e.g. using Adam optimizer (Kingma and Ba, 2015)).
- Meta-objective that specifies the goal of meta-learning and can be understood as a reason *why* the meta-parameters influence the main model in the given way. The meta-objective is influenced by the outer loss $\mathcal{L}^{meta}$, how the data for evaluating the loss are sampled and also the flow of data between the learning levels.

Within meta-learning, the goal is to find meta-parameters ω that minimize the meta-loss $\mathcal{L}^{meta}$ (outer objective) of the model parametrized by θ and trained with loss $\mathcal{L}^{base}$ (inner objective) and ω:

$$\begin{aligned} \omega^* &= \arg\min_{\omega} \mathcal{L}^{meta}\left(\mathcal{D}^{val}_{\mathcal{T}}; \theta^*\right) \\ \text{such that} \quad \theta^* &= \arg\min_{\theta} \mathcal{L}^{base}\left(\mathcal{D}^{train}_{\mathcal{T}}; \theta, \omega\right). \end{aligned} \tag{2.1}$$

Note that meta-parameters ω are directly used only for training the base model and not during the outer loop, reflecting the typical use-case. While parameters of the base model θ are commonly obtained using gradient-based approaches for neural networks, meta-parameters ω can be selected with or without gradient-based approaches.

2.3 Gradient-Based Meta-Learning

2.3.1 Offline Variation

The key feature of gradient-based meta-learning (GBML) is that meta-parameters ω are updated by gradient descent on $\mathcal{L}^{meta}$ with respect to ω, which requires backpropagating through the inner loop. By default the inner loop includes full training of the base model from scratch, per each update of the meta-parameters. Full training is only viable in limited cases, for example in few-shot learning where training of the model takes only a few steps (Finn et al., 2017; Antoniou et al., 2019). Meta-learning with full training in the inner loop is also known as offline meta-learning (Hospedales et al., 2021) and is summarized in Algorithm 1. The challenge with full training in the inner loop is not only that the training itself takes time, but also that backpropagation through the inner loop requires memory proportional to the number of inner loop steps when using common reverse-mode-differentiation implementations (Hospedales et al., 2021). Backpropagation through long inner loops typically also suffers from gradient degradation and instability (Hospedales et al., 2021; Metz et al., 2018, 2022).

Algorithm 1 Offline meta-learning

```
1:  Input: $N$: number of inner-loop steps, $\alpha$: inner-loop learning rate, $\beta$: outer-loop
    learning rate
2:  Output: trained base model $\theta$ and meta-parameters $\omega$
3:  $\omega \sim p(\omega)$
4:  while $\omega$ not converged do
5:      $\theta_0 \sim p(\theta)$
6:      for $i$ in $1..N$ do
7:          Sample training examples $\mathcal{D}_{\mathcal{T}}^{train}$
8:          $\theta_i = \theta_{i-1} - \alpha \nabla_{\theta_{i-1}} \mathcal{L}^{base}\left(\mathcal{D}_{\mathcal{T}}^{train}; \theta_{i-1}, \omega\right)$
9:      end for
10:     Sample validation examples $\mathcal{D}_{\mathcal{T}}^{val}$
11:     $\omega \leftarrow \omega - \beta \nabla_{\omega} \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta_N\right)$
12: end while
```

2.3.2 Online Variation

In order to apply meta-learning also to larger-scale settings (Shu et al., 2019; Xia et al., 2021), the full training in the inner loop can be replaced by a shorter alternative, in particular one-step update taken from the current parameters of the base model. This version is called online meta-learning (Hospedales et al., 2021) and means we jointly train the base model and the meta-parameters, as summarized in Algorithm 2. We perform one step to update the base model in the inner loop and then we perform one step to update the meta-parameters, backpropagating through the one-step inner loop. The process is repeated until the base model is fully trained. The described way of online meta-learning is also called $T_1 - T_2$ (Luketina et al., 2016), with the name selected by referring to the training set as T_1 and the validation set as T_2. Instead of using only one step in the inner loop, a few steps can also be used, and the process is often also known as truncated inner loop (Maclaurin et al., 2015; Shaban et al., 2019). A key limitation of online meta-learning is that it suffers from short-horizon bias (Wu et al., 2018), so the learned meta-parameters are likely to be sub-optimal with regard to the whole training. Nevertheless, online meta-learning is in many cases the only viable option due to computational costs.

Algorithm 2 Online meta-learning

```
1: **Input:** $\alpha$: inner-loop learning rate, $\beta$: outer-loop learning rate
2: **Output:** trained base model $\theta$ and meta-parameters $\omega$
3: $\omega \sim p(\omega)$
4: $\theta \sim p(\theta)$
5: **while** $\theta$ not converged **do**
6:     Sample training $\mathcal{D}_{\mathcal{T}}^{train}$ and validation examples $\mathcal{D}_{\mathcal{T}}^{val}$
7:     $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{base}\left(\mathcal{D}_{\mathcal{T}}^{train}; \theta, \omega\right)$
8:     $\omega \leftarrow \omega - \beta \nabla_\omega \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta\right)$
9: **end while**
```

2.3.3 Decomposition of Gradients

To learn meta-parameters ω according to Equation 2.1 using gradient-based methods, we need to calculate the hypergradient $\nabla_\omega \mathcal{L}^{meta}$. The hypergradient can be expanded using the chain rule as follows:

$$\frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \omega} = \underbrace{\frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \omega}}_{\text{direct gradient}} + \underbrace{\frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega}}_{\text{indirect gradient}}. \tag{2.2}$$

The expanded formula includes the direct gradient and the indirect gradient. The direct gradient is often zero because in most cases meta-parameters ω do not directly influence loss $\mathcal{L}^{meta}$. Meta-parameters ω influence loss $\mathcal{L}^{meta}$ indirectly by impacting how the model parameters θ are updated. Note that the direct gradient can be non-zero, for example when meta-learning the initial parameters of the model. A first-order variant of the popular few-shot learning approach MAML (Finn et al., 2017) relies on it and approximates the hypergradient with only the direct term. Such approximation leads to large speedups and memory savings, with marginal decrease in performance.

A key challenge with the indirect gradient is that it includes higher-order derivatives. We can see how the higher-order derivatives arise by substituting the gradient-based update of the model parameters $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{base}\left(\mathcal{D}_{\mathcal{T}}^{train}; \theta, \omega\right)$ into Equation 2.2:

$$\begin{aligned}
\nabla_\omega \mathcal{L}^{meta} &= \frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \omega} + \frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega} \\
&= \frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \omega} - \alpha \frac{\partial \mathcal{L}^{meta}\left(\mathcal{D}_{\mathcal{T}}^{val}; \theta^*(\omega)\right)}{\partial \theta^*(\omega)} \frac{\partial^2 \mathcal{L}^{base}\left(\mathcal{D}_{\mathcal{T}}^{train}; \theta, \omega\right)}{\partial \omega \partial \theta}
\end{aligned} \tag{2.3}$$

The presence of $\nabla^2_{\omega,\theta} \mathcal{L}^{base}$ in Equation 2.3 shows we need to calculate higher-order gradients to compute the value of the hypergradient $\nabla_\omega \mathcal{L}^{meta}$. The need to calculate higher-order gradients is a bottleneck in terms of memory and time, making it challenging to use larger models. Our EvoGrad approach that we describe in Chapter 3 focuses on resolving it by using evolutionary update during the inner loop. As a result, it significantly accelerates meta-learning and uses noticeably less memory, making it easier to combine meta-learning with larger models. More detailed discussion on how higher-order gradients contribute to increased memory and time is provided in Chapter 3. Chapter 3 also includes a discussion of more advanced GBML methods, especially ones based on Implicit Function Theorem (IFT) (Lorraine et al., 2020; Rajeswaran et al., 2019) (not included in the main background chapter as we do not directly use them).

GBML methods are challenging to apply when trying to learn discrete parameters or when an objective includes non-differentiable operations (Hospedales et al., 2021). In these cases suitable approximations need to be found, or alternatively different types of meta-optimizers can be used. In particular, gradient-free methods can be useful and we discuss them next.

2.4 Gradient-Free Meta-Learning

Diverse families of approaches exist for gradient-free optimization of meta-parameters. The most notable ones include common hyperparameter optimization methods, reinforcement learning and evolutionary search.

Standard hyperparameter optimization (HPO) can be seen as a form of meta-learning because the hyperparameters are optimized and influence how well the model learns. HPO methods perform bilevel optimization since training the base model with the sampled configuration corresponds to the inner loop, and identifying what configurations to try and select as the final option corresponds to the outer loop. The focus is typically on the single-task case, which means that all learning episodes aim to solve the same problem. Note that due to how gradient-free HPO methods operate, they are typically only suitable for optimizing a smaller number of hyperparameters.

There are two families of HPO methods that are the most widely used: Bayesian Optimization and multi-fidelity methods (Feurer and Hutter, 2019). We give a comparatively more detailed discussion on the HPO methods rather than reinforcement learning and evolutionary search because one of our main contributions, the PASHA algorithm described in Chapter 4, is a multi-fidelity HPO method.

2.4.1 Basic Hyperparameter Optimization Methods

Before discussing Bayesian Optimization and multi-fidelity methods, we discuss the simplest HPO approaches: grid search and random search (Bergstra and Bengio, 2012).

As part of grid search we select a set of values for each hyperparameter that we try to tune. We then try all of the combinations, resembling a grid of values. Such way of searching for strong hyperparameters is known to be inefficient, particularly when only some of the hyperparameters have larger impact on the objective that we optimize. When there are both important and unimportant parameters, random search has been shown to be significantly more efficient, as illustrated in Figure 2.1.

Random search (Bergstra and Bengio, 2012) randomly samples combinations of values from the pre-defined distributions. Its key benefit is that by random sampling it evaluates a larger number of values of the important parameters, making it more likely to find a strong configuration with smaller number of trials. When the search space is well-designed, random search can be hard to beat. However, generally it is slow because it fully trains each candidate configuration and is not sample-efficient as it does not use a model to sample the configurations.

2.4.2 Bayesian Optimization

Bayesian Optimization methods utilize a model when deciding which configuration to evaluate next, based on information about the configurations evaluated so far (Feurer

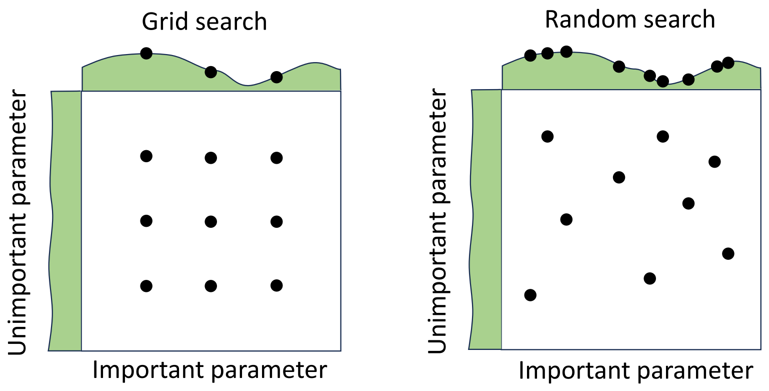


Figure 2.1: Illustration of how grid search and random search cover the search space in the presence of important and unimportant parameters. By sampling the values randomly, random search evaluates more values of the important parameter and can more efficiently find a good value. Own figure inspired by (Bergstra and Bengio, 2012).

and Hutter, 2019; Snoek et al., 2012). These methods often use Gaussian Processes to model the performance (Rasmussen and Williams, 2005; Snoek et al., 2015), but random forests (Hutter et al., 2011) and trees of Parzen estimators (TPE) (Bergstra et al., 2011) can also be used. More specifically these approaches model which parts of the search space should be investigated more to find a better configuration. Suitable trade-off between exploration and exploitation needs to be made as parts of the search space have large uncertainty, but can lead to strong configurations (Feurer and Hutter, 2019). Due to using a model to sample the configurations, such methods are known to be sample-efficient and can identify strong configurations even with fewer trials.

2.4.3 Multi-Fidelity Methods

Multi-fidelity methods sequentially allocate more resources to the more promising configurations. Compared to Bayesian Optimization, multi-fidelity methods have been faster in large-scale settings, can be more easily parallelized and have also become more popular, especially in deep learning (Feurer and Hutter, 2019).

The simplest multi-fidelity method is called *successive halving* (SH) (Karnin et al., 2013; Jamieson and Talwalkar, 2016). SH trains each candidate configuration with a small amount of resources, before successively pruning the less promising configurations and training the promising ones with more resources. Each round of promotion where we prune some of the configurations and promote the other ones by training them with more resources is called a *rung*. We illustrate SH in Figure 2.2.

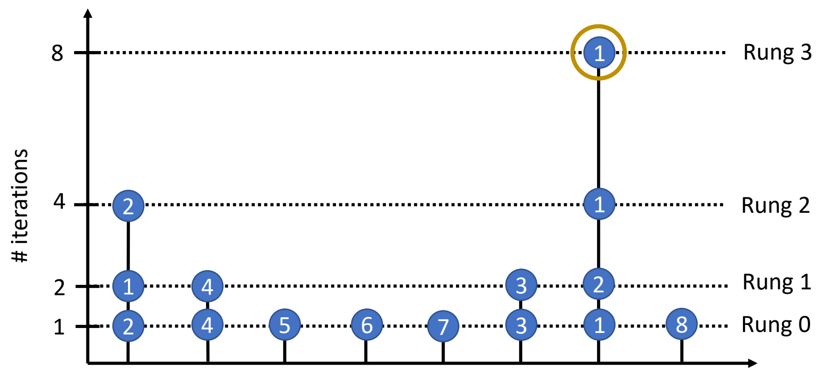


Figure 2.2: Illustration of successive halving, the simplest multi-fidelity method. Each configuration is first trained with a small amount of resources. Configurations are then repeatedly pruned, with the remaining ones trained using increasing amount of resources. Each round of promotion (or pruning) is called a rung.

More advanced multi-fidelity methods than SH exist, especially Hyperband and ASHA that are among the most popular. Hyperband (Li et al., 2018b) extends SH by running SH with various numbers of configurations for various amounts of minimum resources. ASHA (Li et al., 2020a) extends SH by performing asynchronous evaluation of the sampled configurations, which typically results in significant practical speedups due to parallelization. Note that selecting a reasonable amount of minimum resources is often relatively simple, so typically ASHA is noticeably faster than Hyperband and can find similarly strong configurations.

While ASHA is among the fastest HPO methods, there are still ways to significantly improve its speed. In particular, we identify that if the ranking of configurations stabilizes early, then resources can be saved by not training the best configurations with more resources. Based on this observation we develop a method called PASHA (described in Chapter 4). We show PASHA finds similarly strong configurations as ASHA, but usually can do it significantly more quickly, especially for cases where training a configuration takes a long time.

Approaches that combine Bayesian Optimization with multi-fidelity methods are available, as done for example in BOHB (Falkner et al., 2018). Our PASHA method can also be combined with Bayesian Optimization to enable model-based sampling of configurations, and we evaluate such extension in our experiments.

2.4.4 Reinforcement Learning

The outer-loop objective can be optimized using reinforcement learning (RL) methods, which is useful when the inner loop includes non-differentiable components (Cubuk et al., 2019) or if the outer-loop objective itself is not differentiable (Huang et al., 2019). The hypergradient is usually estimated with policy gradient methods such as REINFORCE (Williams, 2004), but such way of meta-learning turns out to be extremely costly in practice (Hospedales et al., 2021). If differentiable approximations can be made, gradient-based meta-learning approaches can drastically reduce the compute requirements. For example, a gradient-based approach for automatic data augmentation (Li et al., 2020d) has been able to obtain comparable accuracy to a RL-based approach (Cubuk et al., 2019), while being about 10,000-50,000x faster depending on the setting.

2.4.5 Evolutionary Search

A further option for optimizing the meta objective is to use evolutionary search (ES) approaches (Salimans et al., 2017; Stanley et al., 2019). ES algorithms are inspired by natural evolution and their general workflow can be described as follows (Salimans et al., 2017): at each iteration (generation), we perturb (mutate) a population of parameters (genotypes) and evaluate their objective function (fitness). We then combine the parameters to form the next generation population. The process is repeated until convergence or for the specified number of iterations, and we summarize it in Algorithm 3.

Algorithm 3 Evolutionary search

```
1: Input: $\alpha$: learning rate, $\sigma$: noise standard deviation, $n$: population size
2: Output: parameters $\omega$
3: Initialize parameters $\omega \sim p(\omega)$
4: while $\omega$ not converged do
5:     Sample $\varepsilon_1, \ldots \varepsilon_n \sim \mathcal{N}(0, I)$
6:     Evaluate fitness of each offspring $F_i = F(\omega + \sigma\varepsilon_i)$ for $i = 1, ..., n$
7:     Update $\omega \leftarrow \omega + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \varepsilon_i$
8: end while
```

ES methods have a number of advantages, including 1) the ability to optimize any type of model and meta-objective, 2) avoiding the challenges associated with backpropagation over long inner loop, and 3) being highly parallelizable (Salimans et al., 2017). Their disadvantages include 1) the need for rapidly increasing population when more parameters are tuned, 2) sensitivity to the details of the evolution strategy, and 3) worse fitting ability than gradient-based methods (Hospedales et al., 2021). ES approaches have often been applied in reinforcement learning settings (Houthooft et al., 2018; Song et al., 2020), but they have also been used for finding useful loss functions (Gonzalez and Miikkulainen, 2020), learning rules (Metz et al., 2022) and architectures (Stanley et al., 2019; Real et al., 2019).

We have been broadly inspired by ES when designing EvoGrad in Chapter 3. While ES methods are commonly used in the outer loop, we have exploited it in the inner loop for efficiency and kept the outer loop gradient-based for precision.

2.5 Applications of Meta-Learning

Meta-learning is a tool that can be used for a wide range of applications, in a way similar to how neural networks can be optimized to solve diverse tasks. Meta-learning has risen to prominence thanks to being able to obtain excellent performance on the challenging problem of few-shot learning (Hospedales et al., 2021). In this section we will discuss a number of problems for which meta-learning has been helpful. Two of our main contributions presented in this thesis (Bohdal et al., 2023c,b) consider novel applications of meta-learning, so we cover existing meta-learning applications in certain depth to provide a more complete overview.

2.5.1 Few-Shot Learning

Few-shot learning is about learning a new concept from only a few examples. It is known to be very challenging because neural networks usually require a large amount of examples for learning new concepts (Marcus, 2018). However, being able to learn from a small number of examples would have significant practical benefits as collecting and labelling large datasets is expensive.

Most commonly few-shot learning is studied for classification problems, where the goal is to learn to distinguish among N classes after seeing k examples of each (N-way k-shot learning). Meta-learning is done across tasks as described in Section 2.2. In each task we sample a combination of N classes and k support examples are sampled for each. The support examples are used in the inner loop, while further query examples from the same classes are used in the outer loop to learn the meta-parameters. We illustrate few-shot learning on a 3-way 2-shot task in Figure 2.3.

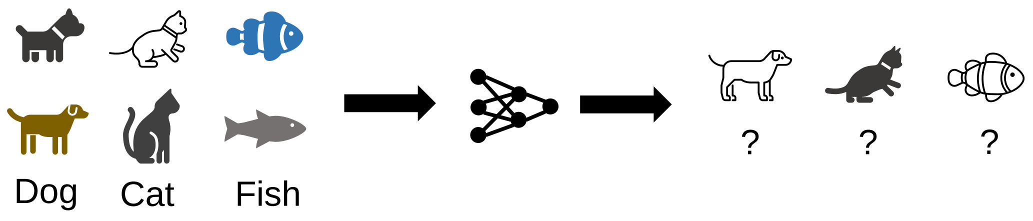


Figure 2.3: Illustration of few-shot learning. We want to learn to distinguish images of dogs, cats and fish after seeing only two examples of each.

A large number of meta-learning approaches have been proposed for few-shot learning. Some of the most famous ones include MAML (Finn et al., 2017), Prototypical networks (Snell et al., 2017) and the Relation network (Sung et al., 2018). MAML meta-learns the initial weights of a neural network that are then fine-tuned using a small number of updates on the support examples. MAML has been extended in various ways, for example by updating also the curvature information (Park and Oliva, 2019) or using a variety of techniques to stabilize its training as part of MAML++ (Antoniou et al., 2019). Prototypical networks meta-learn a metric space in which we make predictions by finding the closest prototype representation of each class. A prototype is created by computing the average feature representation of the support examples of the given class. Relation network also comes from the family of metric-based few-shot learners and compared to Prototypical networks it meta-learns a deep distance metric instead of using a fixed metric. Few-shot learning has also been studied in cross-domain settings (Tseng et al., 2020; Triantafillou et al., 2020), which more realistically reflects scenarios seen during real-world deployment.

Few-shot learning has also been studied for other problems than classification. In computer vision, it has been studied, for example, for segmentation (Min et al., 2021; Hong et al., 2022) and keypoint estimation (Lu and Koniusz, 2022; Xiao and Marlet, 2020). Few-shot learning has additionally been used in other data modalities, including text (Bragg et al., 2021) and speech (Heggan et al., 2022).

In recent years few-shot learning has gained more prominence for the text data modality, in the form of in-context learning (ICL) (Dong et al., 2023) for large language models (LLMs) (Zhao et al., 2023a). When given a small number of examples as context, LLMs are able to adapt their response and make it more personalized and helpful to the user. ICL also helps LLMs perform series of more complex tasks such as mathematical reasoning (Wei et al., 2022). Why ICL works is not fully understood, but various studies have suggested ICL may implicitly perform gradient descent (Von Oswald et al., 2023; Dai et al., 2023), drawing similarities to how MAML performs adaptation. It has also been suggested ICL may emerge as a result of how the pre-training data are distributed, including due to items appearing in clusters (Chan et al., 2022). With such structure and many different sources of data used for the pre-training stage, it can be seen as a way of pre-training across tasks, making it possible to do accurate few-shot learning.

2.5.2 Reinforcement Learning

The main focus of reinforcement learning (RL) is on learning control policies that maximize rewards obtained by performing the selected actions. RL involves a number of challenges, including sample inefficiency coming from sparse rewards, the need to explore the environment and having to deal with high-variance optimizers (Williams, 2004). Meta-learning is well-suited for RL because learning tasks naturally arise (Hospedales et al., 2021), for example, as navigation inside various environments (Mishra et al., 2018) or competition with different agents (Al-Shedivat et al., 2018).

Meta-learning has been utilized for RL in many ways, including for sample-efficient learning, learning better exploration policy or mitigating the challenges associated with optimization in RL. Few-shot learning in RL takes the form of learning a policy for a new task with only a small number of interactions with the environment. Many gradient-based meta-learning approaches can be utilized for RL (Hospedales et al., 2021), including MAML (Finn et al., 2017). Exploration strategy in RL is often based on selecting random actions (Schulman et al., 2017) or using heuristics (Sigaud and Stulp, 2019), but it can be improved by treating the exploration strategy as a set of meta-parameters that we can meta-optimize (Alet et al., 2020; Stadie et al., 2018; Garcia and Thomas, 2019). Optimization is challenging in RL due to the nature of the problem, and it has been shown that using learnable losses or rewards can be particularly beneficial (Zhou et al., 2020b; Bechtle et al., 2021; Kirsch et al., 2020). While meta-learning has many uses in RL, we have only focused on applications outside of RL in our work.

2.5.3 Domain Generalization and Adaptation

The goal of domain generalization (DG) is to train models that are robust and obtain strong performance in out-of-distribution settings (Muandet et al., 2013). The robustness is typically improved by training across various domains, with various strategies to maximize the benefits of such multi-domain training. Meta-learning approaches for

DG usually sample a domain to use in the inner loop and then evaluate performance on another domain in the outer loop (Hospedales et al., 2021).

Many meta-learning approaches have been proposed to tackle the challenge of domain generalization. MLDG (Li et al., 2018a) is one of the first such approaches, and its key idea is that updates to the parameters of the neural network using one domain should also lead to better performance on another domain. MetaReg (Balaji et al., 2018) is another popular DG method and it meta-learns L_1 regularization parameters to improve out-of-distribution generalization. (Li et al., 2019b) have meta-trained Feature-Critic networks to generate an auxiliary loss function that helps domain generalization. In the context of few-shot learning, (Tseng et al., 2020) have meta-learned noise layers that improve the ability to solve few-shot learning tasks coming from new domains.

Domain generalization is challenging, and it has been shown that under a fair evaluation protocol many of the methods do not perform better than simple training across domains (Gulrajani and Lopez-Paz, 2021). If any data from the target domain are available, they can be exploited to improve the performance more reliably. How to use such data is studied as part of domain adaptation (Csurka, 2017), and meta-learning approaches have also been proposed for this setup. Li and Hospedales (2020) have utilized meta-learning to optimize the initialization of current DA algorithms. Bohdal et al. (2024) have considered a domain adaptation setup that models the realistic conditions of adaptation on deployed devices. They have developed a method that meta-learns an attention-based mechanism to find relevant examples for adaptation.

2.5.4 Neural Architecture Search (NAS)

The architecture of a neural network has a large impact on the performance of the model. While well-performing architectures are often discovered by human insight, the process can be automatized (Elsken et al., 2019) and optimized using meta-learning. Meta-parameters specify the architecture of the neural network and are used during the inner loop to construct it and train with it. In the outer loop these meta-parameters are updated so that architectures with good validation performance can be found.

Approaches from various meta-learning families have been developed, initially using reinforcement learning (Zoph and Le, 2017) or evolutionary search (Real et al., 2019; Stanley et al., 2019). However, such approaches are often costly and gradient-based approaches can be significantly more efficient. DARTS (Liu et al., 2019a; Zela et al., 2020) have developed an approach that uses a continuous relaxation of the architecture representation and optimizes it using gradient-based meta-learning. Additionally, hyperparameter optimization techniques can be used for NAS, and we also evaluate our PASHA approach on NAS in Chapter 4.

A variety of challenges are associated with NAS (Hospedales et al., 2021):

- The search space is hard to define, given the large amount of options that can be included. The search space is hence significantly restricted, for example by searching only for the design of the cell that is reused (Liu et al., 2019a).
- NAS is costly and the discovered architecture may not transfer well to new settings, so the search needs to be carefully designed if transferability is desired (Zoph et al., 2018).
- NAS experiments have often been difficult to reproduce as small details determining how the search is done can have a significant impact on the result (Li and Talwalkar, 2020; Elsken et al., 2021).

To mitigate the lack of reproducibility, NAS benchmarks have been developed (Ying et al., 2019; Klein and Hutter, 2019; Dong and Yang, 2020; Siems et al., 2020; Mehta et al., 2022). Such benchmarks are pre-computed or use performance predictors, which makes the field of NAS more accessible.

2.5.5 Neural Network Optimization

Meta-learning can be used to optimize the various components that specify how we optimize or train neural networks. In particular, the optimizer and the loss function have a large impact on training of the neural network. A variety of approaches for meta-learning the neural network optimizer have been proposed (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2019), with VeLO optimizer (Metz et al., 2022) attracting the most attention recently. VeLO achieves excellent performance on a wide variety of tasks and does not need hyperparameter tuning, making it simple to use in practice. The loss function can also be meta-learned, for example to improve the performance (Bechtle et al., 2021), learning speed (Gonzalez and Miikkulainen, 2020), ability to learn with noisy labels (Gao et al., 2021) or better generalization across domains (Gao et al., 2022a).

2.5.6 Dataset Distillation

Dataset distillation is an area that has gained significant traction recently and often relies on meta-learning (Lei and Tao, 2023; Sachdeva and McAuley, 2023). Its main goal is to condense a full-sized dataset into a small synthetic dataset that can be used for training well-performing models. Such small datasets can be useful in various ways, for example for fast neural architecture search or also for memory replay in continual learning (Lei and Tao, 2023; Sachdeva and McAuley, 2023). The distillation is achieved by meta-learning the synthetic data and training with them in the inner loop, while the outer loop evaluates the performance on standard data and updates the synthetic dataset.

Various approaches have been developed, with (Wang et al., 2018b; Bohdal et al., 2020; Zhao et al., 2021) among the first ones. Wang et al. (2018b) have introduced the general problem of dataset distillation, meta-learning the pixel values of the distilled images. Bohdal et al. (2020) have meta-learned synthetic labels for real images (label distillation), showing how it leads to strong dataset distillation performance and is more transferable. Zhao et al. (2021) have utilized gradient-matching as a way to meta-learn high-quality synthetic datasets, leading to significant improvements in performance.

2.5.7 Real-World Applications

Meta-learning has been used in various real-world applications, including medicine, finance, education, communication systems and others. In medicine, it has been used for clinical risk prediction with limited data (Zhang et al., 2019), low-resource medical dialogue generation (Lin et al., 2021) or few-shot medical image classification (Singh et al., 2021). Meta-learning has also found use in finance, for example to do index tracking (Yang and Hospedales, 2023) or to identify fraudulent credit card operations (Zheng et al., 2020b). A particularly interesting real-world use of meta-learning has been within education, where it has been used for providing feedback to students based on only a small number of annotated examples (Wu et al., 2021). The approach has improved upon the quality of feedback provided by teaching assistants and has been deployed to a massive open online course with thousands of students. In the context of communication systems meta-learning has been used for more accurate decoding of noisy signals that were subject to interference during transmission (Park et al., 2020; Li et al., 2021a). Additionally meta-learning has been useful in cyber-security applications (Yang et al., 2023) and for detecting severe weather phenomena in radar images (Kamani et al., 2019).

***

We have discussed both meta-learning algorithms and applications, giving us the background beneficial for understanding our main contributions. Each of the further chapters includes its own related work section to provide further background information relevant to the given chapter. We continue with Part I: *Meta-Learning Algorithms* that introduces our proposed efficient meta-learning algorithms, after which we discuss novel meta-learning applications in Part II: *Meta-Learning Applications*.

Part I

Meta-Learning Algorithms

Chapter 3

EvoGrad: Efficient Gradient-Based Meta-Learning

The content of this chapter corresponds to paper:
EvoGrad: Efficient Gradient-Based Meta-Learning and Hyperparameter Optimization
Ondrej Bohdal, Yongxin Yang, Timothy Hospedales
Advances in Neural Information Processing Systems (NeurIPS), 2021

Gradient-based meta-learning and hyperparameter optimization have seen significant progress recently, enabling practical end-to-end training of neural networks together with many hyperparameters. Nevertheless, existing approaches are relatively expensive as they need to compute second-order derivatives and store a longer computational graph. This cost prevents scaling them to larger network architectures. We present EvoGrad, a new approach to meta-learning that draws upon evolutionary techniques to more efficiently compute hypergradients. EvoGrad estimates hypergradient with respect to hyperparameters without calculating second-order gradients, or storing a longer computational graph, leading to significant improvements in efficiency. We evaluate EvoGrad on three substantial recent meta-learning applications, namely cross-domain few-shot learning with feature-wise transformations, noisy label learning with Meta-Weight-Net and low-resource cross-lingual learning with meta representation transformation. The results show that EvoGrad significantly improves efficiency and enables scaling meta-learning to bigger architectures such as from ResNet10 to ResNet34.

3.1 Introduction

Gradient-based meta-learning and hyperparameter optimization have been of long-standing interest in neural networks and machine learning (Larsen et al., 1996; Maclaurin et al., 2015; Bengio, 2000). Hyperparameters (aka meta-parameters) can take diverse forms, especially under the guise of meta-learning, where there has recently been an explosion of successful applications addressing diverse learning challenges (Hospedales et al., 2021). For example to name just a few: training optimizer initial condition in support of few-shot learning (Finn et al., 2017; Antoniou et al., 2019; Li et al., 2017); training instance-wise weights for cleaning noisy datasets (Shu et al., 2019; Ren et al., 2018b); training loss functions in support of generalization (Li et al., 2019b) and learning speed; and training stochastic regularizers in support of cross-domain robustness (Tseng et al., 2020).

Most of these applications share the property that meta-parameters impact validation loss only indirectly through their effect on model parameters, and so computing validation loss gradients with respect to meta-parameters usually leads to the need to compute second-order derivatives, and store longer computational graphs for backpropagation. This eventually becomes a bottleneck to execution time, and – more severely – to scaling the size of the underlying models, given the practical limitation of GPU memory.

There has been steady progress in the development of diverse practical algorithms for computing validation loss with respect to meta-parameters (Luketina et al., 2016; Lorraine et al., 2020; Maclaurin et al., 2015). Nevertheless they mostly share some form of the aforementioned limitations. In particular, the majority of recent successful practical applications (Shu et al., 2019; Tseng et al., 2020; Li et al., 2019b; Balaji et al., 2018; Bohdal et al., 2020; Liu et al., 2019c; Shan et al., 2020) essentially use some variant of the $T_1 - T_2$ algorithm (Luketina et al., 2016) to estimate the gradient $\frac{\partial \ell_V}{\partial \omega}$ of validation loss w.r.t. hyperparameters ω. This approach computes the gradient online at each step of updating the base model θ, and estimates it as $\frac{\partial \ell_V}{\partial \omega} \approx \frac{\partial \ell_V}{\partial \theta} \frac{\partial^2 \ell_T}{\partial \theta \partial \omega}$, for training loss ℓ_T. As with many alternative estimators, this requires second-order derivatives, and extending the computational graph. Besides the additional computation cost, this limits the size of the base model that can be used in a given GPU, since the memory cost of meta-learning is now multiple times the size of vanilla backpropagation. This in turn prevents the application of meta-learning to problems where large state-of-the-art model architectures are required.

To address this issue, we draw inspiration from evolutionary optimization methods

(Salimans et al., 2017) to develop EvoGrad, a meta-gradient algorithm that requires no higher-order derivatives and as such is significantly faster and lighter than the standard approaches. In particular, we take the novel view of estimating meta-gradients via a putative inner-loop *evolutionary* update to the base model. As this requires no gradients itself, the meta-gradient can then be computed using first-order gradients alone, and without extending the computational graph – leading to efficient hyperparameter updates. Meanwhile for efficient and accurate base model learning, the real inner-loop update can separately be carried out by conventional gradient descent.

Our EvoGrad is a general meta-optimizer applicable to many meta-learning applications, among which we choose three to demonstrate its impact: the LFT model (Tseng et al., 2020) observes that a properly tuned stochastic regularizer can significantly improve cross-domain few-shot learning performance. We show that by training those regularizer parameters with EvoGrad, rather than the standard second-order approach, we can obtain the same improvement in accuracy with significant reduction in time and memory cost. This allows us to scale LFT from the original ResNet10 to ResNet34 within a 12GB GPU. Second, the Meta-Weight-Net (MWN) (Shu et al., 2019) model deals with label noise by meta-learning an auxiliary network that re-weights instance-wise losses to down-weight noisy instances and improve validation loss. We also show that EvoGrad can replicate MWN results with significant cost savings. Third, we demonstrate the benefits of EvoGrad on an application from NLP, in addition to the ones from computer vision: low-resource cross-lingual learning using MetaXL approach (Xia et al., 2021).

To summarize, our main contributions are: 1) We introduce EvoGrad, a novel method for gradient-based meta-learning and hyperparameter optimization that is simple to implement and efficient in time and memory requirements. 2) We evaluate EvoGrad on a variety of illustrative and substantial meta-learning problems, where we demonstrate significant compute and memory benefits compared to standard second-order approaches. 3) In particular, we illustrate that EvoGrad allows us to scale meta-learning to bigger models than was previously possible on a given GPU size, thus bringing meta-learning closer to the state-of-the-art frontier of real applications.

We provide source code for EvoGrad at: `https://github.com/ondrejbohdal/evograd`.

3.2 Related Work

Gradient-based meta-learning solves a bilevel optimization problem where validation loss is optimized with respect to the meta-knowledge by backpropagating through the update of the model on training data and with meta-knowledge. The meta-knowledge updates form an outer loop, around an inner loop of base model updates. The inner loop can run for one (Luketina et al., 2016), few (Shaban et al., 2019; Maclaurin et al., 2015), or many (Lorraine et al., 2020) steps within each outer-loop iteration. Meta-knowledge can take many forms, for example, it can be an initialization of the model weights (Finn et al., 2017), feature-wise transformation layers (Tseng et al., 2020), regularization to improve domain generalization (Balaji et al., 2018) or even a synthetic training set (Wang et al., 2018b; Bohdal et al., 2020). Most substantial practical applications use a one or few-step inner loop for efficiency.

More recently, several methods (Lorraine et al., 2020; Rajeswaran et al., 2019) have utilized Implicit Function Theorem (IFT) to develop new gradient-based meta-learners. These methods use multiple inner-loop steps without the need to backpropagate through whole inner loop, which significantly improves memory efficiency over methods that need keep track of the whole inner loop training process. However, IFT methods assume the model has converged in the inner loop. This makes them unsuited for the majority of practical applications above where training the inner loop to convergence for each hypergradient step is infeasible.

Furthermore, the IFT hypergradient is still more costly compared to one-step $T_1 - T_2$ method. The costs come from the associated overhead with approximating an inverse Hessian of the training data with respect to the model parameters. Note that the Hessian itself does not need to be stored due to the mechanics of reverse-mode differentiation (Griewank, 1993; Baydin et al., 2018). However, this does not eliminate the remaining calculations which still require higher-order gradients that result in backpropagation via longer graphs due to additional gradient nodes. For these reasons, we focus comparison on the more widely used $T_1 - T_2$ strategy which is oriented at single-step inner loops similar to EvoGrad.

Theoretically it is also possible to use hypernetworks (Lorraine and Duvenaud, 2018) to find good hyperparameters in a first-order way. Hypernetworks take hyperparameters as inputs and generate model parameters. However, the approach is not commonly used, likely due to the difficulty of generating well-performing model parameters. We provide experimental results to support this hypothesis in Appendix A.

Meta-learning can be categorized into several groups, depending on the type of meta-knowledge and also if the model is trained from scratch as part of the inner loop (Hospedales et al., 2021). Offline meta-learning approaches train a model from scratch per each update of the meta-knowledge, while online meta-learning approaches train the model and meta-knowledge jointly. As a result, offline meta-learning is extremely expensive (Cubuk et al., 2019; Zoph and Le, 2017) when scaled beyond few-shot learning problems where only a few iterations are sufficient for training (Finn et al., 2017; Antoniou et al., 2019; Li et al., 2017). Therefore most larger-scale problems (Liu et al., 2019a; Jaderberg et al., 2019; Tseng et al., 2020; Xia et al., 2021) use online learning in practice, and this is where we focus our contribution.

The meta-knowledge to learn can take different forms. A particular dichotomy is between the special case where the meta-knowledge corresponds to the base model itself, in the form of an initialization; and the more general cases where it does not. The former initialization meta-learning has been popularized by MAML (Finn et al., 2017), and is widely used in few-shot learning. This can be solved relatively efficiently, for example using a first-order approximation of MAML (Finn et al., 2017), Reptile (Nichol et al., 2018) or minibatch proximal update (Zhou et al., 2019).

On the other hand, there are vastly more cases (Hospedales et al., 2021) where the meta-knowledge is different from the model itself, such as LFT's stochastic regularizer to improve cross-domain generalization (Tseng et al., 2020), MWN's instance-wise loss weighting network for label noise robustness (Shu et al., 2019), a label generation network to improve self-supervised generalization (Liu et al., 2019c), a Feature-Critic loss to improve domain generalization (Li et al., 2019b) and many others. In this more general case, most applications rely on a $T_1 - T_2$-like algorithm, as the efficient approximations specific to MAML do not apply. The ability to significantly improve the efficiency of gradient-based meta-learning would have a large impact as methods like these would directly benefit from it in runtime and energy consumption. More crucially, they could scale to bigger and more state-of-the-art neural network architectures.

3.3 Methodology

3.3.1 Background: Meta-Learning as Bilevel Optimization

We aim to solve a bilevel optimization problem where our goal is to find hyperparameters ω that minimize the validation loss ℓ_V of the model parametrized by θ and trained with

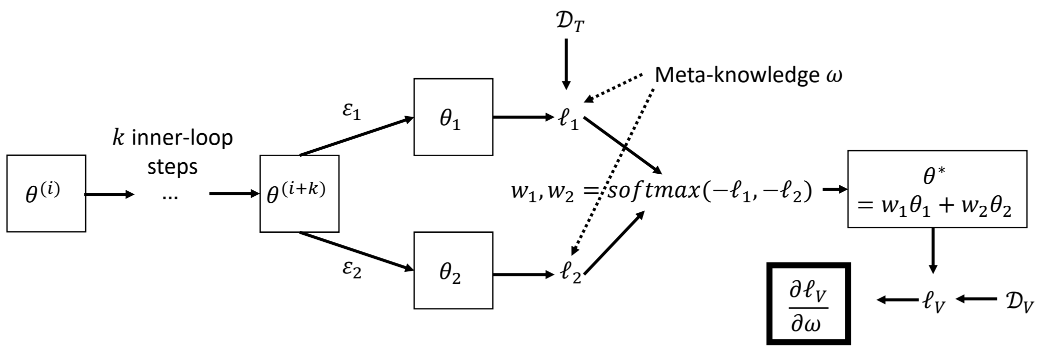


Figure 3.1: Graphical illustration of a single EvoGrad update using $K = 2$ model copies.

loss ℓ_T and ω:

$$\omega^* = \arg\min_{\omega} \ell_V^*(\omega), \text{ where } \ell_V^*(\omega) = \ell_V(\omega, \theta^*(\omega)) \text{ and } \theta^*(\omega) = \arg\min_{\theta} \ell_T(\omega, \theta). \tag{3.1}$$

In order to meta-learn the value of ω using gradient-based methods, we need to calculate the hypergradient $\frac{\partial \ell_V}{\partial \omega}$. We can expand its calculation as follows:

$$\frac{\partial \ell_V^*(\omega)}{\partial \omega} = \frac{\partial \ell_V(\omega, \theta^*(\omega))}{\partial \omega} + \frac{\partial \ell_V(\omega, \theta^*(\omega))}{\partial \theta^*(\omega)} \frac{\partial \theta^*(\omega)}{\partial \omega}. \tag{3.2}$$

In meta-learning and hyperparameter optimization more broadly, the direct term $\frac{\partial \ell_V(\omega, \theta^*(\omega))}{\partial \omega}$ is typically zero because the hyperparameter does not directly influence the value of the validation loss – it influences it via the impact on the model weights θ. However, the model weights θ are themselves trained using gradient optimization, which gives rise to higher-order derivatives. We propose a variation on this step where the update of the model weights is inspired by evolutionary methods, allowing us to eliminate the need for higher-order derivatives. We consider the setting where the hypergradient of hyperparameter ω is estimated online (Luketina et al., 2016) together with updating the base model θ, as this is the most widely used setting in substantial practical applications (Shu et al., 2019; Tseng et al., 2020; Li et al., 2019b; Balaji et al., 2018; Bohdal et al., 2020; Shan et al., 2020; Liu et al., 2019a; Xia et al., 2021).

3.3.2 The EvoGrad Update

Given the current model parameters $\theta \in \mathbb{R}^M$, hyperparameters $\omega \in \mathbb{R}^N$, training loss ℓ_T and validation loss ℓ_V, we aim to estimate $\frac{\partial \ell_V}{\partial \omega}$ for efficient gradient-based hyperparameter learning. The key idea is – solely for the purpose of hypergradient estimation – to consider a simple evolutionary rather than gradient-based inner-loop step on θ.

Evolutionary Inner Step First, we sample random perturbations $\varepsilon \in \mathbb{R}^M \sim \mathcal{N}(0, \sigma I)$, and apply them to θ. Sampling K perturbations, we can create a population of K variants $\{\theta_k\}_{k=1}^K$ of the current model as $\theta_k = \theta + \varepsilon_k$. We can now compute the training losses $\{\ell_k\}_{k=1}^K$ for each of the K models, $\ell_k = f(\mathcal{D}_T|\theta_k, \omega)$ using the current minibatch $\mathcal{D}_T$ drawn from the training set. Given these loss values, we can calculate the weights (sometimes called fitness) of the population of candidate models as

$$w_1, w_2, \ldots, w_K = \text{softmax}([-\ell_1, -\ell_2, \ldots, -\ell_K]/\tau), \tag{3.3}$$

where τ is a temperature parameter that rescales the losses to control the scale of weight variability.

Given the weights $\{w_k\}_{k=1}^K$, we complete the current step of evolutionary learning by updating the model parameters via the affine combination

$$\theta^* = w_1\theta_1 + w_2\theta_2 + \cdots + w_K\theta_K. \tag{3.4}$$

Computing the Hypergradient We now evaluate the updated model θ^* for a minibatch from the validation set $\mathcal{D}_V$ and take gradient of the validation loss $\ell_V = f(\mathcal{D}_V|\theta^*)$ w.r.t. the hyperparameter:

$$\frac{\partial \ell_V}{\partial \omega} = \frac{\partial f(\mathcal{D}_V|\theta^*)}{\partial \omega} \tag{3.5}$$

One can easily verify that the computation in Eq. 3.5 does not involve second-order gradients as no first-order gradients were used in the inner loop. This is in contrast to the typical approach (Luketina et al., 2016; Maclaurin et al., 2015) of applying gradient-based updates in the inner loop and differentiating through it (in either forward-mode or reverse-mode), or even applying the implicit function theorem (IFT) (Lorraine et al., 2020), all of which trigger higher-order gradients and an extended computation graph.

Algorithm Flow In practice we follow the flow of $T_1 - T_2$ (Luketina et al., 2016) used by many substantive applications (Tseng et al., 2020; Shu et al., 2019; Balaji et al., 2018; Liu et al., 2019a; Xia et al., 2021). We take alternating steps on θ using the exact gradient $\frac{\partial \ell_T}{\partial \theta}$, and on ω using the hypergradient $\frac{\partial \ell_V}{\partial \omega}$, which in EvoGrad is estimated as in Eq. 3.5.

3.3.3 EvoGrad Hypergradient as a Random Projection

To understand EvoGrad, observe that the hyper-gradient in Eq. 3.5 expands as

$$\frac{\partial \ell_V}{\partial \omega} = \frac{\partial \ell_V}{\partial \theta^*}\frac{\partial \theta^*}{\partial \omega} = \frac{\partial \ell_V}{\partial \theta^*}\mathcal{E}\frac{\partial w}{\partial \omega} = \frac{\partial \ell_V}{\partial \theta^*}\mathcal{E}\frac{\partial w}{\partial \ell}\frac{\partial \ell}{\partial \omega}, \tag{3.6}$$

where $\mathcal{E} = [\varepsilon_1, \varepsilon_2, \dots, \varepsilon_K]$ is the $M \times K$ matrix formed by stacking ε_k's as columns, w is the K-dimensional ($w = [w_1, w_2, \dots, w_K]$) vector of candidate model weights, and $\ell = [\ell_1, \ell_2, \dots, \ell_K]$ is the K-dimensional vector of candidate model losses.

Recall that $\mathcal{E}$ is a random matrix, so the operation $\frac{\partial \ell_V}{\partial \theta^*}\mathcal{E}$ can be understood as randomly projecting the M-dimensional validation loss' gradient to a new low-dimensional space of dimension $K \ll M$. Alternatively, we can interpret the update as factorising the model-parameter-to-hyperparameter derivative $\frac{\partial \theta^*}{\partial \omega}$ (sized $M \times N$) into two much smaller matrices $\mathcal{E}$ and $\frac{\partial w}{\partial \omega}$ of size $M \times K$ and $K \times N$.

In terms of implementation, $\frac{\partial \ell_V}{\partial \theta^*}$ is obtained by backpropagation and $\mathcal{E}$ is sampled on the fly. The term $\frac{\partial w}{\partial \omega} = \frac{\partial w}{\partial \ell}\frac{\partial \ell}{\partial \omega}$ is computed by the softmax-to-logit derivative ($K \times K$) and the derivative of the K candidate models training losses w.r.t. hyperparameters. It is noteworthy that the K elements of $\frac{\partial \ell}{\partial \omega}$ are completely independent, and can be computed in parallel where multiple GPUs are available.

3.3.4 Comparison to Other Methods

We compare EvoGrad to the most related and widely-used alternative $T_1 - T_2$ (Luketina et al., 2016) in Table 3.1. $T_1 - T_2$ requires higher-order gradients and associated longer computational graphs – due to the need to backpropagate through gradient nodes. This leads to increased memory and time cost compared to vanilla backpropagation. In contrast, EvoGrad requires no higher-order gradients, no large matrices, and no substantial expansion of the computational graph.

Table 3.1: Comparison of hypergradient approximations of $T_1 - T_2$ and EvoGrad.

Method	Hypergradient approximation
$T_1 - T_2$ (Luketina et al., 2016)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times I \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T}$
EvoGrad (ours)	$\frac{\partial \ell_V}{\partial \omega} + \frac{\partial \ell_V}{\partial \theta} \times \mathcal{E}\frac{\partial w}{\partial \ell}\frac{\partial \ell}{\partial \omega} = \frac{\partial \ell_V}{\partial \omega} + \frac{\partial \ell_V}{\partial \theta} \times \mathcal{E}\frac{\partial\, \mathrm{softmax}(-\ell)}{\partial \omega}$

We analyse the asymptotic big-O time and memory requirements of EvoGrad vs $T_1 - T_2$ in Table 3.2. The dominant cost in terms of both memory and time is the cost of backpropagation. Backpropagation is significantly more expensive than forward propagation because forward propagation does not need to store all intermediate variables in memory (Rajeswaran et al., 2019; Griewank, 1993). Note that even if EvoGrad keeps multiple copies of the model weights in memory, this cost is small compared to the cost of backpropagation, and the latter is done with only *one* set of weights θ^*. We remark that our main empirical results are obtained with only $K = 2$ models, so we can safely ignore this in our asymptotic analysis.

In addition, we elaborate on how higher-order gradients contribute to increased memory and time costs. Results from computing the first-order gradients are added into the computation graph as new nodes in the graph so that we can calculate the higher-order gradients. When calculating the higher-order gradients, we backpropagate through this longer computational graph, which directly increases the memory and time costs. The current techniques (Luketina et al., 2016) rely on longer computational graphs, while EvoGrad significantly shortens the graph and reduces memory cost by avoiding this step. This consideration is not visible in the big-O analysis, but contributes to improved efficiency.

Table 3.2: Comparison of asymptotic memory and operation requirements of EvoGrad and $T_1 - T_2$ meta-learning strategies. P is the number of model parameters, H is the number of hyperparameters. $K \ll H$ is the number of model copies in EvoGrad. Note this is a first-principles analysis, so the time requirements are different when using e.g. reverse-mode backpropagation that uses parallelization.

Method	Time requirements	Memory requirements
$T_1 - T_2$ (Luketina et al., 2016)	$O(PH)$	$O(P+H)$
EvoGrad (ours)	$O(KP+H)$	$O(P+H)$

3.4 Experiments

We first consider two simple problems: 1) a 1-dimensional problem where we try to find the minimum of a function, and 2) meta-learning a feature-transformer to find the rotation that correctly aligns images whose training and validation sets differ in rotation. This serves as a proof-of-concept problem to show our method is capable of meta-learning suitable hyperparameters. We then consider three real problems where meta-learning has been used to solve different learning challenges. We show that EvoGrad makes a significant impact in terms of reducing the memory and time costs (while keeping the accuracy improvements brought by meta-learning): 3) Cross-domain few-shot classification via learned feature-wise transformation (Tseng et al., 2020), 4) Meta-Weight-Net: learning an explicit mapping for sample weighting (Shu et al., 2019), 5) MetaXL: meta representation transformation for low-resource cross-lingual learning (Xia et al., 2021). We provide a brief overview of each problem, together with evaluation and analysis. Further details and experimental settings are described in Appendix A.

3.4.1 Illustration Using a 1-Dimensional Problem

In this problem we minimize a validation loss function $f_V(x) = (x-0.5)^2$ where parameter x is optimized using SGD with training loss function $f_T(x) = (x-1)^2 + \omega\|x\|_2^2$ that includes a meta-parameter ω. A closed-form solution for the hypergradient is available and is equal to $g(\omega) = (\omega-1)/(\omega+1)^3$, which allows us to compare EvoGrad against the ground-truth gradient.

Our first analysis studies the estimated EvoGrad hypergradient for a grid of ω values between 0 and 2. For each value of ω we show the mean and standard deviation of the estimated $\partial f_V/\partial\omega$ over 100 repetitions (with random choice of x). We use temperature $\tau = 0.5$, $\epsilon \in \mathbb{R} \sim \mathcal{N}(0,1)$ and consider between 2 and 100 models copies in the population. The results in Figure 3.2 show that EvoGrad estimates have a similar trend to the ground-truth gradient, even if the EvoGrad estimates are noisy. The level of noise decreases with more models in the population, but the correct trend is visible even if we use only 2 models.

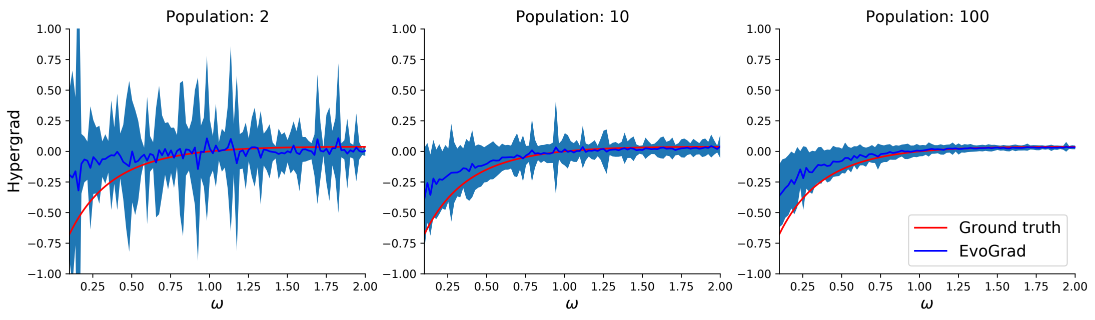


Figure 3.2: Comparison of the hypergradient $\partial f_V/\partial\omega$ estimated by EvoGrad vs the ground-truth.

Our second analysis studies the trajectories that parameters x, ω follow if they are both optimized online using SGD with learning rate of 0.1 for 5 steps, starting from five different positions (circles). The hypergradients are either estimated using EvoGrad or directly using the ground-truth formula. Figure 3.3 shows that the trajectories of both variations are similar, and they become more similar as we use more models in the population. In all cases the parameters converge towards the lightly-coloured region where the validation loss is the lowest at $x = 0.5$.

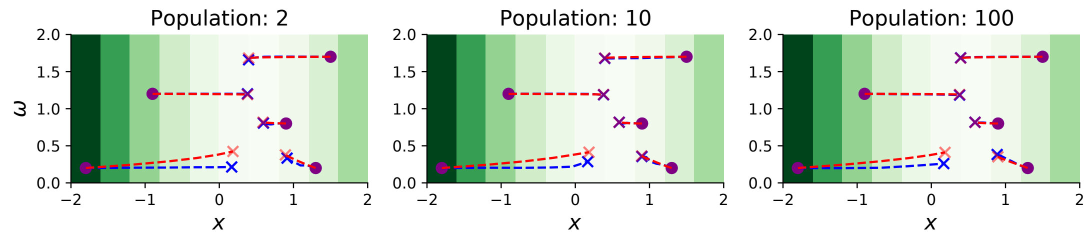


Figure 3.3: Trajectories of parameters x, ω when following $\partial f_T / \partial x$ and $\partial f_V / \partial \omega$ using SGD for 5 random starting positions. Comparison of trajectories using EvoGrad estimated (blue) or ground-truth (red) hypergradient. The initial position is marked with a circle, and the final position after 5 steps is marked with a cross. The shading is validation loss $f_V(x)$.

3.4.2 Rotation Transformation

In this task we work with MNIST images (LeCun et al., 1998), and assume that the validation and test sets are rotated by 30° compared to the conventionally oriented training images. Clearly, directly training a model and applying it will lead to low performance. We therefore assume meta-knowledge in the form of a hidden rotation. The rotation transformation is applied to the training images before learning, and should itself be meta-learned by the validation loss obtained by the CNN trained on the rotated training set. Thus solving the meta-learning problem should result in a 30° rotation, and a base CNN that generalizes to the rotated validation set.

The problem is framed as online meta-learning where each update of the base model is followed by a meta-parameter update using EvoGrad. We use EvoGrad with 2 model copies, temperature $\tau = 0.05$ and $\sigma = 0.001$ for $\epsilon \sim \sigma \text{sign}(\mathcal{N}(0, I))$. Our LeNet (LeCun et al., 1989) base model is trained for 5 epochs.

We repeat the experiments 5 times and show a comparison of the results in Table 3.3. A baseline model achieves $98.40 \pm 0.07\%$ accuracy if the test images are not rotated, but its accuracy drops to $81.79 \pm 0.64\%$ if the same images are rotated by 30°. A model trained with EvoGrad and the rotation transformer is able to accurately classify rotated images, with a similar accuracy as the baseline model can classify unrotated images. This confirms we can successfully optimize hyperparameters with EvoGrad. The meta-learned rotation is also close to the true value.

Table 3.3: Rotation transformation learning. The goal is to accurately classify MNIST test images rotated by 30° degrees compared to the training set orientation. Test accuracies (%) of a baseline model, and one whose training set has been rotated by the EvoGrad's meta-learned rotation, and associated EvoGrad rotation estimate (°). Accuracy for rotation matched train/test sets is 98.40%.

True Rotation	Baseline Acc.	EvoGrad Acc.	EvoGrad Rotation Est.
30°	81.79 ± 0.64	98.11 ± 0.32	$28.47° \pm 5.23°$

3.4.3 Cross-Domain Few-Shot Classification via Learned Feature-Wise Transformation

As the next task we consider cross-domain few-shot classification (CD-FSL). CD-FSL is considered an important and highly challenging problem at the forefront of computer vision. The state-of-the-art approach learned feature-wise transformation (LFT) (Tseng et al., 2020) aims to meta-learn stochastic feature-wise transformation layers that regularize metric-based few-shot learners to improve their few-shot learning generalization in cross-domain conditions. The method includes two key steps: 1) updating the model with the meta-parameters on a pseudo-seen task and 2) updating the meta-parameters by evaluating the model on a pseudo-unseen task by backpropagating through the first step. As feature-wise transformation is not directly used for the pseudo-unseen task, this leads to higher-order gradients. Note that the problem itself is memory-intensive because we work with larger images of size 224×224 within episodic learning tasks. As a result, a significantly more efficient meta-learning approach could allow us to scale from the ResNet10 model used in the paper to a larger model.

We experiment with the LFT-RelationNet (Sung et al., 2018) metric-based few-shot learner and consider the exact same experiment settings as (Tseng et al., 2020) using the official PyTorch implementation associated with the paper. LFT introduces 3712 hyperparameters to train for ResNet10, and 9344 for ResNet34. All our experiments are conducted on Titan X GPUs with 12GB of memory using $K = 2$ for EvoGrad.

Table 3.4 shows the baseline performance of vanilla unregularised ResNet (-), manually tuned FT layers (FT), FT layers meta-learned by second-order gradient (LFT) and by EvoGrad. The results show that EvoGrad matches the accuracy of the original LFT approach, leading to clear accuracy improvements over training with no feature-wise transformation or training with fixed feature-wise parameters selected manually. At the same time EvoGrad is significantly more efficient in terms of the memory and time costs as shown in Figure 3.4. The memory improvements from EvoGrad allow us to scale the base feature extractor to ResNet34 within the standard 12GB GPU. The original LFT with its $T_1 - T_2$ style second-order algorithm cannot be extended in the available memory if we keep the same settings of the few-shot learning tasks. Thus, we are able to improve state-of-the-art accuracy on both 5-way 1 and 5-shot tasks. For ResNet34, we include baselines without any feature-wise transformation and with manually chosen feature-wise transformation to confirm the benefit of meta-learning.

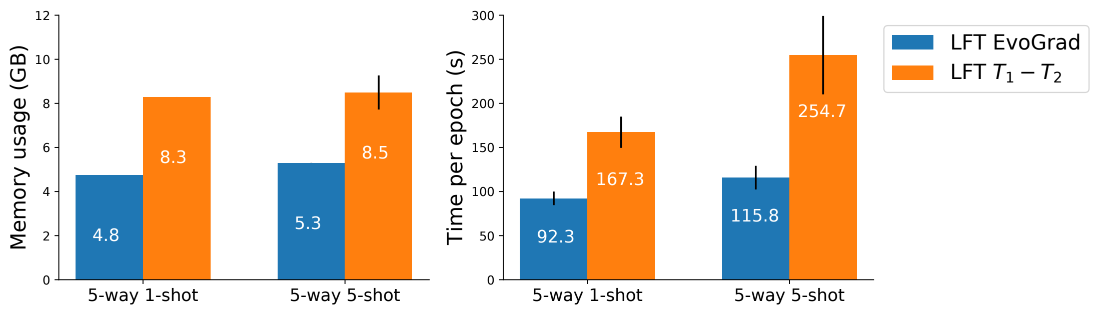


Figure 3.4: Cross-domain few-shot learning with LFT (Tseng et al., 2020): analysis of memory and time efficiency of EvoGrad vs standard second-order $T_1 - T_2$ approach. Mean and standard deviation reported across experiments with different test datasets. EvoGrad is significantly more efficient in terms of both memory usage and time per epoch.

3.4.4 Label Noise with Meta-Weight-Net

We consider a further highly practical real problem where online meta-learning has led to significant improvements – learning from noisy labelled data. The Meta-Weight-Net framework trains an auxiliary neural network that performs instance-wise loss re-weighting on the training set (Shu et al., 2019). The base model is updated using the sum of weighted instance-wise losses for noisy data, while the Meta-Weight-Net

Table 3.4: RelationNet test accuracies (%) and 95% confidence intervals across test tasks on various unseen datasets. LFT EvoGrad can scale to ResNet34 on all tasks within 12GB GPU memory, while vanilla second-order LFT $T_1 - T_2$ cannot. We also report the results of our own rerun of the LFT approach using the official code – denoted as *our run*. EvoGrad can clearly match the accuracies obtained by the original approach that uses $T_1 - T_2$.

	Model	Approach	CUB	Cars	Places	Plantae
5-way 1-shot	ResNet10	-	44.33 ± 0.59	29.53 ± 0.45	47.76 ± 0.63	33.76 ± 0.52
		FT	44.67 ± 0.58	30.38 ± 0.47	48.40 ± 0.64	35.40 ± 0.53
		LFT $T_1 - T_2$	48.38 ± 0.63	32.21 ± 0.51	50.74 ± 0.66	35.00 ± 0.52
		LFT $T_1 - T_2$ (our run)	46.03 ± 0.60	31.50 ± 0.49	49.29 ± 0.65	36.34 ± 0.59
		LFT EvoGrad	47.39 ± 0.61	32.51 ± 0.56	50.70 ± 0.66	36.00 ± 0.56
	ResNet34	-	45.61 ± 0.59	29.54 ± 0.46	48.87 ± 0.65	35.03 ± 0.54
		FT	45.15 ± 0.59	30.28 ± 0.44	49.96 ± 0.66	35.69 ± 0.54
		LFT EvoGrad	45.97 ± 0.60	33.21 ± 0.54	50.76 ± 0.67	38.23 ± 0.58
5-way 5-shot	ResNet10	-	62.13 ± 0.74	40.64 ± 0.54	64.34 ± 0.57	46.29 ± 0.56
		FT	63.64 ± 0.77	42.24 ± 0.57	65.42 ± 0.58	47.81 ± 0.51
		LFT $T_1 - T_2$	64.99 ± 0.54	43.44 ± 0.59	67.35 ± 0.54	50.39 ± 0.52
		LFT $T_1 - T_2$ (our run)	65.94 ± 0.56	43.88 ± 0.56	65.57 ± 0.57	51.43 ± 0.55
		LFT EvoGrad	64.63 ± 0.56	42.64 ± 0.58	66.54 ± 0.57	52.92 ± 0.57
	ResNet34	-	63.33 ± 0.59	40.50 ± 0.55	64.94 ± 0.56	50.20 ± 0.55
		FT	62.48 ± 0.56	41.06 ± 0.52	64.39 ± 0.57	50.08 ± 0.55
		LFT EvoGrad	66.40 ± 0.56	44.25 ± 0.55	67.23 ± 0.56	52.47 ± 0.56

itself is updated by evaluating the updated model on clean validation data and by backpropagating through the model update. We use the official implementation of the approach (Shu et al., 2019) and follow the same experimental settings, using $K = 2$ for EvoGrad.

Our results in Table 3.5 confirm we replicate the benefits of training with Meta-Weight-Net, clearly surpassing the accuracy of the baseline when there is label noise. We also note that EvoGrad can improve the accuracy over the $T_1 - T_2$-based approach because the two approaches are distinct and provide different estimates of the true hypergradient. Figure 3.5 shows that our method leads to significant improvements in memory and time costs (over half of the memory is saved and the runtime is improved by about a third).

Table 3.5: Test accuracies (%) for Meta-Weight-Net label noise experiments with ResNet-32 – means and standard deviations across 5 repetitions for the original second-order algorithm vs EvoGrad. EvoGrad is able to match or even exceed the accuracies obtained by the original MWN approach.

Dataset	Noise rate	Baseline	MWN $T_1 - T_2$	MWN $T_1 - T_2$ (our run)	MWN EvoGrad
CIFAR-10	0%	92.89 ± 0.32	92.04 ± 0.15	91.10 ± 0.19	92.02 ± 0.31
	20%	76.83 ± 2.30	90.33 ± 0.61	89.31 ± 0.40	89.86 ± 0.64
	40%	70.77 ± 2.31	87.54 ± 0.23	85.90 ± 0.45	87.74 ± 0.54
CIFAR-100	0%	70.50 ± 0.12	70.11 ± 0.33	68.42 ± 0.36	69.16 ± 0.49
	20%	50.86 ± 0.27	64.22 ± 0.28	63.43 ± 0.43	64.05 ± 0.63
	40%	43.01 ± 1.16	58.64 ± 0.47	56.54 ± 0.90	57.44 ± 1.25

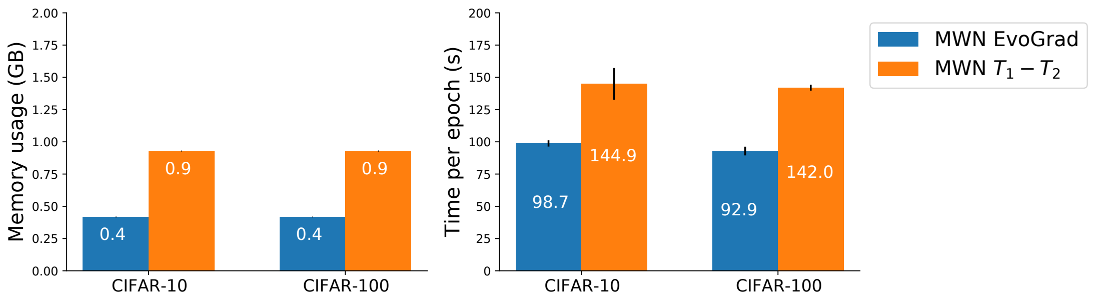


Figure 3.5: Analysis of memory and time cost of MWN EvoGrad vs the original second-order MWN, showing significant efficiency improvements by EvoGrad. Mean and standard deviation is reported across 5 repetitions of 40% label noise problem.

3.4.5 Low-Resource Cross-Lingual Learning with MetaXL

The previous two real applications of meta-learning considered computer vision problems. To highlight EvoGrad is a general method that can make an impact in any domain, we also demonstrate its benefits on a meta-learning application from NLP. More specifically, we use EvoGrad for MetaXL (Xia et al., 2021), which meta-learns meta representation transformation to better transfer from source languages to low-resource target languages.

We have selected the named entity recognition (NER) task with English source language (WikiAnn dataset (Pan et al., 2017)), which is one of the key experiments in the MetaXL paper (Xia et al., 2021). Table 3.6 shows EvoGrad matches and in fact surpasses the average test F1 score of MetaXL with the original $T_1 - T_2$ meta-learning method. Figure 3.6 shows EvoGrad significantly improves both memory and time consumption compared to MetaXL $T_1 - T_2$. Overall these results confirm EvoGrad is suitable for meta-learning in various domains, including both computer vision and NLP.

Table 3.6: Test F1 score in % for named entity recognition task. English source language. The first two rows are taken from the MetaXL paper, while our own runs are in the following rows. EvoGrad clearly matches and even surpasses the performance of $T_1 - T_2$ baseline. Joint-training (JT) represents a simple non-meta-learning baseline.

Method	qu	cdo	ilo	xmf	mhr	mi	tk	gn	Average
JT	66.10	55.83	80.77	69.32	71.11	82.29	61.61	65.44	69.06
MetaXL $T_1 - T_2$	68.67	55.97	77.57	73.73	68.16	88.56	66.99	69.37	71.13
JT (our run)	59.75	49.19	79.43	68.85	68.42	89.94	61.90	69.44	68.37
MetaXL $T_1 - T_2$ (our run)	65.29	56.33	76.50	67.24	71.17	89.41	66.67	64.11	69.59
MetaXL EvoGrad	71.00	57.02	85.99	70.40	65.45	88.12	66.97	70.91	71.98

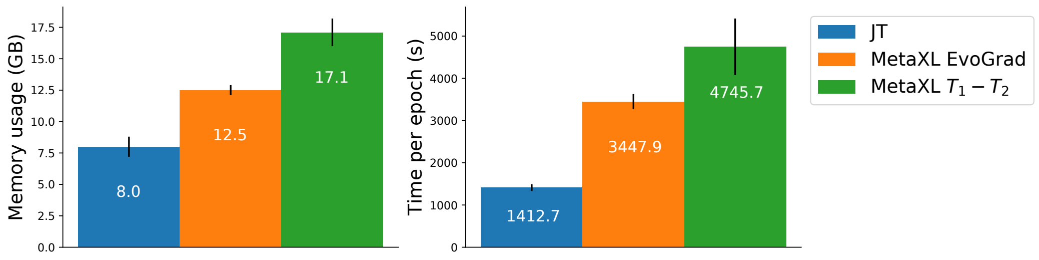


Figure 3.6: Analysis of memory and time cost of MetaXL EvoGrad vs the original second-order MetaXL, in the context of a simple joint-training (JT) baseline. EvoGrad consumes significantly less memory than $T_1 - T_2$ and is faster. Mean and standard deviation is calculated over the 8 different target languages.

3.4.6 Scalability Analysis

We use the Meta-Weight-Net benchmark to study how the number of model parameters affects the memory usage and training time of EvoGrad, comparing it to the standard second-order $T_1 - T_2$ approach. We vary model size by changing the number of filters in the original ResNet32 model, multiplying the filter number $\times 1, \ldots, \times 5$. The smallest model had around 0.5M parameters and the largest one around 11M parameters.

The results in Figure 3.7 show our EvoGrad leads to significantly lower training time and memory usage, and that the margin over the standard second-order optimizer grows as the model becomes larger. Further, we have analysed the impact of modifying the number of hyperparameters – from 300 up to 30,000. The impact on memory and time was negligible, and both remained roughly constant, which is caused by the main model being significantly larger. It is also because of the fact that reverse-mode differentiation costs scale with the number of model parameters rather than hyperparameters (Micaelli and Storkey, 2020) – recall that backpropagation is the main driver of memory and time costs (Rajeswaran et al., 2019). Moreover, we have done experiments that varied the number of model copies in EvoGrad. The results showed the training time per epoch increased slightly, while the memory costs remained similar.

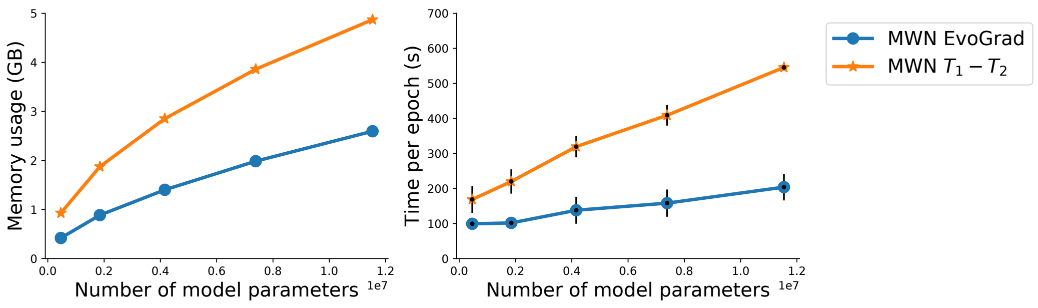


Figure 3.7: Memory and time scaling of MWN EvoGrad vs original second-order Meta-Weight-Net. Efficiency margins of EvoGrad are larger for larger models.

3.5 Discussion

Similar to many other gradient-based meta-learning methods, our method is greedy as it does not consider the whole training process when updating the hyperparameters. However, this greediness allows the method to be used in larger-scale settings where we train the hyperparameters and the model jointly. Further, our method approximates the hypergradient stochastically. While results were good for the suite of problems considered here using only $K = 2$, the gradient estimates may be too noisy in other applications. Alternatively, it may necessitate using a larger model population (Figure 3.2). While as we observed in Section 3.3.3 the candidate models can be trivially parallelized to scale population size, this still imposes a larger energy cost (Schwartz et al., 2019). Another limitation is that similarly to IFT-based estimators (Lorraine et al., 2020), EvoGrad is not suitable for optimizing learner hyperparameters such as learning rate. Currently we have used the simplest possible evolutionary update in the inner loop, and upgrading EvoGrad to a state-of-the-art evolutionary strategy may lead to better gradient estimates and improve results further.

3.6 Further Developments

Since publishing EvoGrad, there have been various developments in the area of gradient-based meta-learning (GBML). In multiple cases, EvoGrad has been utilized or referenced as an efficient meta-learning method. For example, our EvoGrad approach has been used by Prakash et al. (2023) to obtain hypergradients as part of a newly proposed first-order Bayesian Optimization algorithm. EvoGrad has also been mentioned in surveys focused on bilevel optimization and few-shot learning (Chen et al., 2022; Song et al., 2023).

Additionally there have been a number of papers where EvoGrad was included in the related work, for example (Wang et al., 2022b; Chen et al., 2023a,b; Wang et al., 2023a, 2024). These papers cover a diverse set of problems, showing EvoGrad is practically relevant in many scenarios. More specifically, Wang et al. (2022b) develop a method that meta-learns a model initialization from pre-trained models without data. Chen et al. (2023a) propose a method for biological sequence design, while Chen et al. (2023b) propose a meta-weighting strategy for long-tailed dynamic scene graph generation. Further, Wang et al. (2023a) also take inspiration from evolutionary algorithms and propose a meta-learning method for reinforcement learning. Wang et al. (2024) introduce a meta-learning framework for diagnosis of diabetic retinopathy. The diversity of these use-cases highlights the wide relevance of our EvoGrad approach.

The broader field of GBML has also seen a variety of further developments. A number of libraries have been released to make meta-learning more scalable, especially by supporting distributed training across multiple GPUs. These libraries include Betty (Choe et al., 2023b) and TorchOpt (Ren et al., 2023). SAMA approach (Choe et al., 2023a) is also one of the most recent GBML approaches that includes support for distributed training, and it avoids the computation of second-order gradients by utilizing central finite difference, which was part of (Liu et al., 2019a). Other recent first-order bilevel optimization methods include (Liu et al., 2022; Kwon et al., 2023). Implicit GBML methods have also seen improvements, most notably by utilizing the Nyström method, leading to more accurate and scalable implicit differentiation (Hataya and Yamada, 2023). New applications of GBML have also been proposed, for example optimization of data augmentation (Hataya et al., 2022) and task-aware continued pre-training (Dery et al., 2022).

3.7 Conclusions

We have proposed a new efficient method for meta-learning that allows us to scale gradient-based meta-learning to bigger models and problems. We have evaluated the method on a variety of problems, most notably meta-learning feature-wise transformation layers, training with noisy labels using Meta-Weight-Net, and meta-learning meta representation transformation for low-resource cross-lingual learning. In all cases we have shown significant time and memory efficiency improvements, while achieving similar or better performance compared to the existing meta-learning methods.

***

EvoGrad gives us a way to efficiently optimize a large number of meta-parameters, under the assumption they are differentiable and influence the loss function. It is a gradient-based method, but gradient-free approaches for meta-learning also exist and are widely used. Multi-fidelity hyperparameter optimization methods are one popular example. While they are suitable only for optimizing a smaller number of meta-parameters, they can also optimize meta-parameters that do not directly influence the loss function (e.g. learning rate) or are not differentiable. This means EvoGrad would not be suitable for many of the cases for which multi-fidelity methods are often used. The efficiency of multi-fidelity methods can also be significantly improved, which is our focus in the next chapter that introduces our PASHA algorithm.

Chapter 4

PASHA: Efficient Hyperparameter Optimization

The content of this chapter corresponds to paper:
PASHA: Efficient HPO and NAS with Progressive Resource Allocation
Ondrej Bohdal, Lukas Balles, Martin Wistuba, Beyza Ermis, Cedric Archambeau, Giovanni Zappella
The Eleventh International Conference on Learning Representations (ICLR), 2023

Hyperparameter optimization (HPO) and neural architecture search (NAS) are methods of choice to obtain the best-in-class machine learning models, but in practice they can be costly to run. When models are trained on large datasets, tuning them with HPO or NAS rapidly becomes prohibitively expensive for practitioners, even when efficient multi-fidelity methods are employed. We propose an approach to tackle the challenge of tuning machine learning models trained on large datasets with limited computational resources. Our approach, named PASHA, extends ASHA and is able to dynamically allocate maximum resources for the tuning procedure depending on the need. The experimental comparison shows that PASHA identifies well-performing hyperparameter configurations and architectures while consuming significantly fewer computational resources than ASHA.

4.1 Introduction

Hyperparameter optimization (HPO) and neural architecture search (NAS) yield state-of-the-art models, but often are a very costly endeavor, especially when working with large datasets and models. For example, using the results of (Sharir et al., 2020) we can estimate that evaluating 50 configurations for a 340-million-parameter BERT model (Devlin et al., 2019) on the 15GB Wikipedia and Book corpora would cost around $500,000. To make HPO and NAS more efficient, researchers explored how we can learn from cheaper evaluations (e.g. on a subset of the data) to later allocate more resources only to promising configurations. This created a family of methods often described as multi-fidelity methods. Two well-known algorithms in this family are Successive Halving (SH) (Jamieson and Talwalkar, 2016; Karnin et al., 2013) and Hyperband (HB) (Li et al., 2018b).

Multi-fidelity methods significantly lower the cost of the tuning. Li et al. (2018b) reported speedups up to 30x compared to standard Bayesian Optimization (BO) and up to 70x compared to random search. Unfortunately, the cost of current multi-fidelity methods is still too high for many practitioners, also because of the large datasets used for training the models. As a workaround, they need to design heuristics which can select a set of hyperparameters or an architecture with a cost comparable to training a single configuration, for example, by training the model with multiple configurations for a single epoch and then selecting the best-performing candidate.

On one hand, such heuristics lack robustness and need to be adapted to the specific use-cases in order to provide good results. On the other hand, they build on an extensive amount of practical experience suggesting that multi-fidelity methods are often not sufficiently aggressive in leveraging early performance measurements and that identifying the best performing set of hyperparameters (or the best architecture) does not require training a model until convergence. For example, Bornschein et al. (2020) show that it is possible to find the best hyperparameter – number of channels in ResNet-101 architecture (He et al., 2016) for ImageNet (Deng et al., 2009) – using only one tenth of the data. However, it is not known beforehand that one tenth of data is sufficient for the given task.

Our aim is to design a method that consumes fewer resources than standard multi-fidelity algorithms such as Hyperband (Li et al., 2018b) or ASHA (Li et al., 2020a), and yet is able to identify configurations that produce models with a similar predictive performance after full retraining from scratch. Models are commonly retrained on a combination of training and validation sets to obtain the best performance after optimizing the hyperparameters. To achieve the speedup, we propose a variant of ASHA, called Progressive ASHA (PASHA), that starts with a small amount of initial maximum resources and gradually increases them as needed. ASHA in contrast has a fixed amount of maximum resources, which is a hyperparameter defined by the user and is difficult to select. Our empirical evaluation shows PASHA can save a significant amount of resources while finding similarly well-performing configurations as conventional ASHA, reducing the entry barrier to do HPO and NAS.

To summarize, our contributions are as follows: 1) We introduce a new approach called PASHA that dynamically selects the amount of maximum resources to allocate for HPO or NAS (up to a certain budget), 2) Our empirical evaluation shows the approach significantly speeds up HPO and NAS without sacrificing the performance, and 3) We show the approach can be successfully combined with sample-efficient strategies based on Bayesian Optimization, highlighting the generality of our approach.

Our implementation is based on the Syne Tune library (Salinas et al., 2022) and PASHA is included there. Code with additional explanations and notebooks for PASHA is provided at `https://github.com/ondrejbohdal/pasha`.

4.2 Related Work

Real-world machine learning systems often rely on a large number of hyperparameters and require testing many combinations to identify suitable values. This makes data-inefficient techniques such as Grid Search or Random Search (Bergstra and Bengio, 2012) very expensive in most practical scenarios. Various approaches have been proposed to find good parameters more quickly, and they can be classified into two main families: 1) Bayesian Optimization: evaluates the most promising configurations by modelling their performance. The methods are sample-efficient but often designed for environments with limited amount of parallelism; 2) Multi-fidelity: sequentially allocates more resources to configurations with better performance and allows high level of parallelism during the tuning. Multi-fidelity methods have typically been faster when run at scale and will be the focus of this work. Ideas from these two families can also be combined together, for example as done in BOHB by Falkner et al. (2018), and we will test a similar method in our experiments.

Successive Halving (SH) (Karnin et al., 2013; Jamieson and Talwalkar, 2016) is conceptually the simplest multi-fidelity method. Its key idea is to run all configurations using a small amount of resources and then successively promote only a fraction of the most promising configurations to be trained using more resources. Another popular multi-fidelity method, called Hyperband (Li et al., 2018b), performs SH with different early schedules and number of candidate configurations. ASHA (Li et al., 2020a) extends the simple and very efficient idea of successive halving by introducing asynchronous evaluation of different configurations, which leads to further practical speedups thanks to better utilisation of workers in a parallel setting.

Related to the problem of efficiency in HPO, cost-aware HPO explicitly accounts for the cost of the evaluations of different configurations. Previous work on cost-aware HPO for multi-fidelity algorithms such as CAHB (Ivkin et al., 2021) keeps a tight control on the budget spent during the HPO process. This is different from our work, as we reduce the budget spent by terminating the HPO procedure early instead of allocating the compute budget in its entirety. Moreover, PASHA could be combined with CAHB to leverage the cost-based resources allocation.

Recently, researchers considered dataset subsampling to speedup HPO and NAS. Shim et al. (2021) have combined coresets with PC-DARTS (Xu et al., 2020) and showed that they can find well-performing architectures using only 10% of the data and 8.8x less search time. Similarly, Visalpara et al. (2021) have combined subset selection methods with the Tree-structured Parzen Estimator (TPE) for HPO (Bergstra et al., 2011). With a 5% subset they obtained between an 8x to 10x speedup compared to standard TPE. However, in both cases it is difficult to say in advance what subsampling ratio to use. For example, the 10% ratio in (Shim et al., 2021) incurs no decrease in accuracy, while reducing further to 2% leads to a substantial (2.6%) drop in accuracy. In practice, it is difficult to find a trade-off between the time required for tuning (proportional to the subset size) and the loss of performance for the final model because these change, sometimes wildly, between datasets. Further, Zhou et al. (2020a) have observed that for a fixed number of iterations, rank consistency is better if we use more training samples and fewer epochs rather than fewer training samples and more epochs. This observation gives further motivation for using the whole dataset for HPO/NAS and design new approaches, like PASHA, to save computational resources.

4.3 Problem Setup

The problem of selecting the best configuration of a machine learning algorithm to be trained is formalized in (Jamieson and Talwalkar, 2016) as a non-stochastic bandit problem. In this setting the learner (the hyperparameter optimizer) receives N hyperparameter configurations and it has to identify the best performing one with the constraint of not spending more than a fixed amount of resources R (e.g. total number of training epochs) on a specific configuration. R is considered given, but in practice users do not have a good way for selecting it, which can have undesirable consequences: if the value is too small, the model performance will be sub-optimal, while if the budget is too large, the user will incur a significant cost without any practical return. This leads users to overestimate R, setting it to a large amount of resources in order to guarantee the convergence of the model. We maintain the concept of maximum amount of resources in our algorithm but we prefer to interpret R as a "safety net", a cost not to be surpassed (e.g. in case an error prevents a normal behaviour of the algorithm), instead of the exact amount of resources spent for the optimization.

This setting could be extended with additional assumptions, based on empirical observation, removing some extreme cases and leading to a more practical setup. In particular, when working with large datasets we observe that the curve of the loss for configurations (called arms in the bandit literature) continuously decreases (in expectation). Moreover, "crossing points" between the curves are rare (excluding noise), and they are almost always in the very initial part of the training procedure. Viering and Loog (2021); Mohr and van Rijn (2022) provide an analysis of learning curves and note that in practice most learning curves are well-behaved, with Bornschein et al. (2020); Domhan et al. (2015) reporting similar findings.

More formally, let us define R as the maximum number of resources needed to train an ML algorithm to convergence. Given $\pi_m(i)$ the ranking of configuration i after using m resources for training, there exists minimum R^* much smaller than R such that for all amounts of resources r larger than R^* the rankings of configurations trained with r resources remain the same: $\exists R^* \ll R : \forall i \in \{\text{configurations}\}, \forall r > R^*, \pi_{R^*}(i) = \pi_r(i)$. The existence of such a quantity, limited to the best performing configuration, is also assumed by Jamieson and Talwalkar (2016), and it is leveraged to quantify the budget required to identify the best performing configuration. If we knew R^*, it would be sufficient to run all configurations with exactly that amount of resources to identify the best one and then just train the model from scratch with all the data using that configuration. Unfortunately that quantity is unknown and can only be estimated during the optimization procedure. Note that in practice there is noise involved in training of neural networks, so similarly performing configurations will repeatedly swap their ranks.

4.4 Methodology

PASHA is an extension of ASHA (Li et al., 2020a) inspired by the "doubling trick" (Auer et al., 1995). PASHA targets improvements for hyperparameter tuning on large datasets by hinging on the assumptions made about the crossing points of the learning curves in Section 4.3. The algorithm starts with a small initial amount of resources and progressively increases them if the ranking of the configurations in the top two *rungs* (rounds of promotion) has not stabilized. The ability of our approach to stop early automatically is the key benefit. We illustrate the approach in Figure 4.1, showing how we stop evaluating configurations for additional rungs if rankings are stable.

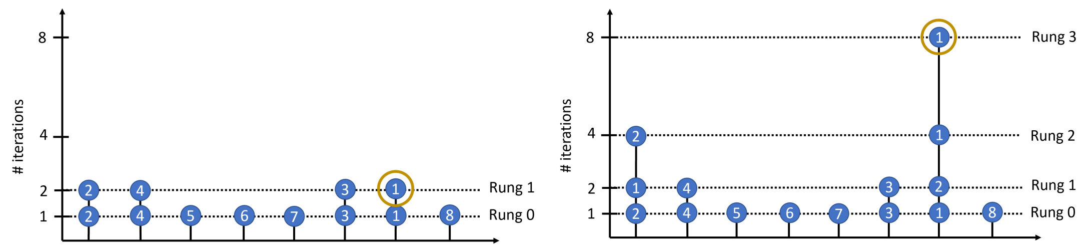


Figure 4.1: Illustration of how PASHA stops early if the ranking of configurations has stabilized. Left: the ranking of the configurations (displayed inside the circles) has stabilized, so we can select the best configuration and stop the search. Right: the ranking has not stabilized, so we continue.

We describe the details of our proposed approach in Algorithm 4. Given η, a hyperparameter used both in ASHA and PASHA to control the fraction of configurations to prune, PASHA sets the current maximum resources R_t to be used for evaluating a configuration using the reduction factor η and the minimum amount of resources r to be used (K_t is the current maximum rung). The approach increases the maximum number of resources allocated to promising configurations each time the ranking of configurations in the top two rungs becomes inconsistent. For example, if we can currently train configurations up to rung 2 and the ranking of configurations in rung 1 and rung 2 is not consistent, then we allow training part of the configurations up to rung 3, i.e. one additional rung.

The minimum amount of resources r is a hyperparameter to be set by the user. It is significantly easier to set compared to R as r is the minimum amount of resources required to see a meaningful difference in the performance of the models, and it can be easily estimated empirically by running a few small-scale experiments.

Algorithm 4 Progressive asynchronous successive halving (PASHA)

```
1: input minimum resource $r$, reduction factor $\eta$
2: function PASHA()
3:     $t = 0, R_0 = \eta r$, $K_0 = \lfloor \log_\eta (R_0/r) \rfloor$
4:     while desired do
5:         for each free worker do
6:             $(\theta, k) =$ get_job()
7:             run_then_return_val_loss$(\theta, r\eta^k)$
8:         end for
9:         for completed job ($\theta$, $k$) with loss $l$ do
10:            Update configuration $\theta$ in rung $k$ with loss $l$
11:            if $k \geq K_t - 1$ then
12:                $\pi_k =$ configuration_ranking$(k)$
13:            end if
14:            if $k = K_t$ and $\pi_k \not\equiv \pi_{k-1}$ then
15:                $t = t + 1$
16:                $R_t = \eta^t R_0$
17:                $K_t = \lfloor \log_\eta (R_t/r) \rfloor$
18:            end if
19:        end for
20:    end while
21: end function
22: function get_job()
23:    // Check if there is a promotable config
24:    for $k = K_t - 1, \ldots, 1, 0$ do
25:        candidates $=$ top_k(rung $k$, $|$rung $k|/\eta$)
26:        promotable $= \{c \in$ candidates : $c$ not promoted$\}$
27:        if $|$promotable$| > 0$ then
28:            return promotable$[0], k + 1$
29:        end if
30:        // If not, grow bottom rung
31:        Draw random configuration $\theta$
32:        return $\theta, 0$
33:    end for
34: end function
```

We also set a maximum amount of resources R so that PASHA can default to ASHA if needed and avoid increasing the resources indefinitely. While it is not generally reached, it provides a safety net.

4.4.1 Soft Ranking

Due to the noise present in the training process, negligible differences in the measured predictive performance of different configurations can lead to significantly different rankings. For these reasons we adopt what we call "soft ranking". In soft ranking, configurations are still sorted by predictive performance but are considered equivalent if the performance difference is smaller than a value ε (or equal to it). Instead of producing a sorted list of configuration, this provides a list of lists where for every position of the ranking there is a list of equivalent configurations. The concept is explained graphically in Figure 4.2, and we also provide a formal definition. For a set of n configurations $c_1, c_2, \cdots, c_i, \cdots, c_n$ and performance metric f (e.g. accuracy) with $f(c_1) \leq f(c_2) \leq \cdots \leq f(c_i) \leq \cdots \leq f(c_n)$, soft rank at position i is defined as

$$\text{soft rank}_i = \left\{c_j \in \text{configurations} : |f(c_i) - f(c_j)| \leq \varepsilon\right\}.$$

When deciding on if to increase the resources, we go through the ranked list of configurations in the top rung and check if the current configuration at the given rank was in the list of configurations for that rank in the previous rung. If there is a configuration which does not satisfy the condition, we increase resources.

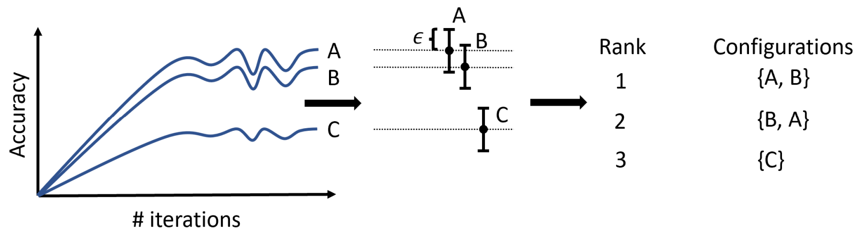


Figure 4.2: Illustration of soft ranking. There are three lists with the first two containing two items because the scores of the two configurations are closer to each other than ε.

4.4.2 Automatic Estimation of ε by Measuring Noise in Rankings

Every operation involving randomization gives slightly different results when repeated, the training process and the measurement of performance on the validation set are no exception. In an ideal world, we could repeat the process multiple times to compute empirical mean and variance to make a better decision. Unfortunately this is not possible in our case since the repeating portions of the training process will defeat the purpose of our work: speeding up the tuning process. Understanding when the differences between the performance measured for different configurations are "significant" is crucial for ranking them correctly. We devise a method to estimate a threshold below which differences are meaningless. Our intuition is that configurations with different performance maintain their relative ranking over time. On the other hand, configurations that repeatedly swap their rankings perform similarly well and the performance difference in the current epoch or rung is simply due to noise. We want to measure this noise and use it to automatically estimate the threshold value ε to be used in the soft-ranking described above.

Formally we can define a set of pairs of configurations that perform similarly well by the following:

$$\begin{aligned} S : \{(c,c') : &\left(\pi_{r_j}(c) > \pi_{r_j}(c') \wedge \pi_{r_k}(c) < \pi_{r_k}(c') \wedge \pi_{r_l}(c) > \pi_{r_l}(c')\right) \\ &\vee \left(\pi_{r_j}(c) < \pi_{r_j}(c') \wedge \pi_{r_k}(c) > \pi_{r_k}(c') \wedge \pi_{r_l}(c) < \pi_{r_l}(c')\right)\}, \end{aligned} \tag{4.1}$$

for resource levels (e.g. epochs – not rungs) $r_j > r_k > r_l$, using the same notation as earlier to refer to resources. In practice we have per-epoch validation performance statistics and use these to find resource levels r_j, r_k, r_l that have configurations with the criss-crossing behaviour (there can be several epochs between such resource levels). We only consider configurations (c,c') that made it to the latest rung, so $r\eta^{K_t-1} \geq r_j > r\eta^{K_t-2}$. However, we allow for the criss-crossing to happen across epochs from any rungs. The value of ε can then be calculated as the N-th percentile of distances between the performances of configurations in S:

$$\varepsilon = \mathrm{P}_{N,(c,c')\in S}|f_{r_j}(c) - f_{r_j}(c')|.$$

The exact value of r_j depends on the considered pair of configurations (c,c'). To uniquely define f_{r_j}, we take the maximum resources r_j currently available for both configurations in the considered pair (c,c'). Let us consider the following example setup: the top rung has 8 epochs and the next one has 4 epochs, there are three configurations c_a, c_b, c_c that made it to the top rung and were trained for 8, 8 and 6 epochs so far respectively. Assuming there was criss-crossing within each pair (c_a,c_b), (c_a,c_c) and (c_b,c_c), the set of distances between configurations in S is $\{|f_8(c_a) - f_8(c_b)|, |f_6(c_a) - f_6(c_c)|, |f_6(c_b) - f_6(c_c)|\}$. The value of ε is recalculated every time we receive new information about the performances of configurations. Initially the value of ε is set to 0, which means that we check for exact ranking if we cannot yet calculate the value of ε.

4.5 Experiments

In this section we empirically evaluate the performance of PASHA. Its goal is not to provide a model with a higher accuracy, but to identify the best configuration in a shorter amount of time so that we can then re-train the model from scratch. Overall, we target a significantly faster tuning time and on-par predictive performance when comparing with the models identified by state-of-the-art optimizers like ASHA. Re-training after HPO or NAS is important because HPO and NAS in general require to reserve a significant part of the data (often around 20 or 30%) to be used as a validation set. Training with fewer data is not desirable because in practice it is observed that training a model on the union of training and validation sets provides better results.

We tested our method on three different sets of experiments. The first set evaluates the algorithm on NAS problems and uses NASBench201 (Dong and Yang, 2020). The second and third sets focus on HPO and were run on two large-scale tasks from the PD1 benchmark (Wang et al., 2021b) and the LCBench benchmark (Zimmer et al., 2021).

4.5.1 Setup

Our experimental setup consists of two phases: 1) run the hyperparameter optimizer until $N = 256$ candidate configurations are evaluated; and 2) use the best configuration identified in the first phase to re-train the model from scratch. For the purpose of these experiments we re-train all the models using only the training set. This avoids introducing an arbitrary choice on the validation set size and allows us to leverage standard benchmarks such as NASBench201. In real-world applications the model can be trained on both training and validation sets. All our results report only the time invested in identifying the best configuration since the re-training time is comparable for all optimizers. All results are averaged over multiple repetitions, with the details specified for each set of experiments separately. We use $N = 90$-th percentile when calculating the value of ε.

We use 4 workers to perform parallel and asynchronous evaluations. The choice of R is sensitive for ASHA as it can make the optimizer consume too many resources and penalize the performance. For a fair comparison, we make R dataset-dependent taking the maximum amount of resources in the considered benchmarks. r is also dataset-dependent and η, the halving factor, is set to 3 unless otherwise specified. The same values are used for both ASHA and PASHA. Runtime reported is the time spent on HPO (without retraining), including the time for computing validation set performance.

We compare PASHA with ASHA (Li et al., 2020a), a widely-adopted approach for multi-fidelity HPO, and other relevant baselines. In particular, we consider "one-epoch baseline" that trains all configurations for one epoch (the minimum available resources) and then selects the most promising configuration, and "random baseline" that randomly selects the configuration without any training. For both one-epoch and random baselines we sample $N = 256$ configurations, using the same scheduler and seeds as for PASHA and ASHA. All reported accuracies are after retraining for $R = 200$ epochs. In addition, two, three and five-epoch baselines are evaluated in Appendix B.1.

4.5.2 Neural Architecture Search Experiments

For our NAS experiments we leverage the well-known NASBench201 (Dong and Yang, 2020) benchmark. The task is to identify the network structure providing the best accuracy on three different datasets (CIFAR-10, CIFAR-100 and ImageNet16-120) independently. We use $r = 1$ epoch and $R = 200$ epochs. We repeat the experiments using 5 random seeds for the scheduler and 3 random seeds for NASBench201 (all that are available), resulting in 15 repetitions. Some configurations in NASBench201 do not have all seeds available, so we impute them by averaging over the available seeds. To measure the predictive performance we report the best accuracy on the combined validation and test set provided by the creators of the benchmark.

The results in Table 4.1 suggest PASHA consistently leads to strong improvements in runtime, while achieving similar accuracy values as ASHA. The one-epoch baseline has noticeably worse accuracies than ASHA or PASHA, suggesting that PASHA does a good job of deciding when to continue increasing the resources – it does not stop too early. Random baseline is a lot worse than the one-epoch baseline, so there is value in performing NAS. We also report the maximum resources used to find how early the ranking becomes stable in PASHA. The large variances are caused by stopping HPO at different rung levels for different seeds (e.g. 27 and 81 epochs). Note that the time required to train a model is about 1.3h for CIFAR-10 and CIFAR-100, and about 4.1h for ImageNet16-120, making the total tuning time of PASHA comparable or faster than the training time.

We also ran additional experiments testing PASHA with a reduction factor of $\eta = 2$ and $\eta = 4$ instead of $\eta = 3$, the usage of PASHA as a scheduler in MOBSTER (Klein et al., 2020) and alternative ranking functions. These experiments provided similar findings as the above and are described next.

Table 4.1: NASBench201 results. PASHA leads to large improvements in runtime, while achieving similar accuracy as ASHA.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
	PASHA	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
	One-epoch baseline	93.30 ± 0.61	0.3h ± 0.0h	8.5x	1.0 ± 0.0
	Random baseline	72.88 ± 19.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
CIFAR-100	ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
	PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
	One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
	Random baseline	42.83 ± 18.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
ImageNet16-120	ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
	PASHA	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
	One-epoch baseline	41.42 ± 4.98	1.0h ± 0.0h	8.8x	1.0 ± 0.0
	Random baseline	20.75 ± 9.97	0.0h ± 0.0h	N/A	0.0 ± 0.0

Reduction Factor An important parameter for the performance of multi-fidelity algorithms like ASHA is the reduction factor. This hyperparameter controls the fraction of pruned candidates at every rung. The optimal theoretical value is e and it is typically set to 2 or 3. In Table 4.2 we report the results of the different algorithms ran with $\eta = 2$ and $\eta = 4$ on CIFAR-100 (the full set of results is in Appendix B.2). The gains are consistent also for $\eta = 2$ and $\eta = 4$, with a larger speedup when using $\eta = 2$ as that allows PASHA to make more decisions and identify earlier that it can stop the search.

Table 4.2: NASBench201 – CIFAR-100 results with various reduction factors η. The speedup is large for both $\eta = 2$ and $\eta = 4$, and accuracy similar to ASHA is retained.

Dataset	Reduction factor	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-100	$\eta = 2$	ASHA	71.67 ± 0.84	3.8h ± 1.0h	1.0x	200.0 ± 0.0
		PASHA	71.65 ± 1.42	0.9h ± 0.1h	4.2x	5.9 ± 2.0
	$\eta = 4$	ASHA	71.43 ± 1.13	2.7h ± 0.9h	1.0x	200.0 ± 0.0
		PASHA	72.09 ± 1.22	1.0h ± 0.4h	2.8x	25.1 ± 49.0

Bayesian Optimization Bayesian Optimization combined with multi-fidelity methods such as Successive Halving can improve the predictive performance of the final model (Klein et al., 2020). In this set of experiments, we verify PASHA can speedup also these kinds of methods. Our results are reported in Table 4.3, where we can clearly see PASHA obtains a similar accuracy result as ASHA with significant speedup.

Table 4.3: NASBench201 results for ASHA with Bayesian Optimization searcher – MOBSTER (Klein et al., 2020) and similarly extended version of PASHA. The results show PASHA can be successfully combined with a smarter selection strategy.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	MOBSTER	94.21 ± 0.28	5.0h ± 1.1h	1.0x	200.0 ± 0.0
	PASHA BO	94.00 ± 0.20	2.6h ± 1.8h	2.0x	70.7 ± 81.6
CIFAR-100	MOBSTER	72.79 ± 0.68	5.7h ± 1.4h	1.0x	200.0 ± 0.0
	PASHA BO	72.16 ± 1.07	1.6h ± 0.5h	3.7x	13.0 ± 8.7
ImageNet16-120	MOBSTER	46.21 ± 0.70	15.1h ± 4.0h	1.0x	200.0 ± 0.0
	PASHA BO	45.36 ± 1.06	3.9h ± 1.2h	3.9x	11.8 ± 7.9

Alternative Ranking Functions We have considered a variety of alternative ranking functions in addition to the soft ranking function that automatically estimates the value of ε by measuring noise in rankings. These include simple ranking (equivalent to soft ranking with $\varepsilon = 0.0$), soft ranking with fixed values of ε or obtained using various heuristics (for example based on the standard deviation of objective values in the previous rung), Rank Biased Overlap (RBO) (Webber et al., 2010), and our own reciprocal rank regret metric (RRR) that considers the objective values of configurations. Details of the ranking functions and additional results are in Appendix B.3.

Table 4.4 shows a selection of the results on CIFAR-100 with full results in the appendix. We can see there are also other ranking functions that work well and that simple ranking is not sufficiently robust – some benevolence is needed. However, the ranking function that estimates the value of ε by measuring noise in rankings (to which we refer simply as PASHA) remains the easiest to use, is well-motivated and offers both excellent performance and large speedup.

4.5.3 Hyperparameter Optimization Experiments

We further utilize the PD1 HPO benchmark (Wang et al., 2021b) to show the usefulness of PASHA in large-scale settings. In particular, we take WMT15 German-English (Bojar et al., 2015) and ImageNet (Deng et al., 2009) datasets that use xformer (Lefaudeux

Table 4.4: NASBench201 – CIFAR-100 results for a variety of ranking functions, showing there are also other well-performing options, even though those are harder to use and are less interpretable.

Approach	Accuracy (%)	Runtime (s)	Speedup factor	Max resources
ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
PASHA direct ranking	71.69 ± 1.05	2.8h ± 0.7h	1.1x	200.0 ± 0.0
PASHA soft ranking $\epsilon = 2.5\%$	71.41 ± 1.15	1.5h ± 0.7h	2.1x	88.3 ± 74.4
PASHA soft ranking $\epsilon = 2\sigma$	71.14 ± 0.97	1.9h ± 0.7h	1.7x	136.4 ± 75.8
PASHA RBO	71.51 ± 0.93	2.4h ± 0.7h	1.3x	180.5 ± 50.6
PASHA RRR	71.42 ± 1.51	1.2h ± 0.5h	2.6x	39.3 ± 51.4
One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
Random baseline	42.83 ± 18.20	0.0h ± 0.0h	N/A	0.0 ± 0.0

et al., 2021) and ResNet50 (He et al., 2016) models. Both of them are datasets with a large amount of training examples, in particular WMT15 German-English has about 4.5M examples, while ImageNet has about 1.3M examples.

In PD1 we optimize four hyperparameters: base learning rate $\eta \in \left[10^{-5}, 10.0\right]$ (log scale), momentum $1-\beta \in \left[10^{-3}, 1.0\right]$ (log scale), polynomial learning rate decay schedule power $p \in [0.1, 2.0]$ (linear scale) and decay steps fraction $\lambda \in [0.01, 0.99]$ (linear scale). The minibatch size used for WMT experiments is 64, while the minibatch size for ImageNet experiments is 512. There are 1414 epochs available for WMT and 251 for ImageNet. There are also other datasets in PD1, but these only have a small number of epochs with 1 epoch being the minimum amount of resources. As a result there would not be enough rungs to see benefits of the early stopping provided by PASHA. If resources could be defined in terms of fractions of epochs, PASHA could be beneficial there too. Most public benchmarks have resources defined in terms of epochs, but in practice it is possible to define resources also in alternative ways. We use 1-NN as a surrogate model for the PD1 benchmark. We repeat our experiments using 5 random seeds and there is only one dataset seed available.

The results in Table 4.5 show that PASHA leads to large speedups on both WMT and ImageNet datasets. The speedup is particularly impressive for the significantly larger WMT dataset where it is about 15.5x, highlighting how PASHA can significantly accelerate the HPO search on datasets with millions of training examples (WMT has about 4.5M training examples). The one-epoch baseline obtains similar accuracy as ASHA and PASHA for WMT, but performs significantly worse on ImageNet dataset. This result suggests that simple approaches such as the one-epoch baseline are not robust and solutions such as PASHA are needed (which we also saw on NASBench201). Selecting the hyperparameters randomly leads to significantly worse performance than any of the other approaches. Note that although PASHA finds a configuration for ImageNet that is much stronger than the baselines, average accuracy compared to ASHA is about 2% lower there. This shows the accuracy of the solution found by PASHA can be worse than that of ASHA in some cases, even if the overall performance is still very strong.

Table 4.5: Results of the HPO experiments on WMT and ImageNet tasks from the PD1 benchmark. Mean and std of the best validation accuracy (or its equivalent as given in the PD1 benchmark).

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
WMT	ASHA	62.72 ± 1.41	43.7h ± 37.2h	1.0x	1357.4 ± 80.4
	PASHA	62.04 ± 2.05	2.8h ± 0.6h	15.5x	37.8 ± 21.6
	One-epoch baseline	62.36 ± 1.40	0.6h ± 0.0h	67.3x	1.0 ± 0.0
	Random baseline	33.93 ± 21.96	0.0h ± 0.0h	N/A	0.0 ± 0.0
ImageNet	ASHA	75.10 ± 2.03	7.3h ± 1.2h	1.0x	251.0 ± 0.0
	PASHA	73.37 ± 2.71	3.8h ± 1.0h	1.9x	45.0 ± 30.1
	One-epoch baseline	63.40 ± 9.91	1.1h ± 0.0h	6.7x	1.0 ± 0.0
	Random baseline	36.94 ± 31.05	0.0h ± 0.0h	N/A	0.0 ± 0.0

We additionally evaluate PASHA on the LCBench benchmark (Zimmer et al., 2021) where only modest speedups can be expected due to a small number of epochs (and hence rungs) available. LCBench limits the maximum amount of resources per configuration to 50 epochs, so when using and setting the minimum resource level to 1 epoch, it is a challenging testbed for an algorithm like PASHA. The hyperparameters optimized include number of layers $\in [1,5]$, max. number of units $\in [64,1024]$ (log scale), batch size $\in [16,512]$ (log scale), learning rate $\in \left[10^{-4},10^{-1}\right]$ (log scale), weight decay $\in \left[10^{-5},10^{-1}\right]$, momentum $\in [0.1,0.99]$ and max. value of dropout $\in [0.0,1.0]$. Similarly as in our other experiments, we use $\eta = 3$ and stop after sampling 256 candidates.

Table 4.6: Results of the HPO experiments on the LCBench benchmark. Mean and std of the test accuracy across five random seeds. PASHA achieves similar accuracies as ASHA, but gives only modest speedups because of the limited number of rung levels and opportunities to stop the HPO early. To enable large speedup from PASHA, we could redefine the rung levels in terms of neural network weights updates rather than epochs.

Dataset	ASHA accuracy (%)	PASHA accuracy (%)	PASHA speedup
APSFailure	97.52 ± 0.92	97.01 ± 0.75	1.3x
Amazon_employee_access	94.01 ± 0.40	94.21 ± 0.00	1.1x
Australian	83.35 ± 0.33	83.35 ± 0.51	1.1x
Fashion-MNIST	86.70 ± 1.87	86.34 ± 1.32	1.1x
KDDCup09_appetency	98.22 ± 0.00	98.22 ± 0.00	1.1x
MiniBooNE	86.13 ± 1.57	86.24 ± 1.62	1.4x
Adult	79.14 ± 0.85	79.14 ± 0.85	1.2x
Airlines	59.57 ± 1.32	59.22 ± 0.76	1.4x
Albert	64.31 ± 0.99	64.23 ± 0.61	1.2x
Bank-marketing	88.34 ± 0.07	88.38 ± 0.00	1.2x
Blood-transfusion-service-center	79.92 ± 0.20	76.99 ± 6.00	1.1x
Car	86.60 ± 6.41	86.60 ± 6.41	1.1x
Christine	71.05 ± 1.17	70.15 ± 1.85	1.2x
Cnae-9	94.10 ± 0.31	94.44 ± 0.11	1.0x
Connect-4	62.28 ± 6.87	65.69 ± 0.04	1.2x
Covertype	59.76 ± 3.24	61.64 ± 1.64	1.2x
Credit-g	70.30 ± 0.84	70.79 ± 0.68	1.1x
Dionis	64.58 ± 9.89	64.58 ± 9.89	1.1x
Fabert	56.11 ± 10.89	53.47 ± 9.75	1.1x
Helena	19.16 ± 3.20	19.16 ± 3.20	1.1x
Higgs	66.48 ± 3.16	66.48 ± 3.16	1.1x
Jannis	58.92 ± 2.38	59.63 ± 2.81	1.4x
Jasmine	75.85 ± 0.36	75.55 ± 0.68	1.0x
Jungle_chess_2pcs_raw_endgame_complete	72.86 ± 4.69	74.94 ± 7.84	1.3x
Kc1	80.32 ± 4.37	80.86 ± 3.37	1.2x
Kr-vs-kp	92.50 ± 3.93	90.95 ± 4.70	1.0x
Mfeat-factors	98.21 ± 0.15	98.15 ± 0.15	1.1x
Nomao	94.12 ± 0.60	94.25 ± 0.64	1.1x
Numerai28.6	52.03 ± 0.55	52.30 ± 0.24	1.3x
Phoneme	76.65 ± 2.65	75.42 ± 2.87	1.1x
Segment	83.15 ± 2.54	83.15 ± 2.54	1.0x
Sylvine	90.57 ± 1.87	90.89 ± 2.04	1.0x
Vehicle	71.76 ± 2.57	71.76 ± 2.57	1.1x
Volkert	50.72 ± 1.91	50.72 ± 1.91	1.1x

Overall, the results in Table 4.6 confirm an accuracy level on-par with ASHA. While, as expected, the speedup is reduced compared to the other experiments, in several cases PASHA achieves a 20+% speedup with peaks around 40%.

If only a small number of epochs is sufficient for training the model on the given dataset, then HPO can be performed on a sub-epoch basis, e.g. defining the rung levels in terms of iterations instead of epochs. PASHA would then be able to give a large speedup even in cases with smaller numbers of epochs – an example of which is LCBench.

4.6 Discussion

PASHA is designed to speed up finding the best configuration, making HPO and NAS more accessible. To do so, PASHA interrupts the tuning process when it considers the ranking of configurations to be sufficiently stable, not spending resources on evaluating configurations in later rungs. However, the benefits of such mechanism will be small in some circumstances. When the number of rungs is small, there will be few opportunities for PASHA to interrupt the tuning and provide large speedups, as demonstrated on the LCBench benchmark (Zimmer et al., 2021).

Public benchmarks usually fix the minimum resources to one epoch, while the maximum is benchmark-dependent (e.g. 200 epochs for NASBench201 and 50 for LCBench), leaving little control for algorithms like PASHA in some cases. Appendix B.4 analyses the impact of these choices.

For practical usage, we recommend having a maximum amount of resources at least 100 times larger than the minimum amount of resources when using $\eta = 3$ (default). This can be achieved by measuring resources with higher granularity (e.g. in terms of gradient updates) if needed.

4.7 Further Developments

PASHA is already making impact within the AutoML community. PASHA has been integrated into Syne Tune hyperparameter optimization library (Salinas et al., 2022) so that it can be easily used in practice, and it has been highlighted as one of the advanced hyperparameter optimization techniques that can be used with the Renate continual learning library (Wistuba et al., 2023)). PASHA has been included in recent surveys that focus on the connections between large language models and AutoML (Tornede et al., 2023; Xu et al., 2024), showing it can be particularly useful in the new era of large-scale models. It has also been mentioned as related work in various papers, including (Ruhkopf et al., 2023) that introduces a new method for algorithm selection and (Egele et al., 2023) that studies the usefulness of a one-epoch baseline.

Since publishing PASHA there have been various developments within hyperparameter optimization and neural architecture search. Improvements have been made in terms of the quality of the found solution as well as the speed of the search. PriorBand (Mallik et al., 2023) is tailored for deep learning applications, and is able to utilize expert inputs as well as cheaper proxy tasks to improve the quality of the found configuration. Kadra et al. (2023) have utilized the power law nature of learning curves for Bayesian Optimization, introducing a solution that dynamically pauses and trains the sampled configurations, leading to best any-time results. Conformal quantile regression has been used to model the target function within model-based HPO approaches in a more realistic and robust way (Salinas et al., 2023), leading to faster HPO convergence. Additionally, a meta-learning method based on task-similarity (Watanabe et al., 2023) has been proposed to speed up multi-objective HPO with tree-structured Parzen estimator.

HPO and NAS have also seen advances more broadly. New variations of HPO have been introduced, including an ordered transfer HPO (Hellan et al., 2023) where tasks come sequentially, with the most recent ones being the most relevant. There has also been work on making HPO more interpretable, by utilizing symbolic explanations to quantify how hyperparameter values impact the model performance (Segel et al., 2023). Further, HPO has been utilized as a novel solution for addressing socially-important problems, as part of the FairTune framework (Dutt et al., 2024) that optimizes parameter-efficient fine-tuning for fairness in medical image analysis. In NAS in particular, one of the most important developments has been the introduction of significantly more expressive search spaces, based on context-free grammars (Schrodi et al., 2023).

4.8 Conclusions

In this work we have introduced a new variant of Successive Halving called PASHA. Despite its simplicity, PASHA leads to strong improvements in the tuning time. For example, in many cases it reduces the time needed to about one third compared to ASHA without a noticeable impact on the quality of the found configuration. For benchmarks with a small number of rungs (LCBench), PASHA provides more modest speedups but this limitation can be mitigated in practice by adopting a more granular unit of measure for resources. Further work could investigate the definition of rungs and resource levels, with the aim of understanding how they impact the decisions of the algorithm. More broadly this applies not only to PASHA but also to multi-fidelity algorithms in general.

PASHA can also be successfully combined with more advanced search strategies based on Bayesian Optimization to obtain improvements in accuracy at a fraction of the time. In the future, we would like to test combinations of PASHA with transfer-learning techniques for multi-fidelity such as RUSH (Zappella et al., 2021) to further decrease the tuning time.

***

We have developed algorithms for more efficient optimization of meta-parameters in two commonly used scenarios. Being able to optimize meta-parameters efficiently makes it possible to apply meta-learning methods to larger-scale settings and make the methods more accessible by decreasing their computational costs. Meta-learning has a wide range of applications, which increases the importance of its accessibility. To highlight the opportunities available we use meta-learning as a novel solution to two problems: uncertainty calibration and general-purpose few-shot learning.

Part II

Meta-Learning Applications

Chapter 5

Meta-Calibration: Learnable Uncertainty Calibration

The content of this chapter corresponds to paper:

Meta-Calibration: Learning of Model Calibration Using Differentiable Expected Calibration Error

Ondrej Bohdal, Yongxin Yang, Timothy Hospedales

Transactions on Machine Learning Research (TMLR), 2023

Calibration of neural networks is a topical problem that is becoming more and more important as neural networks increasingly underpin real-world applications. The problem is especially noticeable when using modern neural networks, for which there is a significant difference between the confidence of the model and the probability of correct prediction. Various strategies have been proposed to improve calibration, yet accurate calibration remains challenging. We propose a novel framework with two contributions: introducing a new differentiable surrogate for expected calibration error (DECE) that allows calibration quality to be directly optimized, and a meta-learning framework that uses DECE to optimize for validation set calibration with respect to model hyperparameters. The results show we achieve competitive performance with existing calibration approaches. Our framework opens up a new avenue and toolset for tackling calibration, which we believe will inspire further work on this important challenge.

5.1 Introduction

When deploying neural networks to real-world applications, it is crucial that models' own confidence estimates accurately match their probability of making a correct prediction. If a model is over-confident about its predictions, we cannot rely on it; while well-calibrated models can abstain or ask for human feedback in the case of uncertain predictions. Models with accurate confidence estimates about their own predictions can be described as well-calibrated. This is particularly important in applications involving safety or human impact – such as autonomous vehicles (Bojarski et al., 2016; Wiseman, 2022) and medical diagnosis (Jiang et al., 2012; Caruana et al., 2015; El-Sappagh et al., 2023), and tasks that directly rely on calibration such as outlier detection (Hendrycks and Gimpel, 2017; Liang et al., 2018; Wang et al., 2023b). However, modern neural networks are known to be badly calibrated (Guo et al., 2017; Gawlikowski et al., 2021).

This challenge of calibrating neural networks has motivated a growing area of research. Perhaps the simplest approach is to post-process predictions with techniques such as temperature scaling (Guo et al., 2017). However, this has limited efficacy (Wang et al., 2021a) and fails in the common situation of distribution shift between training and testing data (Ovadia et al., 2019; Tomani et al., 2021). It also reduces network's confidence in correct predictions. Another family of approaches modifies the model training regime to improve calibration. (Müller et al., 2019) show that label smoothing regularisation improves calibration by increasing overall predictive entropy. But it is unclear how to set the label smoothing parameter so as to optimize calibration. (Mukhoti et al., 2020) show that Focal loss leads to better calibrated models than standard cross-entropy, and (Kumar et al., 2018) introduce a proxy for calibration error to be minimised along with standard cross-entropy on the training set. However, this does not ensure calibration on the test set. (Ovadia et al., 2019; Mosser and Naeini, 2022) show that Bayesian neural network approaches are comparatively well-calibrated, however these are difficult and expensive to scale to large modern architectures.

The above approaches explore the impact of various architectures and design parameters on calibration. In this work we step back and consider how to optimize for calibration. Direct optimization for calibration would require a differentiable metric for calibration. However, calibration is typically measured using expected calibration error (ECE), which is not differentiable due to its internal binning/counting operation. Therefore our first contribution is a high-quality differentiable approximation to ECE, which we denote DECE. A second consideration is how to optimize for calibration –

given that calibration itself is a quantity with a generalization gap between training and validation (Carrell et al., 2022). We illustrate this challenge in Figure 5.1, which shows how validation calibration worsens as training calibration improves during training. To this end, our second contribution is to introduce a framework for meta-learning model calibration: We fit a model on the training set using a given set of hyperparameters, evaluate it on a disjoint validation set, and optimize for the hyperparameters that lead to the best *validation* calibration as measured by DECE. Our framework for differentiable optimization of validation calibration is generic and can potentially be used with any continuous model hyperparameters, of which we demonstrate two. Our third contribution is a specific choice of hyperparameters which, when meta-learned with a suitable calibration objective, are effective for tuning the base model's calibration. Specifically, we propose non-uniform label-smoothing, which can be tuned by meta-learning to penalise differently each unique combination of true- and predicted-label.

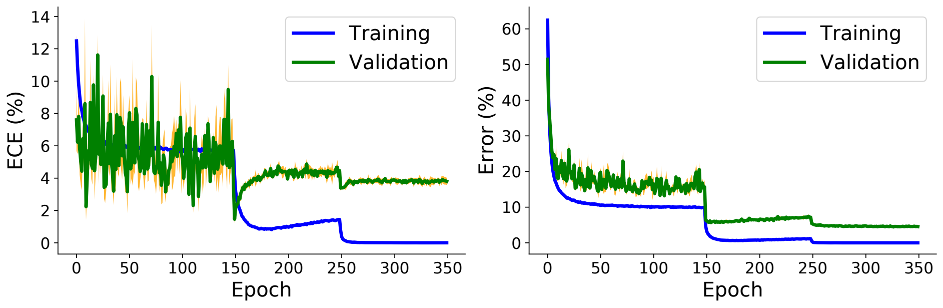


Figure 5.1: Illustration of the calibration generalization gap. ECE and classification error during training of ResNet18 on CIFAR-10, using cross-entropy loss and showing mean and std. across three random seeds. Training ECE and error fall to 0. However, calibration overfitting occurs and validation ECE increases. This motivates the need for meta-learning to tune hyperparameters to optimize *validation* calibration.

To summarise: We present a novel framework and toolset for improving model calibration by differentiable optimization of model hyperparameters with respect to validation calibration. We analyse our differentiable calibration metric in detail, and show that it closely matches the original non-differentiable metric. When instantiated with label smoothing hyperparameters, our empirical results show that our framework produces high-accuracy and well-calibrated models that are competitive with existing methods across a range of benchmarks and architectures. We provide the source code for Meta-Calibration at `https://github.com/ondrejbohdal/meta-calibration`.

5.2 Related Work

5.2.1 Calibration

Since finding modern neural networks are typically miscalibrated (Guo et al., 2017), model calibration has become a popular area for research (Gawlikowski et al., 2021) with many applications, including medical image segmentation (Judge et al., 2022) and object detection (Munir et al., 2023). (Guo et al., 2017) study a variety of potential solutions and find simple post-training rescaling of the logits – temperature scaling – works relatively well. (Kumar et al., 2018) propose a kernel-based measure of calibration called MMCE that they use as regularization during training of neural networks. (Mukhoti et al., 2020) show Focal loss – a relatively simple weighted alternative to cross-entropy – can be used to train well-calibrated neural networks. The classic Brier score (Brier, 1950), which is the squared error between the softmax vector with probabilities and the ground-truth one-hot encoding, has also been shown to work well. Similarly, label smoothing (Müller et al., 2019) has been shown to improve model calibration. These aforementioned methods do not optimize for calibration metric (e.g., ECE) directly, because the calibration metrics are usually non-differentiable. In this work, we propose a new high-quality differentiable approximation to ECE, and utilize it with meta-learning.

Karandikar et al. (2021) have proposed soft-binned ECE (SB-ECE) as an auxiliary loss to be used during training to encourage better calibration. The approach makes the binning operation used in ECE differentiable, leading to an additional objective that is more compatible with gradient-based methods. SB-ECE does not make the accuracy component of ECE differentiable, but we make all components of ECE differentiable. We also try to obtain a highly accurate approximation of the binning operation, while SB-ECE binning estimate for the left-most and right-most bin can be inaccurate as a result of using bin's center value. We provide additional comparison with SB-ECE in Appendix C, showing DECE provides a closer approximation to ECE than SB-ECE, and that empirically our meta-learning approach with DECE leads to better calibration.

5.2.2 Meta-Learning

We use the newly introduced DECE metric as part of the meta-objective for gradient-based meta-learning. Gradient-based meta-learning has become popular since the seminal work MAML (Finn et al., 2017) has successfully used it to solve the challenging problem of few-shot learning. Nevertheless, gradient-based meta-learning is not limited to few-shot learning problems, but it can also be used to solve various other challenges, including training with noisy labels (Shu et al., 2019; Algan and Ulusoy, 2022), dataset distillation (Wang et al., 2018b; Bohdal et al., 2020; Yu et al., 2023), domain generalization (Li et al., 2019b; Balaji et al., 2018), molecular property prediction (Chen et al., 2023c) and many others.

Gradient-based meta-learning is typically formulated as a bilevel optimization problem where the main model is trained in the inner loop and the meta-knowledge or hyperparameters are trained in the outer loop. In the case of few-shot learning it is possible to fully train the model within the inner loop – also known as offline meta-learning. In more realistic and larger scale settings such as ours, it is only feasible to do one or a few updates in the inner loop. This approach is known as online meta-learning (Hospedales et al., 2021) and means that we jointly train the main model as well as the meta-knowledge. Online meta-learning is most commonly done using the so-called $T_1 - T_2$ (Luketina et al., 2016) that updates the meta-knowledge by backpropagating through one step of the main model update. This is the approach that we adopt, however more advanced or efficient approaches are also available (Lorraine et al., 2020; Bohdal et al., 2021).

5.2.3 Label Smoothing

We use label smoothing as the meta-knowledge that we use to demonstrate the benefits of using our DECE metric. Label smoothing has been proposed by (Szegedy et al., 2016) as a technique to alleviate overfitting and improve the generalization of neural networks. It consists of replacing the one-hot labels by their softer alternative that gives non-zero target probabilities to the incorrect classes. Label smoothing has been studied in more detail, for example (Müller et al., 2019) have observed that label smoothing can improve calibration, but at the same time it can hurt knowledge distillation if used for training the teacher. (Mukhoti et al., 2020) have compared Focal loss with label smoothing among other approaches, showing that simple label smoothing strategy has a limited scope for state-of-the-art calibration.

However, we demonstrate that using meta-learning and our DECE objective, a more expressive form of label smoothing can achieve state-of-the-art calibration results. Note that meta-learning has already been used for label smoothing (Li et al., 2020b), but using it as meta-knowledge to directly optimize calibration is new.

5.3 Methodology

5.3.1 Preliminaries

We first discuss expected calibration error (ECE) (Naeini et al., 2015), before we derive a differentiable approximation to it. ECE measures the expected difference (in absolute value) between the accuracies and the confidences of the model on examples that belong to different confidence intervals. ECE is defined as

$$\mathrm{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\mathrm{acc}\left(B_m\right) - \mathrm{conf}\left(B_m\right)\right|,$$

where accuracy and confidence for bin B_m are

$$\begin{aligned}\mathrm{acc}\left(B_m\right) &= \frac{1}{|B_m|} \sum_{i \in B_m} \mathbf{1}\left(\hat{y}_i = y_i\right) \\ \mathrm{conf}\left(B_m\right) &= \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i.\end{aligned}$$

There are M interval bins each of size $1/M$ and n samples. Confidence $\hat{p}_i$ is the probability of the top prediction as given by the model for example i. We group the confidences into their corresponding bins, with bin B_m covering interval $(\frac{m-1}{M}, \frac{m}{M}]$. The predicted class of example i is $\hat{y}_i$, while y_i is the true class of example i and $\mathbf{1}$ is an indicator function.

ECE metric is not differentiable because assigning examples into bins is not differentiable and also accuracy is not differentiable due to the indicator function. We propose approximations to both binning and accuracy and derive a new metric called differentiable ECE (DECE).

5.3.2 Differentiable ECE

ECE is composed of accuracy, confidence and binning components, but only the confidence component is differentiable. Hence we need to find differentiable approximations for accuracy and binning.

Differentiable Accuracy In order to obtain a differentiable approximation to accuracy, we consider approaches that allow us to find a differentiable way to calculate the rank of a given class. Two approaches stand out: differentiable ranking (Blondel et al., 2020) and an all-pairs approach (Qin et al., 2010). While both allow us to approximate the rank in a differentiable way, differentiable ranking is implemented on CPU only, which would introduce a potential bottleneck for modern applications. All-pairs approach has asymptotic complexity of $O(n^2)$ for n classes, while differentiable ranking is $O(n \log n)$. However, if the number of classes is not in thousands or millions, differentiable ranking would be slower because of not using GPUs. We use the all-pairs approach to estimate the rank of a given class.

All-pairs (Qin et al., 2010) calculates a rank of class i as $[R(\cdot)]_i = 1 + \sum_{j \neq i} \mathbf{1}\left[\phi_i < \phi_j\right]$, where ϕ are the logits. We can obtain soft ranks by replacing the indicator function with a sigmoid scaled with some temperature value τ_a to obtain reliable estimates of the rank of the top predicted class. Once the rank $[R(\cdot)]_l$ for true class l is calculated, we can estimate the accuracy as $\mathrm{acc} = \max(0, 2 - [R]_l)$.

Soft Binning Our approach is similar to (Yang et al., 2018). We take confidence $\hat{p}_i$ for example x_i and pass it through one-layer neural network $\mathrm{softmax}((w\hat{p}_i + b)/\tau_b)$ parameterized with different values of w and b as explained in (Yang et al., 2018), with temperature τ_b to control the binning. This leads to M different probabilities, saying how likely it is that $\hat{p}_i$ belongs to the specific bin $B_{m \in 1..M}$. We will denote these probabilities as $o_m(x_i) = p(B_m|\hat{p}_i)$.

Putting these parts together, we define DECE using a minibatch of n examples as:

$$\mathrm{DECE} = \sum_{m=1}^{M} \frac{\sum_{i=1}^{n} o_m(x_i)}{n} \left|\mathrm{acc}(B_m) - \mathrm{conf}(B_m)\right|,$$

$$\mathrm{acc}(B_m) = \frac{1}{\sum_{i=1}^{n} o_m(x_i)} \sum_{i=1}^{n} o_m(x_i) \mathbf{1}(\hat{y}_i = y_i),$$

$$\mathrm{conf}(B_m) = \frac{1}{\sum_{i=1}^{n} o_m(x_i)} \sum_{i=1}^{n} o_m(x_i) \hat{p}_i.$$

5.3.3 Meta-Learning

Differentiable ECE provides an objective to optimize, but we still need to decide how to utilize it. One option could be to directly use it as an extra objective in combination with standard cross-entropy, as used by a few existing attempts (Karandikar et al., 2021; Kumar et al., 2018). However, we expect this to be unhelpful as calibration on the *training* set is usually good – the issue being a failure of calibration generalization to the held out validation or test set (Carrell et al., 2022), as illustrated in Figure 5.1. Meanwhile multi-task training with such non-standard losses may negatively affect the learning dynamics of existing well tuned model training procedures. To optimize for calibration of held-out data, without disturbing standard model training dynamics, we explore the novel approach of using DECE as part of the objective for hyperparameter meta-learning in an outer loop that wraps an inner learning process of conventional cross-entropy-driven model training.

Meta-Learning Objective We formulate our approach as a bilevel optimization problem. Our model is assumed to be composed of feature extractor θ and classifier ϕ. These are trained to minimise $\mathcal{L}_{CE}^{train}$, cross-entropy loss on the training set. The goal of meta-learning is to find hyperparameters ω so that training with them optimizes the meta-objective computed on the meta-validation set. In our case the meta-objective is a combination of cross-entropy and DECE to reflect that the meta-learned hyperparameters should lead to both good generalization and calibration. More specifically the meta-objective is $\mathcal{L}_{CE+\lambda DECE}^{val}$, with hyperparameter λ specifying how much weight is placed on calibration. The bilevel optimization problem can then be summarized as:

$$\begin{aligned} \omega^* &= \arg\min_{\omega} \mathcal{L}_{CE+\lambda DECE}^{val}(\phi^* \circ \theta^*(\omega)), \\ \phi^*, \theta^*(\omega) &= \arg\min_{\phi,\theta} \mathcal{L}_{CE}^{train}(\phi \circ \theta, \omega). \end{aligned} \tag{5.1}$$

To solve this efficiently, we adopt online meta-learning approach (Luketina et al., 2016; Hospedales et al., 2021) where we alternate base model and hyperparameter updates. This is an efficient strategy as we do not need to backpropagate through many inner-loop steps or retrain the model from scratch for each update of meta-knowledge.

When simulating training during the inner loop, we only update the classifier and keep the feature extractor frozen for efficiency, as suggested by (Balaji et al., 2018). Base model training is done separately using a full model update and a more advanced optimizer.

We give the overview of our meta-learning algorithm in Algorithm 5. The inner loop that trains the main model (θ, ϕ) (line 10) is conducted using hyperparameters ω, while the outer loop (line 12) that trains the hyperparameters does not directly use it for evaluating the meta-objective (e.g. no learnable label smoothing is applied to the meta-validation labels that are used in the outer loop). We backpropagate through one step of update of the main model.

Hyperparameter Choice A key part of meta-learning is to select suitable meta-knowledge (hyperparameters) that we will optimize to achieve the meta-learning goal (Hospedales et al., 2021). Having cast calibration optimization as a meta-learning process, we are free to use any of the wide range of hyperparameters surveyed in (Hospedales et al., 2021). Note also that in contrast to grid search that standard temperature scaling (Guo et al., 2017) and other approaches rely on, we have gradients with respect to hyperparameters and so can therefore potentially optimize calibration

Algorithm 5 Meta-Calibration

```
1: Input: $\alpha$, $\beta$: inner and outer-loop learning rates
2: Output: trained feature extractor $\theta$, classifier $\phi$ and label smoothing $\omega$
3: $\omega \sim p(\omega)$
4: $\phi, \theta \sim p(\phi), p(\theta)$
5: while training do
6:     Sample minibatch of training $x_t, y_t$ and meta-validation $x_v, y_v$ examples
7:     // For LS: use $\omega$ to smooth $y_t$
8:     // For L2: add unit-wise weight-decay $\omega$
9:     Calculate $\mathcal{L}_i = \mathcal{L}_{CE}\left(f_{\phi\circ\theta}(x_t), y_t, \omega\right)$
10:    Update $\theta, \phi \leftarrow \theta, \phi - \alpha\nabla_{\theta,\phi}\mathcal{L}_i$
11:    Calculate $\mathcal{L}_o = \mathcal{L}_{CE+\lambda DECE}\left(f_{\phi\circ\theta}(x_v), y_v\right)$
12:    Update $\omega \leftarrow \omega - \beta\nabla_{\omega}\mathcal{L}_o$
13: end while
```

with respect to high-dimensional hyperparameters. In this work we explore two options to demonstrate this generality: 1) Unit-wise L2 regularisation coefficients of each weight in the classifier layer, inspired by (Balaji et al., 2018) and (Lorraine et al., 2020); and 2) various types of learnable label smoothing (LS) (Müller et al., 2019). However, we found LS to be better overall, so our experiments focus on this and selectively compare against L2.

Learnable Label Smoothing Learnable label smoothing learns one or more coefficients to smooth the one-hot encoded labels. More formally, if there are K classes in total, y_k is 1 for the correct class $k = c$ and y_k is 0 for all classes $k \neq c$, then with learnable label smoothing ω the soft label for class k becomes

$$y_k^{LS} = y_k(1 - \omega_{c,k}) + \omega_{c,k}/K.$$

In the scalar case of label smoothing ω_c is the same for all classes, while for the vector case it takes different values for each class c. We consider scalar and vector variations as part of ablation.

Our main variation of meta-calibration uses non-uniform label smoothing. It is computed using the overall strength of smoothing ω_c^s for the correct class c and also $\omega_{c,k}^d$ saying how ω_c^s is distributed across the various incorrect classes $k \neq c$. Given correct class c, with this variation the soft label for class k is calculated as:

$$y_k^{LS} = y_k(1 - \omega_c^s) + \omega_c^s \frac{\omega_{c,k}^d}{\varepsilon + \sum_{i=1}^K \omega_{c,i}^d},$$

where we normalize the distribution weights to sum to 1 and use small value ε to avoid division by 0. Learnable label smoothing parameters are restricted to non-negative values, with the total label smoothing strength at most 0.5. In practice the above is implemented by learning a vector of K elements specifying the strengths of overall label smoothing for different correct classes c, and a matrix of $K \times (K-1)$ elements specifying how the label smoothing is distributed across incorrect classes $k \neq c$.

Learnable L2 Regularization In the case of learnable L2 regularization (cf: (Balaji et al., 2018)), the goal is to find unit-wise L2 regularization coefficients ω for the classifier layer ϕ so that training with them optimizes the meta-objective that includes DECE (θ is the feature extractor). The inner loop loss becomes

$$\mathcal{L}_i = \mathcal{L}_{CE}\left(f_{\phi\circ\theta}(x_t), y_t\right) + \omega\|\phi\|^2.$$

5.4 Experiments

Our experiments show that DECE-driven meta-learning can be used to obtain excellent calibration across a variety of benchmarks and models.

5.4.1 Datasets and Settings

We experiment with CIFAR-10 and CIFAR-100 benchmarks (Krizhevsky, 2009), SVHN (Netzer et al., 2011) and 20 Newsgroups dataset (Lang, 1995), covering both computer vision and NLP. For CIFAR benchmark, we use ResNet18, ResNet50, ResNet110 (He et al., 2016) and WideResNet26-10 (Zagoruyko and Komodakis, 2016) models. For SVHN we use ResNet18, while for 20 Newsgroups we use global pooling CNN (Lin et al., 2014). We extend the implementation provided by (Mukhoti et al., 2020) to implement and evaluate our meta-learning approach. We use the same hyperparameters as selected by the authors for fair comparison, which we summarize next.

CIFAR and SVHN models are trained for 350 epochs, with a multi-step scheduler that decreases the initial learning rate of 0.1 by a factor of 10 after 150 and 250 epochs. Each model is trained with SGD with momentum of 0.9, weight decay of 0.0005 and minibatch size of 128. 90% of the original training set is used for training and 10% for validation. In the case of meta-learning, we create a further separate *meta-validation* set that is of size 10% of the original training data, so we directly train with 80% of the original training data. 20 Newsgroups models are trained with Adam optimizer with the default parameters, 128 minibatch size and for 50 epochs. As the final model we select the checkpoint with the best validation accuracy.

For DECE, we use $M = 15$ bins and scaling parameters $\tau_a = 100, \tau_b = 0.01$. Learnable label smoothing coefficients are optimized using Adam (Kingma and Ba, 2015) optimizer with learning rate of 0.001. The meta-learnable parameters are initialized at 0.0 (no label smoothing or L2 regularization initially). The total number of meta-parameters is $K \times K$, 1 and K for the non-uniform, scalar and vector label smoothing respectively, while it is $512 \times K + K$ for learnable L2 regularization. We use $\lambda = 0.5$ in the meta-objective, and we have selected it based on validation set calibration and accuracy after trying several values.

5.4.2 Results

We first follow the experimental setup of (Mukhoti et al., 2020) and compare with the following alternatives: 1) cross-entropy, 2) Brier score (Brier, 1950), 3) Weighted MMCE (Kumar et al., 2018) with $\lambda = 2$, 4) Focal loss (Lin et al., 2017) with $\gamma = 3$, 5) Adaptive (sample dependent) focal loss (FLSD) (Mukhoti et al., 2020) with $\gamma = 5$ and $\gamma = 3$ for predicted probability $p \in [0, 0.2)$ and $p \in [0.2, 1)$ respectively. 6) Label smoothing (LS) with a fixed smoothing factor of 0.05. In all cases we report the mean and standard deviation across 3 repetitions to obtain a more reliable estimate of the performance. In contrast, (Mukhoti et al., 2020) report their results on only one run, so in the tables we include our own results for the comparison with the baselines.

We show the test ECE, test ECE after temperature scaling (TS) and test error rates in Tables 5.1, 5.2 and 5.3 respectively. Meta-Calibration leads to excellent intrinsic calibration without the need for post-processing (Table 5.1), which is practically valuable because post-processing is not always possible (Kim and Yun, 2020) or reliable (Ovadia et al., 2019). However, even after post-processing using TS Meta-Calibration gives competitive performance (Table 5.2), as evidenced by the best average rank across the considered scenarios. Table 5.3 shows that Meta-Calibration maintains comparable accuracy to the competitors, even if it does not have the best average rank there. Overall Meta-Calibration leads to significantly better intrinsic calibration, while keeping similar or only slightly worse accuracy. Note that while Brier score, Focal loss and FLSD modify the base model's loss function, our Meta-Calibration corresponds to the vanilla cross-entropy baseline, but where label smoothing is tuned by our DECE-driven hyperparameter meta-learning rather than being selected by hand.

Table 5.1: Test ECE (%, $\downarrow$): Our Meta-Calibration (MC) leads to excellent intrinsic calibration.

Dataset	Model	CE	Brier	MMCE	FL-3	FLSD-53	LS	MC (Ours)
CIFAR-10	ResNet18	4.23 ± 0.15	1.23 ± 0.03	4.36 ± 0.16	2.11 ± 0.09	2.22 ± 0.04	3.63 ± 0.06	1.17 ± 0.26
	ResNet50	4.20 ± 0.01	1.95 ± 0.15	4.49 ± 0.18	1.48 ± 0.19	1.68 ± 0.14	2.58 ± 0.26	1.09 ± 0.09
	ResNet110	4.81 ± 0.12	2.58 ± 0.17	4.20 ± 0.74	1.82 ± 0.20	2.16 ± 0.22	1.96 ± 0.36	1.07 ± 0.12
	WideResNet26-10	3.37 ± 0.11	1.03 ± 0.08	3.48 ± 0.06	1.57 ± 0.32	1.50 ± 0.15	3.68 ± 0.10	0.94 ± 0.10
CIFAR-100	ResNet18	8.79 ± 0.59	5.19 ± 0.18	7.41 ± 1.30	2.83 ± 0.27	2.47 ± 0.12	6.87 ± 0.29	2.52 ± 0.35
	ResNet50	12.56 ± 1.44	4.82 ± 0.36	9.02 ± 1.72	4.78 ± 1.00	5.43 ± 0.31	5.94 ± 0.52	3.07 ± 0.18
	ResNet110	14.96 ± 0.83	6.52 ± 0.56	12.29 ± 1.25	6.64 ± 1.42	7.38 ± 0.25	10.69 ± 0.39	2.80 ± 0.58
	WideResNet26-10	12.39 ± 1.44	4.26 ± 0.30	8.35 ± 2.79	2.36 ± 0.13	2.30 ± 0.36	3.94 ± 0.96	3.86 ± 0.34
SVHN	ResNet18	2.98 ± 0.08	1.94 ± 0.10	3.14 ± 0.10	2.69 ± 0.06	2.83 ± 0.17	3.88 ± 0.01	1.14 ± 0.12
20 Newsgroups	Global Pooling CNN	18.58 ± 0.80	16.49 ± 0.70	14.68 ± 1.03	7.51 ± 0.51	6.13 ± 1.84	5.14 ± 0.64	2.56 ± 0.38
	Average rank	6.4	3.5	6.1	2.8	3.1	4.8	1.3

Table 5.2: Test ECE with temperature scaling (%, $\downarrow$): Our Meta-Calibration (MC) obtains excellent calibration also after temperature scaling.

Dataset	Model	CE	Brier	MMCE	FL-3	FLSD-53	LS	MC (Ours)
CIFAR-10	ResNet18	1.16 ± 0.10 (2.30)	1.23 ± 0.03 (1.00)	1.25 ± 0.08 (2.30)	1.10 ± 0.11 (0.90)	1.48 ± 0.31 (0.87)	1.31 ± 0.08 (0.90)	1.34 ± 0.47 (0.97)
	ResNet50	1.20 ± 0.17 (2.53)	0.97 ± 0.02 (1.17)	1.35 ± 0.38 (2.50)	1.10 ± 0.16 (1.07)	1.21 ± 0.31 (1.07)	1.42 ± 0.15 (0.90)	1.09 ± 0.09 (1.00)
	ResNet110	1.49 ± 0.19 (2.57)	1.55 ± 0.35 (1.13)	1.20 ± 0.45 (1.90)	1.21 ± 0.07 (1.10)	1.33 ± 0.14 (1.10)	2.16 ± 0.21 (0.90)	1.33 ± 0.37 (0.97)
	WideResNet26-10	1.14 ± 0.13 (2.20)	1.03 ± 0.08 (1.00)	0.99 ± 0.19 (2.23)	1.20 ± 0.29 (0.87)	1.09 ± 0.02 (0.90)	1.32 ± 0.04 (0.90)	0.94 ± 0.10 (1.00)
CIFAR-100	ResNet18	5.47 ± 0.22 (1.33)	4.21 ± 0.23 (0.90)	6.09 ± 0.39 (1.13)	2.83 ± 0.27 (1.00)	2.47 ± 0.12 (1.00)	4.37 ± 0.45 (0.90)	2.71 ± 0.58 (1.03)
	ResNet50	2.51 ± 0.23 (1.57)	3.43 ± 0.32 (1.10)	3.19 ± 0.53 (1.37)	2.25 ± 0.69 (1.10)	2.53 ± 0.11 (1.10)	4.28 ± 0.42 (1.10)	2.51 ± 0.45 (1.07)
	ResNet110	3.77 ± 0.51 (1.57)	3.71 ± 0.67 (1.17)	2.74 ± 0.45 (1.40)	3.97 ± 0.28 (1.10)	4.13 ± 0.40 (1.10)	6.04 ± 0.31 (1.10)	2.55 ± 0.33 (1.03)
	WideResNet26-10	3.08 ± 0.26 (1.80)	2.49 ± 0.13 (1.10)	4.52 ± 0.52 (1.40)	2.20 ± 0.12 (1.03)	2.30 ± 0.36 (1.00)	3.62 ± 0.74 (1.07)	2.72 ± 0.19 (1.10)
SVHN	ResNet18	0.74 ± 0.04 (2.10)	0.83 ± 0.09 (0.90)	1.10 ± 0.01 (2.30)	0.90 ± 0.43 (0.83)	1.11 ± 0.37 (0.87)	1.45 ± 0.51 (0.87)	1.14 ± 0.12 (1.00)
20 Newsgroups	Global Pooling CNN	2.85 ± 0.34 (3.67)	4.32 ± 0.79 (2.97)	4.00 ± 0.22 (2.60)	3.59 ± 0.34 (1.43)	2.76 ± 0.20 (1.33)	3.19 ± 0.30 (1.10)	2.50 ± 0.32 (0.97)
	Average rank	3.7	3.8	4.4	3.0	3.9	6.2	2.8

Table 5.3: Test error (%, $\downarrow$): Our Meta-Calibration (MC) obtains excellent calibration with only small increases in the test error.

Dataset	Model	CE	Brier	MMCE	FL-3	FLSD-53	LS	MC (Ours)
CIFAR-10	ResNet18	4.99 ± 0.14	5.27 ± 0.21	5.17 ± 0.19	5.06 ± 0.09	5.22 ± 0.04	4.94 ± 0.13	5.22 ± 0.06
	ResNet50	4.90 ± 0.02	5.15 ± 0.14	5.13 ± 0.12	5.27 ± 0.22	5.26 ± 0.15	4.77 ± 0.11	5.46 ± 0.05
	ResNet110	5.40 ± 0.10	5.97 ± 0.17	5.70 ± 0.12	5.67 ± 0.33	5.87 ± 0.13	5.45 ± 0.11	6.09 ± 0.22
	WideResNet26-10	3.99 ± 0.07	4.20 ± 0.03	4.11 ± 0.06	4.18 ± 0.03	4.22 ± 0.05	4.05 ± 0.07	4.36 ± 0.20
CIFAR-100	ResNet18	22.85 ± 0.17	23.50 ± 0.17	23.80 ± 0.18	22.87 ± 0.16	23.23 ± 0.32	22.35 ± 0.27	23.88 ± 0.20
	ResNet50	22.41 ± 0.24	24.81 ± 0.33	22.43 ± 0.05	22.27 ± 0.13	22.76 ± 0.27	21.85 ± 0.06	23.22 ± 0.48
	ResNet110	22.99 ± 0.19	28.29 ± 1.42	23.81 ± 0.58	23.12 ± 0.26	23.71 ± 0.24	23.08 ± 0.15	24.51 ± 0.41
	WideResNet26-10	20.41 ± 0.12	20.77 ± 0.05	20.60 ± 0.10	19.80 ± 0.40	19.97 ± 0.25	20.82 ± 0.42	22.35 ± 0.03
SVHN	ResNet18	4.11 ± 0.08	3.90 ± 0.19	4.15 ± 0.08	4.20 ± 0.07	4.18 ± 0.06	4.13 ± 0.09	4.08 ± 0.02
20 Newsgroups	Global Pooling CNN	26.64 ± 0.27	26.59 ± 0.72	26.92 ± 0.32	27.65 ± 0.38	27.59 ± 0.94	26.10 ± 0.31	27.26 ± 0.59
	Average rank	2.1	4.9	4.2	3.9	4.8	2.1	5.9

5.4.3 Further Analysis

Alternative Calibration Metrics We investigate if models meta-trained using DECE also perform well when evaluated using more advanced calibration metrics than ECE. In particular, we evaluate performance using class-wise ECE (CECE) (Kumar et al., 2019; Widmann et al., 2019; Vaicenavicius et al., 2019; Kull et al., 2019) that considers the scores of all classes in the predicted distribution, instead of only the class with the highest probability. The results in Table 5.4 confirm that models meta-trained using DECE have excellent calibration also in terms of the CECE criterion.

Table 5.4: Test Classwise-ECE (%, $\downarrow$): Our Meta-Calibration (MC) leads to excellent calibration also when using a more advanced calibration metric.

Dataset	Model	CE	Brier	MMCE	FL-3	FLSD-53	LS-0.05	MC (Ours)
CIFAR-10	ResNet18	0.87 ± 0.04	0.46 ± 0.02	0.91 ± 0.03	0.52 ± 0.02	0.53 ± 0.03	0.73 ± 0.01	0.41 ± 0.01
	ResNet50	0.88 ± 0.01	0.46 ± 0.02	0.93 ± 0.04	0.44 ± 0.02	0.44 ± 0.03	0.63 ± 0.02	0.48 ± 0.04
	ResNet110	0.99 ± 0.02	0.58 ± 0.04	0.89 ± 0.14	0.50 ± 0.02	0.55 ± 0.05	0.66 ± 0.03	0.45 ± 0.01
	WideResNet	0.71 ± 0.01	0.37 ± 0.00	0.74 ± 0.01	0.43 ± 0.02	0.43 ± 0.04	0.72 ± 0.02	0.34 ± 0.00
CIFAR-100	ResNet18	0.23 ± 0.01	0.24 ± 0.00	0.22 ± 0.01	0.20 ± 0.00	0.20 ± 0.00	0.26 ± 0.00	0.19 ± 0.00
	ResNet50	0.29 ± 0.03	0.20 ± 0.01	0.24 ± 0.03	0.20 ± 0.00	0.20 ± 0.01	0.21 ± 0.00	0.19 ± 0.00
	ResNet110	0.34 ± 0.02	0.23 ± 0.01	0.29 ± 0.02	0.22 ± 0.01	0.23 ± 0.00	0.26 ± 0.00	0.19 ± 0.00
	WideResNet	0.29 ± 0.02	0.19 ± 0.00	0.23 ± 0.03	0.18 ± 0.00	0.18 ± 0.00	0.21 ± 0.01	0.19 ± 0.00
SVHN	ResNet18	0.62 ± 0.02	0.52 ± 0.02	0.65 ± 0.02	0.67 ± 0.03	0.68 ± 0.04	0.81 ± 0.05	0.29 ± 0.03
20 Newsgroups	Global Pooling CNN	2.01 ± 0.07	1.80 ± 0.04	1.63 ± 0.09	1.29 ± 0.03	1.22 ± 0.14	0.97 ± 0.06	1.00 ± 0.04
	Average rank	6.0	3.3	5.8	2.5	2.8	5.1	1.6

Alternative Hyperparameter Choice We present a general metric that can be used for optimizing hyperparameters for superior calibration. While our main experiments are conducted with non-uniform label smoothing, we demonstrate the generality of the framework by also learning alternative meta-parameters. In particular, we also consider scalar and vector version of label smoothing as well as learnable L2 regularisation. We perform the additional evaluation using ResNet18 on the CIFAR benchmark.

The results in Table 5.5 confirm learnable L2 regularisation also leads to clear improvement in ECE over the cross-entropy baseline. However, the error rate is slightly increased compared to learnable LS, hence we focused on the latter for our other experiments. Scalar and vector LS (MC-S and MC-V) have both improved the calibration, but non-uniform label smoothing (MC) has worked better thanks to its larger expressivity.

Ablation Study on Meta-Learning Objective Design Recall our framework in Equation 5.1 is setup to perform conventional model training in the inner optimization, given hyperparameters; and meta-learning of hyperparameters in the outer optimization,

Table 5.5: Comparison of hyperparameter choice for meta-calibration: CIFAR benchmark with ResNet18 model. Test errors (%, $\downarrow$) and test ECE (%, $\downarrow$). Other variants of Meta-Calibration also lead to strong improvements in calibration, with non-uniform label smoothing leading to the best calibration overall.

Dataset	Method	ECE ($\downarrow$)	Error ($\downarrow$)
CIFAR-10	CE	4.23 ± 0.15	4.99 ± 0.14
	MC	1.17 ± 0.26	5.22 ± 0.06
	MC-S	1.48 ± 0.26	5.17 ± 0.13
	MC-V	1.51 ± 0.26	5.07 ± 0.03
	MC-L2	1.78 ± 0.22	5.49 ± 0.14
CIFAR-100	CE	8.79 ± 0.59	22.85 ± 0.17
	MC	2.52 ± 0.35	23.88 ± 0.20
	MC-S	6.13 ± 1.20	24.07 ± 0.17
	MC-V	3.98 ± 0.23	23.96 ± 0.12
	MC-L2	4.18 ± 0.26	26.10 ± 0.14

by minimising a combination of cross-entropy and our DECE metric as evaluated on the meta-validation set. While we view this setup as being the most intuitive, other architectures are also possible in terms of choice of objective for use in the inner and outer layer of the bilevel optimization. As a comparison to our DECE, we also evaluate the prior metric MMCE previously proposed as a proxy for model calibration in (Kumar et al., 2018).

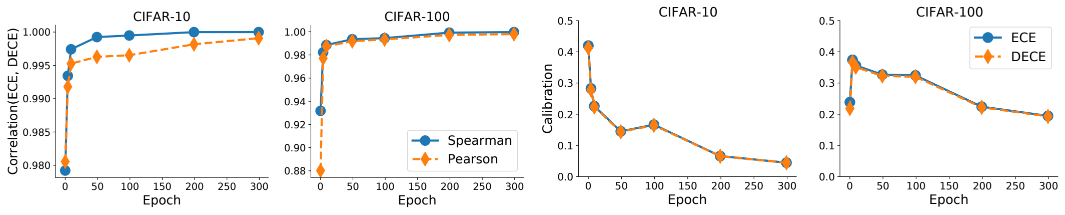
From the results in Table 5.6 we can conclude that: 1) Meta-learning with combined CE and DECE meta-objective is beneficial for improving calibration (M5 vs M0). 2) Alternative outer-loop objectives CE (M2) and DECE (M3) improve calibration but not as significantly as the combined meta-objective (M5 vs M2 and M5 vs M4). 3) MMCE completely fails as a meta-objective (M3). 4) DECE improves calibration when used as a secondary loss in multi-task learning, but at greater detriment to test error (M1 vs M0). 5) Our combined meta-objective (M5) is the best overall.

Evaluating DECE Approximation to ECE A key contribution of this work is DECE, a differentiable approximation to expected calibration error. In this section we investigate the quality of our DECE approximation. We trained the same ResNet18 backbone on both CIFAR-10 and CIFAR-100 benchmarks for 350 epochs, recording DECE and ECE values at various points. The results in Figure 5.2a show both Spearman and Pearson correlation coefficient between DECE and ECE. In both cases they are close to 1, and become even closer to 1 as training continues. This shows that DECE accurately estimates ECE, while providing differentiability for end-to-end optimization.

Table 5.6: Ablation study on losses for inner and outer objectives in bilevel optimization using CIFAR-10 and CIFAR-100 with ResNet18.

			CIFAR-10		CIFAR-100	
Model	Meta-Loss	Loss	ECE (%, $\downarrow$)	Error (%, $\downarrow$)	ECE (%, $\downarrow$)	Error (%, $\downarrow$)
M0: Vanilla CE	-	CE	4.23 ± 0.15	4.99 ± 0.14	8.79 ± 0.59	22.85 ± 0.17
M1: Multi-task	-	CE + DECE	3.80 ± 0.03	10.24 ± 0.21	4.40 ± 0.39	29.49 ± 0.17
M2: Meta-Calibration	CE	CE	1.31 ± 0.36	5.13 ± 0.24	3.00 ± 1.12	23.72 ± 0.40
M3: Meta-Calibration	MMCE	CE	44.24 ± 0.70	6.77 ± 0.25	21.94 ± 2.39	25.40 ± 0.31
M4: Meta-Calibration	DECE	CE	1.26 ± 0.44	5.21 ± 0.14	3.28 ± 0.31	23.83 ± 0.14
M5: Meta-Calibration	CE + DECE	CE	1.17 ± 0.26	5.22 ± 0.06	2.52 ± 0.35	23.88 ± 0.20

We further we show in Figure 5.2b that their mean values are very close to each other.



(a) Correlation between DECE and ECE is close to 1.

(b) Mean ECE and DECE are close to each other.

Figure 5.2: Evaluation of how DECE approximates ECE, using ResNet18 on CIFAR.

What Hyperparameters are Learned? We show our approach learns non-trivial hyperparameter settings to achieve its excellent calibration performance. Figure 5.3 shows how the learned overall strength of smoothing evolves during training for both CIFAR-10 and CIFAR-100 benchmarks – using ResNet18. We show the mean and standard deviation across 3 repetitions and all classes.

From the figure we observe label smoothing changes in response to changes in learning rate, which happens after 150 and 250 epochs. For CIFAR-100 with more classes it starts with large smoothing values and finishes with smaller values. The large standard deviations are due to the model making use of a wide range of class-wise smoothing parameters. It would be infeasible to manually select a curriculum for label smoothing at different stages of training, as it would be to tune a range of smoothing parameters: The ability to optimize these hyperparameters automatically is a key benefit of our framework.

We also analyse how the smoothing is distributed across the different classes in Figure 5.4 and 5.5. The results show that the smoothing is indeed non-uniform, demon-

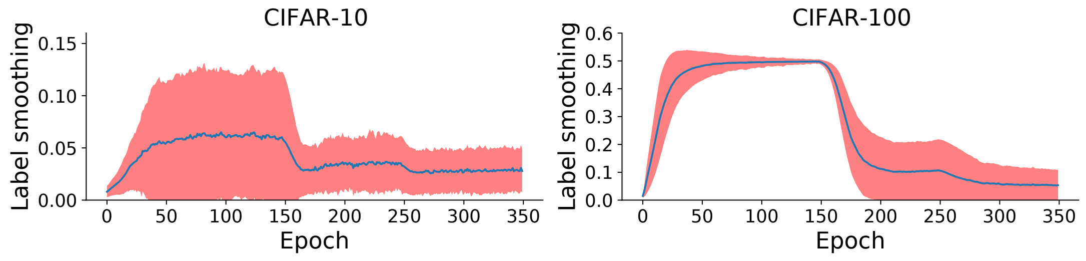
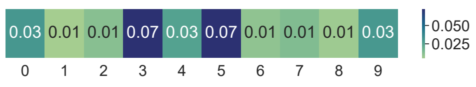
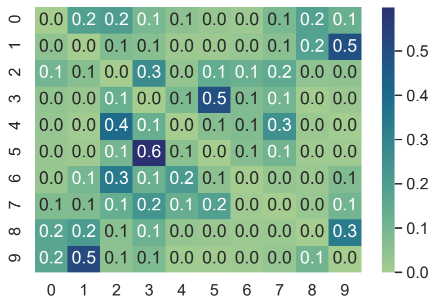


Figure 5.3: Overall label smoothing during training for CIFAR, using ResNet18. The learned smoothing strategy is non-trivial and adapts according to the learning rate schedule.

strating the model does exploit the ability to learn a complex label-smoothing distribution. The learned non-uniform label-smoothing distribution can be observed to subject visually similar classes to more smoothing (Figure 5.4(b)), which makes sense to reduce the confidence of the most likely kinds of specific errors. This idea is quantified more systematically for CIFAR-100 in Figure 5.5, which compares the average degree of smoothing between classes in the same superclass, and those in different superclasses. The results show that within-superclass smoothing is generally much stronger than across-superclass smoothing, even though the model receives no annotation or supervision about superclasses. It learns this smoothing structure given the objective of optimizing (meta-)validation calibration.



(a) Overall level of smoothing in various classes.



(b) Distribution of smoothing across various classes.

Figure 5.4: Analysis of learned non-uniform label smoothing for CIFAR-10, using ResNet18 model. Visually similar classes receive more smoothing – e.g. cat and dog (3 and 5), and automobile and truck (1 and 9).

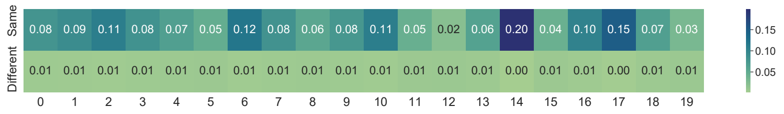


Figure 5.5: Meta-Calibration learns to give more label smoothing within the same superclass compared to other superclasses in CIFAR-100, using ResNet18 model.

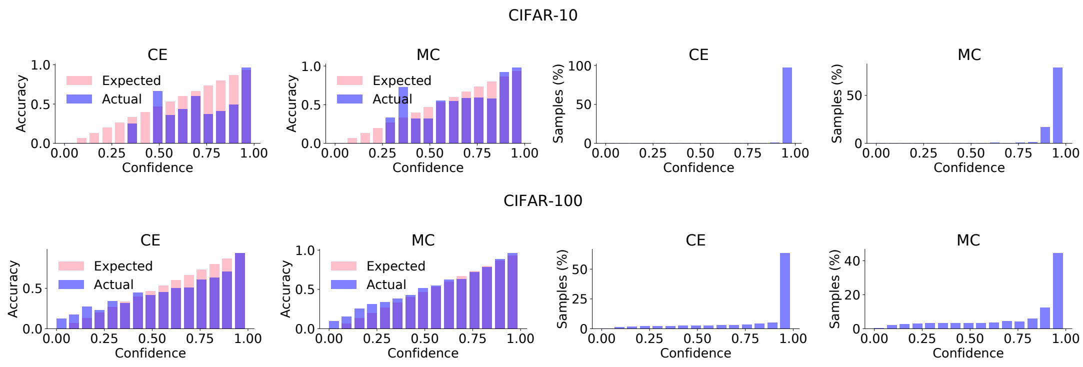


Figure 5.6: Reliability analysis for CIFAR-10 and CIFAR-100, using ResNet18 model.

We further analysed the hyperparameters in the case of learnable L2 regularization and show it in Appendix C. The figure shows we learn a range of regularization values to achieve a good calibration outcome. This highlights the value of our differentiable framework that enables efficient gradient-based optimization of many hyperparameters.

Reliability Analysis We perform reliability analysis and show the percentage of samples with various confidence levels in Figure 5.6 for CIFAR-10 and CIFAR-100. We use ResNet18 and take the best model from training – early stopping. The figure shows learnable label smoothing leads to visually better alignment between the expected and actual confidence binning. It also leads to softening the confidences of predictions, which is expected for label smoothing.

Analysis of Accuracy vs Calibration Using Simulated Data Meta-Calibration leads to significantly better calibration, while keeping similar or only marginally worse accuracy. To study if there is a trade-off between accuracy vs calibration we perform an experiment using simulated data. More specifically the experiment with simulated data consists of 1) sampling parameters of a binary logistic regression model (oracle) over 2D data, 2) sampling the label for each data point from a binomial distribution with a class probability given by the oracle. 3) We then use the sampled data to train (i) a

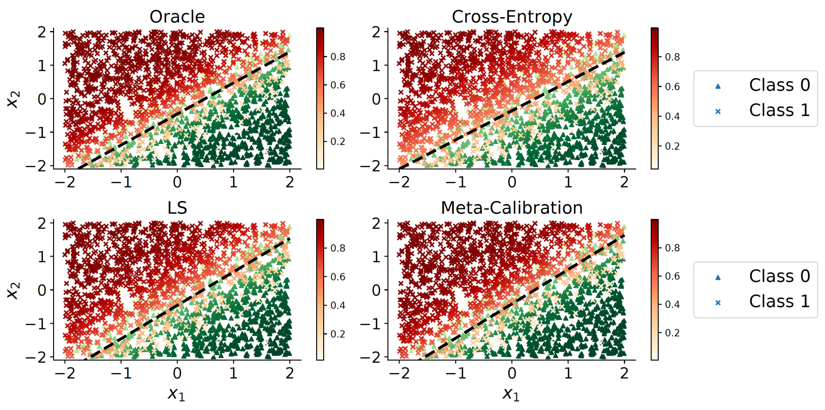


Figure 5.7: Visualization of the estimated probabilities across test data points.

vanilla logistic regression model, (ii) label-smoothing logistic regression model and (iii) a Meta-Calibration logistic regression model. This enables us to compare the best-case classifier accuracy and calibration with the results of learned models. Our experiment is repeated across 3 random seeds so that we can report the mean and standard deviation.

The results in Table 5.7 confirm Meta-Calibration matches both the test accuracy and ECE of the oracle, obtaining close to the best possible calibration. Analogous cross-entropy training as well as simple label smoothing (LS) have a significantly larger ECE. The ECE of Meta-Calibration and oracle is close to 0, but not precisely 0 due to sampling effects (i.e. because we do not use an infinite amount of data). We have also visualized the estimated class probabilities of different data points, together with the decision boundary. The visualization shows that Meta-Calibration and the oracle are similarly calibrated (e.g., similar point shades close to the decision boundary), while a difference in calibration (point shading) is perceptible for cross-entropy and LS.

Table 5.7: Analysis of accuracy and calibration on simulated data with known oracle. Test accuracy and ECE (%) are reported, with the mean and standard deviation computed across 3 random seeds. Our Meta-Calibration (MC) matches both the accuracy and ECE of the oracle, obtaining close to perfect calibration.

Metric	Oracle	Cross-Entropy	LS	MC (Ours)
Accuracy (↑)	87.62 ± 1.87	87.55 ± 1.86	87.57 ± 1.87	87.55 ± 1.81
ECE (↓)	1.38 ± 0.32	9.81 ± 1.14	3.61 ± 0.43	1.40 ± 0.19

5.5 Discussion

This work is a pioneering step in using meta-learning to directly optimize model calibration. Learnable rather than hand-tuned calibration is important as different models and datasets have very different calibration properties, precluding a one-size-fits-all solution (Minderer et al., 2021). There are many ways our work could be extended in the future. One direction is to target different hyperparameters beyond the label smoothing and L2 classifier regularization evaluated here, such as loss learning. While we did not explore this here, the framework could also be used to unify post-hoc methods such as temperature scaling by treating temperature as the hyperparameter. Secondly, better meta-learning algorithms such as implicit meta-learning (Lorraine et al., 2020) could better optimize the meta-objective. A third direction is to extend the differentiable metric itself to e.g. adapt it to various domain-specific calibration measures – for example ones relevant to the finance industry (Liu et al., 2019b).

A drawback of our approach is the computational overhead added by meta-learning compared to basic model training. However, it is still manageable and may be worth it when well-calibrated models are crucial. We give an overview in Table C.1 in Appendix C. Ongoing advances in meta-learning algorithms (Lorraine et al., 2020; Bohdal et al., 2021) can make the overhead smaller. In terms of social implications, our work aims to improve the reliability of neural networks, but there still are risks the neural networks will fail to accurately estimate their confidence.

Our EvoGrad approach is well-suited to be the underlying meta-learning method for Meta-Calibration to decrease its computational costs. PASHA could be helpful for scalar or vector variations of learnable label smoothing, but it is unlikely to work well for non-uniform label smoothing or learnable L2 regularization as these cases have many meta-parameters. Further, PASHA does not support the dynamically changing value of meta-parameters, which may also contribute to the success of Meta-Calibration.

5.6 Further Developments

Meta-Calibration has been utilized in a number of ways. Inspired by our work, researchers have started looking at calibration as a meta-learning problem, with for example (Yang et al., 2022; Wang and Golebiowski, 2023; Iwata and Kumagai, 2023) following our meta-learning of calibration framework. Yang et al. (2022) have applied our framework in a few-shot learning setting, while Wang and Golebiowski

(2023) have targetted the same general setup as us and used a different type of meta-parameters. Iwata and Kumagai (2023) have extended the concept to calibration of Gaussian processes in regression settings. Our DECE metric has also been utilized by other researchers. Emde et al. (2023) have utilized DECE in the context of certified calibration, which is important when encountering adversarial attacks. Marques et al. (2023) have used DECE as part of their Generalized Learned Fusion Strategy that targets the problem of overconfidence in semantic 3D mapping. Parts of our differentiable calibration objective have also been repurposed, with Huang et al. (2023) using our differentiable accuracy part of the objective. Additionally Jiang and Deng (2023) compare with our method among others, and (Wang et al., 2022a; Patra et al., 2023; Tao et al., 2023a,b) are a few of the papers that mention Meta-Calibration as a recent approach for calibration. Meta-Calibration has also been included in a survey mapping state-of-the-art approaches for calibration in deep learning (Wang, 2023).

The broader area of uncertainty calibration has also seen progress. Park et al. (2023) have analysed existing regularization-based approaches for uncertainty calibration and shown these can be interpreted as variants of label smoothing. Based on this insight, they have developed a new loss function that includes adaptive and conditional label smoothing, for example with the smoothing proportional to the logit values. With such approach, they have obtained state-of-the-art uncertainty calibration results on both classification and segmentation problems. Uncertainty calibration is being increasingly often studied for object detection, with Munir et al. (2023) proposing a new training-time loss and Popordanoska et al. (2024) introducing a new differentiable estimator of detection calibration error.

Uncertainty calibration is a topic that is important also for foundational models such as CLIP (Radford et al., 2021). Levine et al. (2023) have studied the calibration of large vision-language models in the zero-shot inference mode, and they found models such as CLIP are miscalibrated. As a solution they proposed a variant of temperature scaling. Oh et al. (2023) have studied the calibration of large vision-language models under distribution shift after fine-tuning, similarly discovering insufficient calibration. Their proposed solution is called calibrated robust fine-tuning and utilizes a variation of label smoothing. The area of uncertainty calibration for foundational models is in early stages, but it will likely become a popular direction because of its practical importance and the widespread use of foundational models.

5.7 Conclusions

We introduced a new DECE metric that accurately represents the common calibration ECE measure and makes it differentiable. With DECE, we can directly optimize hyperparameters for calibration and obtain competitive results with hand-designed architectures. We believe DECE opens up a new avenue for the community to tackle the challenge of model calibration in optimization-based ways.

***

Uncertainty calibration of neural networks is one useful application of meta-learning. As another application where meta-learning is beneficial we introduce the problem of general-purpose few-shot learning. While meta-learning has been widely used in single-task few-shot learning, we make the extensions needed to generalize existing approaches across various computer vision task families.

Chapter 6

Meta Omnium: General-Purpose Learning-to-Learn

The content of this chapter corresponds to paper:
Meta Omnium: A Benchmark for General-Purpose Learning-To-Learn
Ondrej Bohdal, Yinbing Tian, Yongshuo Zong, Ruchika Chavhan, Da Li, Henry Gouk, Li Guo, Timothy Hospedales
Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023

Meta-learning and other approaches to few-shot learning are widely studied for image recognition, and are increasingly applied to other vision tasks such as pose estimation and dense prediction. This naturally raises the question of whether there is any few-shot meta-learning algorithm capable of generalizing across these diverse task types? To support the community in answering this question, we introduce Meta Omnium, a dataset-of-datasets spanning multiple vision tasks including recognition, keypoint localization, semantic segmentation and regression. We experiment with popular few-shot meta-learning baselines and analyze their ability to generalize across tasks and to transfer knowledge between them. Meta Omnium enables meta-learning researchers to evaluate model generalization to a much wider array of tasks than previously possible, and provides a single framework for evaluating meta-learners across a wide suite of vision applications in a consistent manner.

6.1 Introduction

Meta-learning is a long-standing research area that aims to replicate the human ability to learn from a few examples by learning-to-learn from a large number of learning problems (Thrun and Pratt, 1998). This area has become increasingly important recently, as a paradigm with the potential to break the data bottleneck of traditional supervised learning (Hospedales et al., 2021; Wang et al., 2020). While the largest body of work is applied to image recognition, few-shot learning algorithms have now been studied in most corners of computer vision, from semantic segmentation (Li et al., 2020c) to pose estimation (Patacchiola et al., 2020) and beyond. Nevertheless, most of these applications of few-shot learning are advancing independently, with increasingly divergent application-specific methods and benchmarks. This makes it hard to evaluate whether few-shot meta-learners can solve diverse vision tasks. Importantly it also discourages the development of meta-learners with the ability to learn-to-learn across tasks, transferring knowledge from, e.g., keypoint localization to segmentation – a capability that would be highly valuable for vision systems if achieved.

Table 6.1: Feature comparison between Meta Omnium and other few-shot meta-learning benchmarks. Meta Omnium uniquely combines a rich set of tasks and visual domains with a lightweight size for accessible use.

Dataset	Num Tasks	Num Domains	Num Imgs	Categories	Size	Lightweight	Multi-Task	Multi-Domain
Omniglot (Lake et al., 2015)	1	1	32K	1623	148MB	✓	✗	✗
MiniImageNet (Vinyals et al., 2016)	1	1	60K	100	1GB	✓	✗	✗
Meta-Dataset (Triantafillou et al., 2020)	1	7∼10	53M	43∼1500	210GB	✗	✗	✓
VTAB (Zhai et al., 2019)	1	3∼19	2.2M	2∼397	100GB	✗	✗	✓
FSS1000 (Li et al., 2020c)	1	1	10000	1000	670MB	✓	✗	✗
Meta-Album (Ullah et al., 2022)	1	10∼40	1.5M	19∼706	15GB	✓	✗	✓
Meta Omnium	4	21	160K	2∼706	3.1GB	✓	✓	✓

The overall trend in computer vision (Ghiasi et al., 2021; Radford et al., 2021) and AI (Reed et al., 2022; Baevski et al., 2022) more generally is towards more general-purpose models and algorithms that support many tasks and ideally leverage synergies across them. However, it has not yet been possible to explore this trend in meta-learning, due to the lack of few-shot benchmarks spanning multiple tasks. State-of-the-art benchmarks (Triantafillou et al., 2020; Ullah et al., 2022) for visual few-shot learning are restricted to image recognition across a handful of visual domains. There is no few-shot benchmark that poses the more substantial challenge (Yu et al., 2020; Ruder, 2017) of generalizing across different *tasks*. We remark that the term *task* is used differently in few-shot meta-learning literature (Finn et al., 2017; Wang et al., 2020; Hospedales et al., 2021)

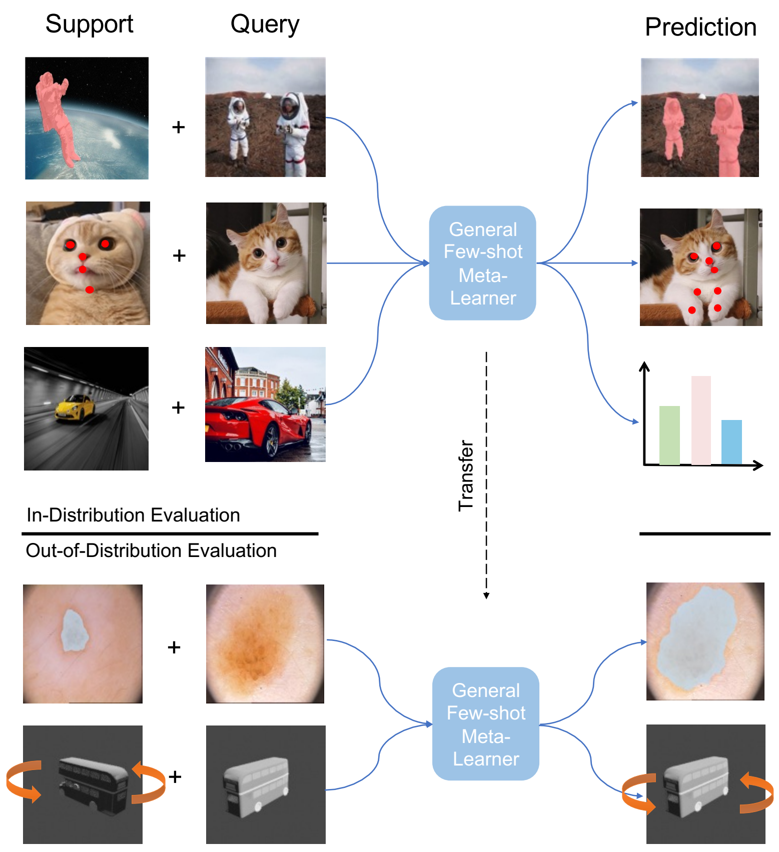

Support
Query
Prediction
General Few-shot Meta-Learner
Transfer
In-Distribution Evaluation
Out-of-Distribution Evaluation
General Few-shot Meta-Learner

Figure 6.1: Illustration of the diverse visual domains and task types in Meta Omnium. Meta-learners are required to generalize across multiple task types, multiple datasets, and held-out datasets.

(to mean different image recognition problems, such as cat vs dog or car vs bus) and the multi-task literature (Yu et al., 2020; Ruder, 2017; Ghiasi et al., 2021; Yang and Hospedales, 2015) (to mean different kinds of image understanding problems, such as classification vs segmentation). In this chapter, we will use the term *task* in the multi-task literature sense, and the term *episode* to refer to tasks in the meta-learning literature sense, corresponding to a support and query set.

We introduce Meta Omnium, a dataset-of-datasets spanning multiple vision tasks including recognition, semantic segmentation, keypoint localization/pose estimation, and regression as illustrated in Figure 6.1. Specifically, Meta Omnium provides the following important contributions: 1) Existing benchmarks only test the ability of meta-learners to learn-to-learn within tasks such as classification (Triantafillou et al., 2020; Ullah et al., 2022), or dense prediction (Li et al., 2020c). Meta Omnium uniquely tests the ability of meta-learners to learn across multiple task types. 2) Meta Omnium covers multiple visual domains (from natural to medical and industrial images). 3) Meta

Omnium provides the ability to thoroughly evaluate both in-distribution and out-of-distribution generalization. 4) Meta Omnium has a clear hyperparameter tuning (HPO) and model selection protocol, to facilitate future fair comparison across current and future meta-learning algorithms. 5) Unlike popular predecessors, (Triantafillou et al., 2020), and despite the diversity of tasks, Meta Omnium has been carefully designed to be of moderate computational cost, making it accessible for research in modestly-resourced universities as well as large institutions. Table 6.1 compares Meta Omnium to other relevant meta-learning datasets.

We expect Meta Omnium to advance the field by encouraging the development of meta-learning algorithms capable of knowledge transfer across different tasks – as well as across learning episodes within individual tasks as is popularly studied today (Finn et al., 2017; Wang et al., 2020). In this regard, it provides the next step of the level of a currently topical challenge of dealing with heterogeneity in meta-learning (Triantafillou et al., 2020; Vuorio et al., 2019; Abdollahzadeh et al., 2021; Li et al., 2021a). While existing benchmarks have tested *multi-domain* heterogeneity (e.g., recognition of written characters and plants within a single network) (Triantafillou et al., 2020; Ullah et al., 2022) and shown it to be challenging, Meta Omnium tests *multi-task learning* (e.g., character recognition vs plant segmentation). This is substantially more ambitious when considered from the perspective of common representation learning. For example, a representation tuned for recognition might benefit from rotation *invariance*, while one tuned for segmentation might benefit from rotation *equivariance* (Ericsson et al., 2022; Xiao et al., 2021; Chavhan et al., 2023). Thus, in contrast to conventional within-task meta-learning benchmarks that have been criticized as relying more on common representation learning than learning-to-learn (Tian et al., 2020; Raghu et al., 2020), Meta Omnium better tests the ability of learning-to-learn since the constituent tasks require more diverse representations. Code and dataset are available at `https://edi-meta-learning.github.io/meta-omnium`.

6.2 Related Work

6.2.1 Meta-Learning Benchmarks

The classic datasets in few-shot meta-learning for computer vision are Omniglot (Lake et al., 2015) and miniImageNet (Vinyals et al., 2016). Later work criticized these for having insufficient task (episode) diversity and tieredImageNet (Ren et al., 2018a) used the class hierarchy of ImageNet to enforce more diversity between meta-train and meta-test episodes. The main contemporary benchmarks are CD-FSL (Guo et al., 2020), which challenges few-shot learners to generalize to new visual domains; and Meta-Dataset (Triantafillou et al., 2020) and Meta-Album (Ullah et al., 2022), which go further in requiring few-shot learners to learn from a mixture of visual domains. Such multi-domain heterogeneous meta-learning turns out to be challenging. A related benchmark to Meta-Dataset is VTAB (Zhai et al., 2019), which similarly provides multiple domains for evaluating data-efficient visual recognition, but their focus is on evaluating representation transfer from large-scale pre-training rather than learning-to-learn and meta-learning. VTAB+MD (Dumoulin et al., 2021) compare representation transfer and meta-learning approaches on the Meta-Dataset tasks. However, none of these benchmarks address multi-task meta-learning as considered here (Figure 6.1).

Outside of recognition, task-specific few-shot benchmarks have been proposed in vision problems of semantic segmentation (Li et al., 2020c), regression (Gao et al., 2022b), pose/keypoint estimation (Xu et al., 2022), etc. These are mostly slightly behind the complexity of the recognition benchmarks with regards to being single-domain, with the exception of (Xu et al., 2022). With regard to multi-task meta-learning as considered here, the only existing benchmark is Meta-World (Yu et al., 2019), which is specific to robotics and reinforcement learning rather than vision.

We also mention Taskonomy (Zamir et al., 2018) as a popular dataset that has been used for multi-task learning. However, it is not widely used for few-shot meta-learning. This is because, although Taskonomy has many tasks, unlike the main meta-learning benchmarks (Triantafillou et al., 2020; Li et al., 2020c), there are not enough visual concepts within each task to provide a large number of concepts for meta-training and a disjoint set of concepts to evaluate few-shot learning for meta-validation and meta-testing.

6.2.2 Heterogeneity in Meta-Learning

There are several sophisticated methods in the literature that highlighted the challenge of the task to address heterogeneity in meta-learning. These have gone under various names such as multi-modal meta-learning (Vuorio et al., 2019; Abdollahzadeh et al., 2021; Liu et al., 2021) – in the sense of multi-modal probability distribution (over tasks/episodes). However, with the exception of (Liu et al., 2021), these have mostly not been shown to scale to the main multi-modal benchmarks such as Meta-Dataset

(Triantafillou et al., 2020). A more common approach to achieving high performance across multiple heterogeneous domains such as those in Meta-Dataset is to train an ensemble of feature extractors across available training domains, and fuse them during meta-testing (Dvornik et al., 2020; Li et al., 2021b,b). However, this obviously incurs a substantial additional cost of maintaining a model ensemble. In our evaluation, we focus on the simpler meta-learners that have been shown to work in challenging multi-domain learning scenarios (Triantafillou et al., 2020; Ullah et al., 2022), while leaving sophisticated algorithmic and ensemble-based approaches for future researchers to evaluate on the benchmark.

6.3 Meta Omnium Benchmark and Datasets

6.3.1 Motivation and Guiding Principles

We first explain the motivating goals and guiding principles behind the design of Meta Omnium. The goal is to build a benchmark for multi-task meta-learning that will: 1) Encourage the community to develop meta-learners that are flexible enough to deal with greater task heterogeneity than before, and thus are more likely to be useful in practice with less curated episode distributions. This was identified as a major challenge in the discussion arising in several recent meta-learning and computer vision workshops and challenges. 2) Ultimately progress on this benchmark should provide practical improvements in data-efficient learning for computer vision through the development of methods that can better transfer across different task types.

In developing this benchmark, we established a few principles that we used to guide design choices. These include: 1) The benchmark should be lightweight in terms of storage and computing, making it accessible to a broad range of researchers and not only large corporations. 2) The benchmark should cover multiple tasks with heterogeneous output spaces (as opposed to all classification, all regression, or all dense prediction), as well as multiple visual domains. In these regards, Meta Omnium is compared to alternatives in Table 6.1. 3) The initial baselines should have only minimal *task-specific decoders*. This is in contrast to the state of the art within various sub-disciplines of FSL such as segmentation (Min et al., 2021; Hong et al., 2022), keypoint (Lu and Koniusz, 2022; Xiao and Marlet, 2020), and classification (Afrasiyabi et al., 2022; Ye et al., 2020) where specially designed decoders are often used. This is to evaluate and encourage future research on *learning-to-learn* across tasks, rather than primarily benchmarking how well we can manually engineer prior knowledge of optimal task-specific decoders. While we are not opposed to future competitors on this benchmark developing task-specific decoders, these should be evaluated separately against the minimal-decoder competitors. 4) The benchmark should provide distinct datasets for in-distribution (ID) training and out-of-distribution (OOD) evaluation, to evaluate the robustness of the distribution-shift. This is already provided by (Triantafillou et al., 2020; Ullah et al., 2022) for classification, and we extend such an ID and OOD dataset ensemble to multiple tasks. Figure 6.2 illustrates our dataset and task-split. 5) The benchmark should provide a clear hyperparameter tuning protocol. With a number of recent studies showing that hyperparameter tuning can dominate other effects of interest in computer vision (Gulrajani and Lopez-Paz, 2021; Musgrave et al., 2021; Li et al., 2022), this is important for a future-proof meta-learning benchmark. This is also related to the first cost point 1) above: Only for a benchmark with a modest cost can most institutions realistically expect to conduct hyperparameter tuning. We provide the hyperparameter tuning protocol. 6) Finally, following the debate in (Tian et al., 2020; Raghu et al., 2020; Li et al., 2021a) as to the value of meta-learning vs conventional transfer learning, the dataset should support both episodic meta-learning and conventional transfer learning approaches.

6.3.2 Data Splits and Tasks

For each main task (classification, segmentation, keypoint localization), we split the datasets into seen datasets available for meta-training, and unseen datasets that are completely held out for out-of-domain meta-validation and meta-testing. Similarly to (Triantafillou et al., 2020; Ullah et al., 2022), for the seen datasets, we construct category-wise splits into meta-train/val/test. While for the unseen datasets, there is no category-wise split as episodes from all categories from the whole dataset will be used for validation and testing respectively. The overall split organization is illustrated in Figure 6.2. We additionally have a completely held-out task: regression. Datasets from this task are not used during meta-training.

Our multi-task setup enables us to define and compare two training protocols: **Single-task meta-learning** (STL) which evaluates how well meta-learning performs when trained and tested within a particular task family (within each plate in Figure 6.2); and **Multi-task meta-learning** (MTL) which evaluates how well meta-learning performs when trained across all available task families (across all plates in Figure 6.2).

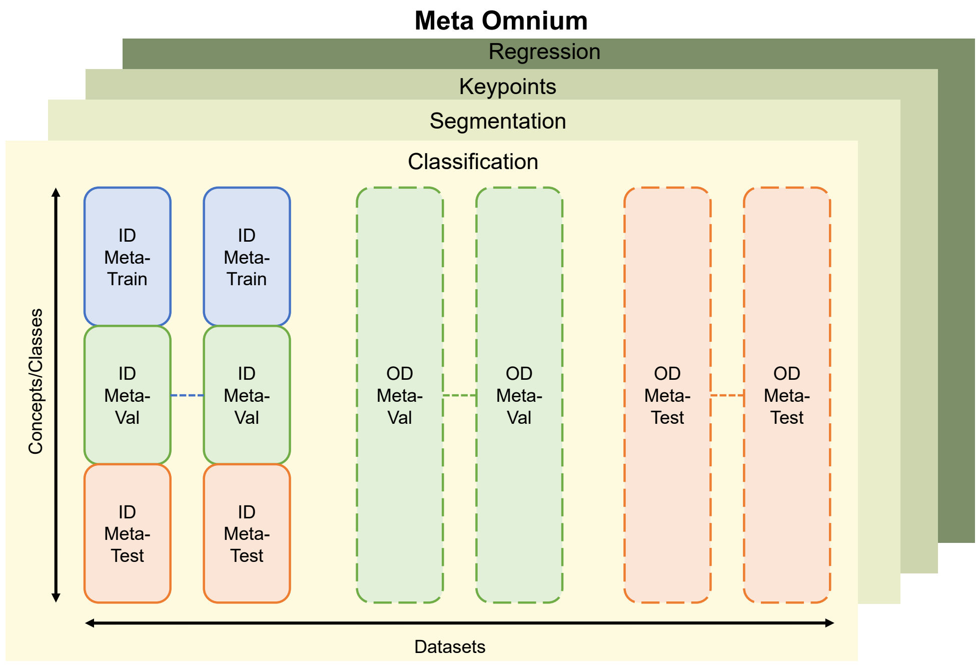


Figure 6.2: Schematic of benchmark and dataset splits. For each task, there are multiple datasets, which are divided into seen (solid border) and unseen (dash border) datasets. The seen (ID) datasets are divided class-wise into meta-train/meta-val/meta-test splits. The unseen datasets are held out for out-distribution (OD) evaluation. Meta-training is conducted on the ID-meta-train split of the seen datasets (blue). Models are validated on ID validation class splits, or OD validation datasets (green). Results are reported on the ID test class splits and OD test datasets (orange). We also hold out an entire task family (regression) for evaluating novel task generalization.

With this organization we can separately evaluate: **Within-distribution generalization (ID)**: How well do meta-learners generalize to novel test concepts within the seen datasets?; and **Out-of-distribution generalization (OOD)**: How well do meta-learners generalize to novel concepts in unseen datasets?

We provide two sources of validation data: ID and OOD, and our models are selected based on the combined performance across both. OOD validation is not supposed by the most popular Meta-Dataset benchmark (Triantafillou et al., 2020) as despite its larger size it does not provide OOD validation datasets.

6.3.3 Datasets and Metrics

Given the considerations in Section 6.3.1, our benchmark consists of three main tasks (classification, segmentation, keypoints/pose) and one held-out task (regression).

Classification For classification we take the 10 datasets from the initial public release of Meta-Album (Ullah et al., 2022). These images are all 128×128 and contain 19–706 classes per dataset, with 40 images per class. Three of these datasets are reserved for out-of-distribution meta-validation, and four for out-of-distribution meta-test.

Segmentation For segmentation, we take FSS1000 (Li et al., 2020c) for in-distribution (10,000 images, 1,000 classes), and combine it with VizWiz (Tseng et al., 2022) for OOD meta-validation (862 images, 22 classes), and modified Pascal5i (Shaban et al., 2017) (7,242 images, 6 classes) and the very distinct medical imaging dataset PH2 (Mendonça et al., 2013) (200 images, 3 classes) for meta-testing. The segmentation images originally were of diverse sizes. We resize them all to 224×224 for Meta Omnium. Note that VizWiz and Pascal datasets originally contain more classes and images. We exclude the classes that overlap with those in the FSS1000 dataset for few-shot learning, and thus there are no classes overlapping among all the datasets.

Keypoints For keypoint (pose) estimation, we take animal-pose (Cao et al., 2019) for in-distribution, synthetic animal-pose (Mu et al., 2020) for OOD meta-validation, and MPII human-pose (Andriluka et al., 2014) for OOD meta-testing. All images are resized to 128×128. MPII includes about 40k people in over 25k images with annotated body keypoints. Animal Pose includes 5 animal categories for 6K instances in over 4k images. Each animal is cropped from the original image. We keep cats and dogs for training, horses and sheep for in-domain validation, and cow for in-domain testing. Synthetic animal pose generates synthetic images using animal CAD models rendered from various viewpoints and lightings on a random background. We keep only the horse and tiger categories in our final datasets.

Regression For evaluating regression as a held-out task, we use four datasets corresponding to the test splits of (Gao et al., 2022b): ShapeNet1D, ShapeNet2D, Distractor and Pascal1D (Yin et al., 2020). All images are resized to 128×128. ShapeNet1D aims to predict azimuth angles. It contains 30 categories in total and we keep the 3 categories from the test set. ShapeNet2D further includes 2D rotation with azimuth angles and elevation. The test set of ShapeNet2D contains 300 categories in total with 30 images per category. Distractor aims to predict the position of a target object in the presence of a distractor. It contains 12 categories in total and the test set has 2 categories. Each category contains 1000 objects with 36 images for each. Pascal1D aims to predict azimuth angle. The whole Pascal1D contains 65 objects from 10 categories. The test set contains 15 objects with 100 images for each object. Appendix D provides full details of all datasets and splits.

6.3.4 Training API

For episodic learning, we proceed by 1) sampling a task, 2) sampling a dataset, 3) sampling an episode. Under our main protocol we consider variable 1 to 5-shot evaluation, but also evaluate separate 1 and 5 shot settings (training is always done with a variable number of shots – support examples). For classification tasks, we follow (Ullah et al., 2022) in generating 5-way episodes. For segmentation tasks, we follow (Li et al., 2020c) in considering each episode to be a binary foreground/background classification problem for a novel class and generate 2-way episodes. For keypoint, we form episodes by randomly selecting a class (e.g., animal category) and then randomly selecting a subset of 5 keypoints to localize for each episode. For regression tasks, we generate variable 5 to 25-shot episodes because it is a common practice to use more shots for regression tasks (Gao et al., 2022b).

For non-episodic/transfer learning, we provide access to the meta-train portions of the seen datasets in conventional mini-batches for conventional single task and multi-task supervised learning.

Evaluation Metrics For classification tasks, we use top-1 accuracy; for segmentation tasks, we use mean intersection-over-union (mIOU) that averages over IoU values of all object classes; for keypoint prediction, we report the Percentage of Correct Keypoints (PCK). In detail, a detected joint is considered correct if the distance between the predicted and the true joint is within a certain threshold. In our experiments, the threshold is 0.01 for normalized value, which stands for about 12.8 pixel of input image resolution. For regression tasks, we follow (Gao et al., 2022b) and use the same metrics.

6.3.5 Architecture and Baseline Competitors

As discussed in Section 6.3.1, we aim to establish baselines that can be adapted to tasks with heterogeneous outputs, with minimal reliance on task-specific decoders. We follow (Triantafillou et al., 2020; Ullah et al., 2022) in using a ResNet18 CNN (He et al., 2016) as a feature extractor architecture. For recognition tasks, we perform multi-class classification immediately after ResNet's Global Average Pooling (GAP). For regression tasks, we perform linear regression directly after the ResNet's GAP. For keypoint tasks, we consider them to be a regression problem from the feature map to the keypoint location. For segmentation tasks, we use a simplified PSPNet-like (Zhao et al., 2017) strategy. We concatenate the extracted feature maps from ResNet's feature pyramid, with upsampling where appropriate, to generate a feature map of size

$w \times h$, and then do pixel-wise classification with 1×1 convolutional layer to obtain the final segmentation map. All tasks thus have only *one* learnable weight as a minimal classifier/decoder after the common ResNet feature encoder. Based on this common encoder and minimal decoder architecture, we describe our meta-learning baselines.

Prototypical Network (Snell et al., 2017) is a classic meta-learner that exploits nearest-centroid metric learning for few-shot classification. ProtoNets were adapted to segmentation tasks in PANet (Wang et al., 2019a), by performing pixel-level feature matching between support prototypes and query pixels. We use the same principle together with the PSPNet-like features described earlier. To generalize ProtoNets to regression tasks such as keypoint prediction, we must relax the prototype assumption, and use them as simple Gaussian kernel-regression models (Bishop, 2006). Specifically, we generate a feature embedding for each support example, and then for query examples, we calculate the negative exponential distance to the support examples, and use this inverse distance-weighted sum of support set labels as the prediction. Thus for regression tasks with a support set $S = \{(x_i, y_i)\}$ and query example x_q, ProtoRegression predicts

$$f(y_q|x_q, S) \propto \sum_i y_i \exp(-\tau(f_\theta(x_i) - f_\theta(x_q))^2) \tag{6.1}$$

enabling us to learn deep feature f_θ in the usual episodic meta-learning way. We use cross-entropy loss for classification and segmentation, and MAE loss for regression tasks.

DDRR Deep differentiable ridge-regression has been considered for few-shot recognition (Bertinetto et al., 2019), tracking (Zheng et al., 2020a), and other tasks. It is related to ProtoNet in that the feature is not adapted after the meta-train stage, but different in that the decoder/classifier layer is learned by differentiable ridge-regression rather than nearest centroid or kernel regression. An elegant property of DDRR methods is that they naturally address regression tasks, although they have been repurposed for classification (Bertinetto et al., 2019) by conducting MSE-loss regression to a target 1-hot vector. Thus they are a natural choice for our benchmark. For application to segmentation, we apply DDRR in a 1×1 convolution-like way, to perform pixel classification for the output mask with a DDRR classifier at each pixel. Further, we calibrate the prediction for binary cross entropy loss with a learnable scale and bias following (Bertinetto et al., 2019). DDRR uses MAE loss for regression tasks and MSE loss for other tasks.

MAML The seminal few-shot meta-learner MAML (Finn et al., 2017) aims to learn an initial condition for per-episode gradient-descent. MAML is straightforward to adapt to different types of tasks. Based on each episode's support set, a new output layer is learned, and the feature extractor is updated, both by a few steps of gradient descent. Similarly to Meta-Dataset (Triantafillou et al., 2020), we do not learn an initialization for the output layer, since it can change size between episodes drawn from multiple tasks. To alleviate this challenge, we also follow Meta-Dataset in evaluating Proto-MAML – a variant that initializes the MAML output layer based on the linear classifier/regressor suggested by nearest-centroid prior to gradient descent. Going beyond this, to adapt Proto-MAML to regression tasks, we also initialize the output layer based on the ridge-regression solution to the support set.

Meta-Curvature Meta-Curvature (Park and Oliva, 2019) is an enhancement of MAML that learns a pre-conditioning matrix to improve inner-loop adaptation, as well as an initial condition as in standard MAML. Meta-Curvature outperforms MAML in simpler single-task few-shot benchmarks.

Transfer Learning We also consider standard supervised learning on the meta-train tasks for transfer to the target tasks, a strategy reported to be competitive with meta-learning (Tian et al., 2020). For adaptation, we explore both linear readout (Wang et al., 2019b; Tian et al., 2020) and fine-tuning (Ullah et al., 2022). Besides learning a new output layer from scratch, we also consider a fine-tuning version that initializes the classifier weights using class prototypes (recognition/segmentation) or ridge regression weights (keypoints/regression), inspired by Proto-MAML.

Train-from-Scratch (TFS) We lastly consider training each episode from scratch using only the support set (Ullah et al., 2022).

6.3.6 Hyperparameter Optimization

As part of our benchmark, we perform hyperparameter optimization (HPO) to ensure we select appropriate hyperparameters for the diverse tasks that we consider. Multiobjective HPO under restricted resources is challenging, so we devise the following HPO protocol: estimate the performance of each candidate configuration on a lower fidelity (lower number of iterations) and then identify the configuration that works the best across all validation datasets considered (combination of in-domain and out-of-domain datasets, across various task families). Note that fast multi-fidelity methods such as Hyperband (Li et al., 2018b), ASHA (Li et al., 2020a) or PASHA (Bohdal et al., 2023a) are not applicable out of the box in our multi-objective setup, so we decided to train each candidate configuration using fixed 5,000 training iterations. Since different tasks/datasets are of different difficulties (and use different metrics), we normalize the score of each configuration for each validation dataset by the best score for that dataset across all candidate configurations. We then select the configuration with the best average normalized score.

Note that due to resource constraints we are only able to sample a relatively smaller number of candidates (30), so we utilize a sample efficient state-of-the-art Multi-Objective TPE method (Ozaki et al., 2020), available from the Optuna library (Akiba et al., 2019). We perform HPO for multi-task and single-tasks setups separately, so single task classification, segmentation and keypoint estimation have their own set of hyperparameters; and the multi-task case has its own set. The hyperparameters include the meta-learning rate and optimizer, momentum, and various method-specific hyperparameters (full details are in Appendix D). Once the hyperparameters are chosen, we perform standard training of the model for the full number of iterations.

6.4 Experiments

In this section, we aim to use our benchmark to answer the following questions: 1) *Which meta-learner performs best on average across a heterogenous range of tasks?* Existing benchmarks have evaluated meta-learners for one task at a time, we now use our common evaluation platform to find out if any meta-learner can provide general-purpose learning to learn across different task types, or whether each task type prefers a different learner. Similarly, we can ask *which meta-learner is most robust to out-of-distribution tasks?* 2) Having defined the first multi-task meta-learning benchmark, and generalizations of seminal meta-learners to different kinds of output spaces, we ask *which meta-learner performs best for multi-task meta-learning?* More generally, *is there a trend in gradient-based vs metric-based meta-learner success?* 3) *Does multi-task meta-learning improve or worsen performance compared to single-task?* The former obviously provides more meta-training data, which should be advantageous, but the increased heterogeneity across meta-training episodes in the multi-task case also makes it harder to learn (Vuorio et al., 2019; Triantafillou et al., 2020). 4) *How does meta-learning perform compared to simple transfer learning, or learning from scratch?*

6.4.1 Experimental Settings

We train each meta-learner for 30,000 meta-training iterations, with meta-validation after every 2,500 iterations (used for checkpoint selection). For evaluation during meta-testing we use 600 tasks for each corresponding dataset, and for meta-validation we use 1200 tasks together. We use random seeds to ensure that the same tasks are used across all methods that we compare. For transfer learning approaches (fine-tuning, training from scratch, etc.), we use 20 update steps during evaluation. We only retain the meta-learned shared feature extractor across tasks, and for each new evaluation task, we randomly initialize the output layer so that we can support any number of classes as well as novel task families during meta-testing (in line with (Ullah et al., 2022)).

6.4.2 Results

The main experimental results are shown in Table 6.2, where rows correspond to different few-shot learners, and columns report the average performance on test episodes, aggregated across multiple datasets in each task family, and broken down by "seen" datasets (ID) and "unseen" datasets (OOD). The table also reports the average rank of each meta-learner across each dataset, both overall and broken down by ID and OOD datasets. More specifically, for each setting (e.g. cls. ID) we calculate the rank of each method (separately for STL and MTL), and then we average those ranks across cls., seg. and keypoints. From the results, we can draw the following main conclusions:

1) *ProtoNet is the most versatile meta-learner*, as shown by its highest average rank in the single-task scenario. This validates our novel Kernel Regression extension of ProtoNet for tackling regression-type keypoint localization tasks. Somewhat surprisingly, *ProtoNet is also the most robust to out-of-distribution episodes* (OOD) which is different from the conclusion of (Triantafillou et al., 2020) and others who suggested that gradient-based adaptation is crucial to adapt to OOD data. However, it is also in line with the results of (Ullah et al., 2022) and the strong performance of prototypes more broadly (Bohdal et al., 2022).

2) Coming to multi-task meta-learning the situation is similar in that ProtoNet dominates the other competitors, but now sharing the first place with Proto-MAML.

3) To compare single-task and multi-task meta-learning (top and bottom blocks of Table 6.2) more easily, Figure 6.3 shows the difference in meta-testing episode performance after STL and MTL meta-training for each method. Overall STL outperforms the MTL condition, showing that the difficulty of learning from heterogeneous tasks

Table 6.2: Main Results. Results are presented as averages across the datasets within each task type and separately for in-distribution (ID) and out-of-distribution (OOD) datasets. Classification, segmentation, and keypoint results are reported in accuracy (%), mIOU (%), and PCK (%) respectively. The upper and lower groups correspond to multi-task and single-task meta-training prior to evaluation on the same set of meta-testing episodes. Upper and lower sub row groups correspond to meta-learners and non-meta learners respectively. See Appendix D for a full breakdown over individual datasets.

		Classification		**Segmentation**		**Keypoints**		**Average Rank**		
		ID	OOD	ID	OOD	ID	OOD	ID	OOD	AVG
Single-Task	MAML	58.7	61.6	54.7	42.1	25.4	33.0	4.3	3.3	3.8
	Proto-MAML	50.5	49.7	46.4	44.1	23.6	22.5	6.0	6.3	6.2
	Meta-Curvature	64.8	61.4	65.6	49.8	43.5	16.0	2.0	4.3	3.2
	ProtoNet	70.4	59.4	75.8	57.2	27.8	33.3	**1.3**	**1.7**	**1.5**
	DDRR	63.1	58.7	66.7	48.0	20.5	31.9	4.7	3.7	4.2
	Proto-FineTuning	50.8	50.7	60.0	43.4	21.3	33.1	5.3	4.3	4.8
	FineTuning	42.3	48.2	50.5	40.0	25.7	30.0	5.7	6.7	6.2
	Linear-Readout	48.6	53.4	34.0	22.7	22.1	26.9	7.3	6.7	7.0
	TFS	31.5	42.0	42.8	37.6	21.0	26.0	8.3	8.0	8.2
Multi-Task	MAML	59.1	58.5	43.3	37.4	24.3	23.9	**2.7**	4.7	3.7
	Proto-MAML	58.5	63.7	53.0	43.2	21.6	33.3	3.0	**1.7**	**2.3**
	Meta-Curvature	70.4	66.9	42.6	34.5	18.2	25.3	4.3	4.7	4.5
	ProtoNet	65.9	58.8	63.3	49.7	20.1	33.0	**2.7**	2.0	**2.3**
	DDRR	52.8	51.9	40.4	37.3	22.8	30.1	5.0	4.7	4.8
	Proto-FineTuning	52.4	53.2	44.8	37.8	21.2	30.0	4.3	4.0	4.2
	FineTuning	44.1	51.2	41.3	36.1	18.1	20.5	7.7	7.0	7.3
	Linear-Readout	46.0	50.9	41.5	32.6	19.9	23.5	6.3	8.0	7.2
	TFS	21.9	23.8	38.7	35.8	14.1	11.0	9.0	8.3	8.7

(Yu et al., 2020; Ruder, 2017) outweighs the benefit of the extra multi-task data.

4) Finally, comparing meta-learning methods with simple transfer learning methods as discussed in (Tian et al., 2020; Dumoulin et al., 2021), the *best meta-learners are clearly better than transfer learning for both single and multi-task scenarios.*

We also note that Proto-MAML is better than MAML in the multi-task case, likely due to the importance of a good output-layer initialization in the case of heterogeneous episodes, as per (Triantafillou et al., 2020). Meta-Curvature outperforms MAML in single-task in-domain scenarios, in line with previous results (Park and Oliva, 2019), but it did not achieve stronger performance out-of-domain or in the multi-task case. Finally, while DDRR is perhaps the most elegant baseline in terms of most naturally spanning all task types, its overall performance is middling.

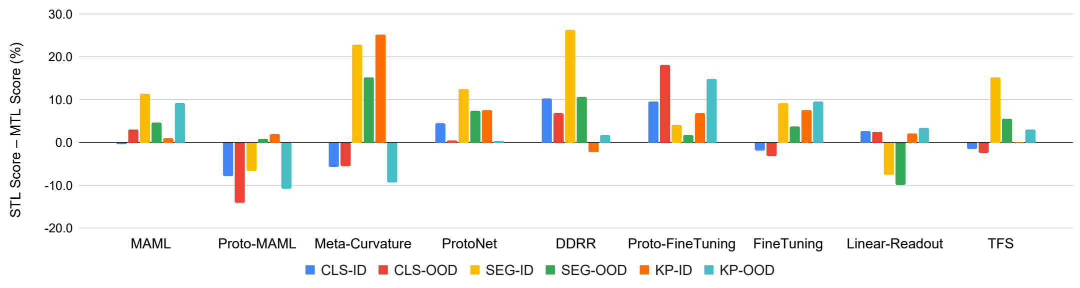


Figure 6.3: Analysis of the differences in scores between single-task (STL) and multi-task (MTL) learning for different methods.

6.4.3 Additional Analysis

How Well Can Multi-Task Meta-Learners Generalize to Completely New Held-Out Tasks? We take the multi-task meta-learners (trained on classification, segmentation, keypoints) and evaluate them on four regression benchmarks inspired by (Gao et al., 2022b): ShapeNet1D, ShapeNet2D, Distractor and Pascal1D. Because the metrics differ across datasets, we analyse the rankings and summarise the results in Table 6.3. We see the basic TFS performs the worst, with MAML, ProtoNets, DDRR being the best. However, in several cases the results were not better than predicting the mean (full results in Appendix D), showing that learning-to-learn of completely new task families is an open challenge.

Table 6.3: Average ranking of the different methods across four out-of-task regression datasets.

MAML	PMAML	MC	PN	DDRR	PFT	FT	LR	TFS
3.3	6.5	4.8	3.5	3.5	3.8	5.8	4.3	8.5

How Much Does External Pre-Training Help? Our focus is on assessing the efficacy of meta-learning rather than representation transfer, but we also aim to support researchers investigating the impact of representation learning on external data prior to

Table 6.4: Analysis of the impact of external data pretraining for selected meta-learners in multi-task learning condition. The results show that ImageNet pretraining does not necessarily help improve performance. Cls., Segm., Keyp. represent classification, segmentation and keypoint respectively.

Method	Pretrain	**Cls.**		**Segm.**		**Keyp.**	
		ID	OOD	ID	OOD	ID	OOD
Proto-MAML	✗	58.5	63.7	53.0	43.2	21.6	33.3
ProtoNet	✗	66.0	58.8	63.3	49.7	20.1	33.0
Proto-MAML	✓	63.9	62.7	56.2	45.3	21.8	33.3
ProtoNet	✓	63.5	58.6	62.0	49.0	20.1	33.1

meta-learning (Zhai et al., 2019; Dumoulin et al., 2021; Hu et al., 2022). We therefore specify evaluation conditions where external data outside our defined in-distribution meta-training set is allowed.

We take two high-performing approaches in the multi-task scenario, Proto-MAML and ProtoNet, and we investigate to what extent external pre-training helps. We use the standard ImageNet1k pre-trained ResNet18, prior to conducting our meta-learning pipeline as usual. We use the same hyperparameters as selected earlier for these models to ensure consistent evaluation, and ensure that the differences in performance are not due to a better selection of hyperparameters. The results in Table 6.4 show that pretraining is not necessarily helpful in the considered multi-task setting, in contrast to purely recognition-focused evaluations (Zhai et al., 2019; Dumoulin et al., 2021; Hu et al., 2022), which were unambiguously positive about representation transfer from external data.

Analysis of Runtimes We analyze the times that the different meta-learning approaches spend on meta-training, meta-validation and meta-testing in the multitask learning case of our benchmark. The results in Table 6.5 show that all experiments are relatively lightweight, despite the ambitious goal of our benchmark to learn a meta-learner that can generalize across various task families. Most notably we observe that ProtoNets are the fastest approach, alongside being the best-performing one. Note that fine-tuning and training from scratch are expensive during the test time as they use backpropagation with a larger number of steps.

Discussion and Future Work In future Meta Omnium can be used in a variety of ways beyond benchmarking multi-task meta-learning per-se. These include: studying the multi-task optimization (Yu et al., 2020) in meta-learning, studying HPO for meta-learning, developing validation strategies in meta-learning (using ID vs OD val. sets (Li et al., 2022)), and studying the benefit of task-specific decoders and external data.

Table 6.5: Analysis of times needed by different algorithms in the multitask setting (using one NVIDIA 1080 Ti GPU and 4 CPUs).

Method	Train Time	Val Time	Test Time	Total Time
MAML	1.8h	1.9h	0.9h	5.0h
Proto-MAML	1.9h	1.9h	0.9h	5.1h
Meta-Curvature	3.4h	2.6h	1.3h	7.6h
ProtoNet	0.8h	0.4h	0.2h	1.8h
DDRR	1.4h	0.6h	0.3h	2.7h
Proto-FineTuning	1.7h	4.5h	2.3h	8.9h
FineTuning	1.5h	8.1h	4.9h	14.9h
Linear-Readout	1.2h	5.1h	2.8h	9.6h
TFS	0.0h	0.8h	6.2h	7.0h

Considering our earlier contributions in this thesis, EvoGrad can be used to make MAML, one of the approaches that we evaluated on Meta Omnium, first-order and hence more efficient. However, it is not needed since direct first-order approximation exists for the case of few-shot learning where we try to meta-learn the neural network weight initialization. PASHA could be considered for optimizing the hyperparameters of the different meta-learning approaches, but as we have noted earlier, it is not applicable out of the box in our multi-objective setup. As a result, non-trivial extensions would be needed. Uncertainty calibration would be valuable also in the case of few-shot learning classification, so one could try to design an approach that extends the ideas from Meta-Calibration. For example, it could be possible to also add label smoothing as a meta-parameter that is trained across tasks, with the outer loop objective regularized with DECE.

6.5 Further Developments

Few-shot learning continues to be a practically valuable problem setting and remains an active area of research, especially in cross-domain scenarios. A recent approach that obtains state-of-the-art results performs neural fine-tuning search (Eustratiadis et al., 2024), showing how techniques from neural architecture search can be valuable also in few-shot learning settings. Other recent approaches include Task-aware adaptive network (Guo et al., 2023), which estimates a parameter configuration for each task and

DiffKendall (Zheng et al., 2023) that improves performance of metric-based methods by using differentiable Kendall's rank correlation as the similarity metric. New approaches continue to be developed also for diverse practical applications, including 3D point cloud object detection (Tang et al., 2023), video classification (Xia et al., 2023), industrial anomaly detection (Fang et al., 2023), image generation (Li et al., 2023) or novel view synthesis (Seo et al., 2023). Our Meta Omnium has been released only relatively recently and so is yet to be utilized by other researchers. Nevertheless, it has been included in a survey on few-shot learning by Tsoumplekas et al. (2024), and highlighted for enabling evaluation of generalization across various learning problems. We believe Meta Omnium has potential and will receive attention thanks to its usefulness for studying few-shot learning in more ambitious setups as well as its accessibility.

Few-shot learning is also increasingly often studied as part of foundational models. It has been shown that language models such as GPT-3 (Brown et al., 2020), or vision-language models such as CLIP (Radford et al., 2021) and Flamingo (Alayrac et al., 2022) offer strong few-shot learning abilities. As a result, there has been growing interest in using these models for few-shot learning. The area has also become known as in-context learning because the few examples serve as context for the responses given by the large models (Dong et al., 2023).

6.6 Conclusions

We have introduced Meta Omnium, the first multi-task few-shot meta-learning benchmark for computer vision. The benchmark is challenging in multiple highly topical ways such as requiring learning on heterogeneous task distributions, evaluating generalization to out-of-distribution datasets, and uniquely challenging meta-learners to learn-to-learn and transfer knowledge across tasks with heterogeneous output spaces. Meta Omnium is nevertheless lightweight enough to be of broad interest and use for driving future research, including research in hyperparameter optimization for meta-learning.

***

We have showcased two novel applications of meta-learning. As a general tool, meta-learning can be used in many more applications, and we hope our work will inspire further research in how meta-learning can be used to solve various other challenges. We discuss several options as part of the conclusion. We also include a wider discussion of the remaining limitations and options for future research more broadly.

Chapter 7

Conclusion

7.1 Summary

Meta-learning enables both living organisms and machines to learn how to improve their learning process. In the context of deep learning it has led to various practical benefits, offering a good solution to multiple challenges, including learning new concepts from a few examples, finding strong neural network architectures and improved robustness to domain shift. Despite these successes, meta-learning is known to be computationally expensive, making it difficult to apply it to many problems where it could be helpful for solving them.

As part of this thesis we have developed two new algorithms for meta-learning and also applied meta-learning to solve two challenges. Our new meta-learning algorithms significantly improve the efficiency of meta-learning, making it possible to use meta-learning for larger models and problem settings. We have also applied meta-learning as a novel solution to two challenging problems: calibration of neural networks and general-purpose few-shot learning.

7.1.1 Efficient Algorithms for Meta-Learning

The first efficient meta-learning algorithm that we have presented is named EvoGrad. The approach focuses on resolving a key bottleneck present in gradient-based bilevel optimization of many meta-parameters. More specifically, the bottleneck is that expensive higher-order gradients arise from backpropagating through inner loop that involves updating parameters of the neural network model using backpropagation.

As part of EvoGrad we replace the inner loop by an evolutionary update, and we keep the outer loop gradient-based so that we can meta-learn precise values of the meta-parameters. The evolutionary update in the inner loop is only a simulation of the training step to improve efficiency. To ensure strong performance of the trained neural network, the actually-used update to the parameters of the neural network is done separately after the inner loop and uses backpropagation. Overall this way of meta-learning leads to large savings in time and memory. At the same time, it maintains or even improves over the performance of the models that are trained with standard meta-learning approaches.

The second approach that we have presented in this thesis is called PASHA. PASHA can optimize any type of meta-parameters and is suitable when there is a smaller number of meta-parameters. It targets inefficiencies coming from training sampled configurations for longer than needed for finding a strong configuration. PASHA starts with a small amount of resources available for each configuration and then progressively expands them based on the need. More specifically it evaluates the stability of rankings of the sampled configurations. If the rankings are stable, it is a sign we do not need to add more resources and the search for strong meta-parameters can be stopped.

Finding an amount of resources that avoids inefficiencies is challenging without adaptive strategies such as PASHA, and so the amount of resources used is typically far larger than needed. PASHA hence leads to large speedups, for example being three times as fast, while identifying similarly-performing configurations. For large datasets the speedups can be even larger, for example more than ten times as fast. Crucially, PASHA is simple to use as it automates the selection of its key parameters that evaluate the stability of rankings.

7.1.2 New Applications of Meta-Learning

We have used meta-learning to develop novel solutions to two challenges, the first of which is calibration of neural networks. Calibration is an important problem within deep learning as it concerns to what extent we can rely on the confidence of the prediction estimated by a neural network. If the neural network is not confident, it is a sign that we should be careful about the given prediction and potentially use a more advanced model or ask a person in the specific situation. However, if the neural network is not well-calibrated, it can be over-confident, potentially posing safety risks.

We have used meta-learning to learn suitable values of non-uniform label smoothing and optimize for well-calibrated models that obtain similar accuracy. By meta-learning non-uniform label smoothing we have been able to obtain excellent calibration that surpasses the calibration obtained using standard methods. Our approach has shown that by casting calibration as a meta-learning problem we can define calibration as a higher-level objective and optimize for it.

We have also applied meta-learning as a solution to a newly defined problem of general-purpose few-shot learning in the area of computer vision. As part of this problem setting we test the ability of few-shot learners to adapt to tasks from multiple task families, including recognition, segmentation, keypoint estimation and regression. Earlier computer vision benchmarks only evaluated for ability to generalize across domains, e.g. from general images to images of birds, but we push the boundaries and challenge the different few-shot learners to a significantly larger degree. In our case the generalization is across different task families, rather than only within classification tasks or only segmentation tasks.

In order to test for such general-purpose few-shot learning abilities, we have developed a new benchmark and also extended the existing few-shot learners to support multi-task learning. Using our benchmark we have been able to identify which few-shot learning approaches are the most powerful in terms of general-purpose few-shot learning, as well as study a variety of other questions related to this setting. Most notably, our experiments have confirmed the benefits of using meta-learning for multi-task few-shot learning, as opposed to simply using transfer learning methods. Finally, our benchmark has been designed to be lightweight so that it can attract wide attention and be easy to use for researchers with various levels of resources.

7.2 Discussion

Our work has introduced two algorithms for more efficient meta-learning and also showcased two applications where meta-learning can be impactful. Nevertheless, efficiency of meta-learning remains an open challenge, and there are also more applications where meta-learning could make an impact.

Meta-learning takes various forms and in our work we have focused on two of them. EvoGrad has focused on the case where we meta-learn suitable meta-parameters alongside training the main model. This form of meta-learning is known to suffer from short-horizon bias or greediness due to using only partial one-step or few-step training unrolls in the inner loop, which means the meta-parameters are unlikely to be optimal. In practice it means that such online meta-learning can be harder to apply out of the box. Substantial work is required to find a combination of meta-parameters and meta-learning schedule that works well and leads to improvements over existing manually developed baselines. In some cases casting a high-level problem (such as calibration of neural networks) as a meta-learning problem will simplify the challenge and save significant time, but in other cases it can be easier to find a good solution manually without redefining the problem. As a result it could be said that current online meta-learning algorithms are not sufficiently robust and more work is needed to make them easier to apply successfully.

Our second method, PASHA, has focused on a part of meta-learning commonly known as hyperparameter optimization. In this case the approach does not suffer from short-horizon bias and could be seen as more robust, but can only optimize a smaller number of meta-parameters because the combinations of parameters are sampled either randomly or using Bayesian Optimization. If there are millions of meta-parameters, gradient-based methods need to be used, but they introduce the challenges that we have mentioned earlier.

Ultimately we would like to be able to backpropagate through the whole learning process to update the meta-parameters, then perform full training with the newly updated meta-parameters, repeating until the meta-parameters converge. This brings various challenges, including extreme compute costs and also gradient instability due to backpropagating through too long loops. There are signs that full-length training in the inner loop can be useful. For example, learned optimizer VeLO (Metz et al., 2022) has used these in combination with evolution strategies to avoid the gradient instability. The learned VeLO optimizer offers compelling performance and does not need hyperparameter tuning, but training it has required about four thousand TPU-months of compute (Metz et al., 2022).

Online meta-learning that has short-horizon bias (also known as myopia) represents a trade-off between using manageable amount of resources, ease of use and the performance of the obtained solution. New approaches that try to alleviate the described limitations are actively developed. For example, bootstrapped meta-learning (Flennerhag et al., 2022) has been able to alleviate myopia by bootstrapping a target from the meta-learner and then optimizing the meta-learner to be close to the target. Libraries to make meta-learning more scalable are also being developed, most recently Betty library (Choe et al., 2023b) that enables parallelization of meta-learning. Overall, there remain many open questions about how to make meta-learning easier to use successfully in practice, and generally to use it to power the next revolution in deep learning.

7.3 Open Questions and Future Work

We have outlined various limitations of meta-learning during our discussion. Some of these relate to the fact that current more scalable approaches often suffer from short-horizon bias, which in turn makes meta-learning less robust and challenging to apply in many situations. On the other hand, approaches that do not make these simplifications are only scalable to smaller-scale settings such as few-shot learning or otherwise are too expensive. Overcoming these trade-offs hence remains one of the key questions that remain to be answered as part of future work.

There are also open questions of more applied nature. Our Meta-Calibration has shown how calibration can be framed as a meta-learning problem by introducing a suitable meta-objective and then using it to optimize meta-parameters. With meta-learning we have been able to obtain superior calibration, and it is likely a similar workflow could be successfully applied for other applications. One example is the problem of fairness (Mehrabi et al., 2019) where we try to ensure the neural network does not discriminate against any of the sub-groups. Future work can hence develop meta-learning approaches for more of the challenges related to deep learning.

Our Meta Omnium benchmark also raises a variety of open questions that can be studied as part of future work. The most direct one regards the development of new meta-learning approaches that target the problem of general-purpose few-shot learning. Other potential questions are of more exploratory nature and include, for example, studying the benefit of task-specific decoders, using external data, various ways to do hyperparameter optimization in multi-task few-shot learning or studying properties of multi-task optimization in meta-learning. While Meta Omnium focuses on different task types within computer vision, it would be interesting to extend the benchmark to also include tasks from various data modalities such as NLP (Bragg et al., 2021), speech (Heggan et al., 2022) or channel coding (Li et al., 2021a). It is possible to have one model to solve tasks from various modalities (Kaiser et al., 2017), so few-shot learning across various data modalities should be also possible in principle. Such benchmark would test how general-purpose existing few-shot learners are to a yet larger extent.

In-context learning (ICL), closely related to few-shot learning, is likely to remain popular. We expect to see more research into how to maximise the performance of ICL, as well as how to use it for achieving various goals, including how to improve AI safety. As large models increasingly become multimodal (Liu et al., 2023; Gemini Team Google, 2023), it is likely also ICL research will increasingly often target multimodal settings. In addition, better understanding of ICL is likely to remain one of the directions of research in this area.

Longer-term it has been envisioned that meta-learning could play a key role in AI-generating algorithms as an alternate paradigm towards artificial general intelligence (Clune, 2019). More specifically it could be used for meta-learning strong architectures as well as the learning algorithms themselves. Meta-learning could presumably also be useful for continuously improving the abilities of the AI model.

Appendix A

Appendix: EvoGrad

A.1 Code Illustration

A.1.1 EvoGrad Code

EvoGrad update is simple to implement if we use *higher* library (Grefenstette et al., 2019) for the perturbed parameters of different model copies. We show only the part that is relevant to the meta-update.

```
model_parameter = [i.detach() for i in get_func_params(model)]
theta_list = [[j + sigma * torch.sign(torch.randn_like(j))
              for j in model_parameter] for i in range(n_model_candidates)]
pred_list = [model_patched(feature_transformer(inputs), params=theta)
             for theta in theta_list]
loss_list = [criterion(pred, targets) for pred in pred_list]
weights = torch.softmax(-torch.stack(loss_list) / temperature, 0)
theta_updated = [sum(map(mul, theta, weights))
                 for theta in zip(*theta_list)]
preds_meta = model_patched(inputs_meta, params=theta_updated)
loss_meta = criterion(preds_meta, targets_meta)

meta_opt.zero_grad()
loss_meta.backward()
meta_opt.step()
```

Listing A.1: EvoGrad code example.

A.1.2 $T_1 - T_2$ Code – for Comparison

For comparison with EvoGrad, we also show how online $T_1 - T_2$-style meta-learning is often implemented using so-called *fast weights*. This approach has been, for example, used in (Chen et al., 2019; Tseng et al., 2020). The meta-update itself is concise, but it requires us to implement layers that support fast weights, which is a significantly longer part.

```
preds = model(feature_transformer(inputs))
loss = criterion(preds, targets)
optimizer.zero_grad()
grads = torch.autograd.grad(loss, model.parameters(), create_graph=True)
for k, weight in enumerate(model.parameters()):
    weight.fast = weight - meta_lr * grads[k]

preds_meta = model(inputs_meta)
loss_meta = criterion(preds_meta, targets_meta)

meta_opt.zero_grad()
loss_meta.backward()
meta_opt.step()
```

Listing A.2: $T_1 - T_2$ code example.

We also show the definition of a linear layer that supports fast weights:

```
class Linear_fw(nn.Linear):
    def __init__(self, in_features, out_features, bias=True):
        super(Linear_fw, self).__init__(in_features, out_features,
                                        bias=bias)
        self.weight.fast = None
        self.bias.fast = None

    def forward(self, x):
        if self.weight.fast is not None and self.bias.fast is not None:
            out = F.linear(x, self.weight.fast, self.bias.fast)
        else:
            out = super(Linear_fw, self).forward(x)
        return out
```

Listing A.3: Code example for a linear layer that supports fast weights.

Normally we would use simple `nn.Linear(in_features, out_features)`.

A.2 How to Select EvoGrad Hyperparameters

EvoGrad as an algorithm has a few hyperparameters common to most evolutionary approaches: perturbation value σ, temperature τ and the number of model copies K. In practice we use only 2 models as it is enough and improves the efficiency. The other hyperparameter values can be selected relatively easily by looking at the training loss of the unperturbed model and the training loss of the perturbed models. The losses should be similar to each other, but not the same – we want to make sure the perturbed weights can still be successfully used. We have found that in practice value $\sigma = 0.001$ is reasonable. Once we have selected the value of σ, we can select the value of temperature

τ which leads to reasonably different weights for the two (or more) model copies. In practice we have found $\tau = 0.05$ to be a value which leads to suitable weights. For example, 0.48 and 0.52 for two model copies could be considered reasonable, while 0.5001 and 0.4999 would be too similar. Note that in the special case of a 1-dimensional toy problem, suitable EvoGrad hyperparameters are different than what is useful for practical problems.

A.3 Additional Details

We include an algorithmic description of the details as well as additional description of the experimental settings for all five problems that we discuss in the chapter.

A.3.1 Illustration Using a 1-Dimensional Problem

We provide more detailed descriptions of how we perform both analyses. In the first analysis, we calculate the EvoGrad hypergradient estimate for 100 values of ω between 0 and 2, starting with 0.1 and ending with 2.0. In each case we perform 100 repetitions to obtain an estimate of the mean and standard deviation of the hypergradient, considering the stochastic nature of EvoGrad. Given a value of ω, the process of EvoGrad estimate can be summarized using Algorithm 6. As a reminder, we use training loss function $f_T(x, \omega) = (x-1)^2 + \omega\|x\|_2^2$ that includes a meta-parameter ω and validation loss function $f_V(x) = (x-0.5)^2$ that does not include the meta-parameter. The value of temperature is 0.5 and the number of model candidates varies between 2, 10 and 100.

Algorithm 6 EvoGrad hypergradient estimate for the 1D problem

```
1: **Input:** $\omega$: target hyperparameter; $K$: number of model candidates; $\tau$: temperature;
   $f_T, f_V$: training and validation loss functions
2: **Output:** $g$: hypergradient estimate
3: Sample $x \sim \mathcal{N}(0,1)$
4: Sample $K$ noise parameters $\epsilon_k \sim \mathcal{N}(0,1)$ and use them to create $x_k = x + \epsilon_k$
5: Calculate losses $\ell_k = f_T(x_k, \omega)$ for $k$ between 1 and $K$
6: Calculate weights $w_1, w_2, \ldots, w_K = \text{softmax}([-\ell_1, -\ell_2, \ldots, -\ell_K]/\tau)$
7: Calculate $x^* = w_1 x_1 + w_2 x_2 + \cdots + w_K x_K$
8: Calculate $\ell_V = f_V(x^*)$
9: Calculate hypergradient $g = \frac{\partial \ell_V}{\partial \omega}$ by backpropagating through $x^*$ computation
```

The second analysis evaluates the trajectories that values of x, ω take if we update them with SGD with the hypergradient estimated by EvoGrad compared to the ground-truth. We can summarize the process using Algorithm 7. When using the ground-truth hypergradient, we simply replace lines 6 to 10 by directly updating the value of ω using the closed-form formula for the hypergradient: $g(\omega) = (\omega - 1)/(\omega + 1)^3$. We use 5 steps, learning rate of 0.1 and temperature 0.5.

Algorithm 7 Training with EvoGrad – 1D problem

```
1:  **Input:** $x_0, \omega_0$: initial values of $x, \omega$; $N$: number of steps; $\alpha$: learning rate; $K$: number of model candidates; $\tau$: temperature; $f_T, f_V$: training and validation loss functions
2:  **Output:** Optimized values of $x, \omega$
3:  Initialize $x = x_0$ and $\omega = \omega_0$
4:  **for** $i$ between 1 and $N$ **do**
5:      Update $x \leftarrow x - \alpha \frac{\partial f_T(x,\omega)}{\partial x}$
6:      Sample $K$ noise parameters $\varepsilon_k \sim \mathcal{N}(0,1)$ and use them to create $x_k = x + \varepsilon_k$
7:      Calculate losses $\ell_k = f_T(x_k, \omega)$ for $k$ between 1 and $K$
8:      Calculate weights $w_1, w_2, \dots, w_K = \text{softmax}([-\ell_1, -\ell_2, \dots, -\ell_K]/\tau)$
9:      Calculate $x^* = w_1 x_1 + w_2 x_2 + \cdots + w_K x_K$
10:     Update $\omega \leftarrow \omega - \alpha \frac{\partial f_V(x^*)}{\partial \omega}$
11: **end for**
```

A.3.2 Rotation Transformation

As part of the rotation transformation problem, we try to prepare a model for the classification of rotated images. We use MNIST images (LeCun et al., 1998) and train the base model with unrotated training images, while testing is done with images rotated by 30°. We split the original training set to create a meta-validation set of size 10,000 with images rotated by 30°.

To prepare the model for the target problem, we meta-learn a rotation transformation alongside training the base model – which we apply to the unrotated images. Our base model is LeNet (LeCun et al., 1989) that has two CNN layers followed by three fully-connected layers. We use ReLU non-linearity and max-pooling. The base model is trained with Adam optimizer (Kingma and Ba, 2015) with 0.001 learning rate, while the meta-parameter is optimized with Adam optimizer with learning rate of

0.01. We use a batch size of 128 and cross-entropy loss ℓ. EvoGrad parameters are $\tau = 0.05, \sigma = 0.001, K = 2$. We sample the noise parameters as $\epsilon_k \sim \sigma\text{sign}(\mathcal{N}(0,I))$, and we use this formulation also in the further practical meta-learning problems – it is a better-controlled version of simple $\mathcal{N}(0,\sigma I)$. We train the models for 5 epochs and repeat the experiments 5 times. The algorithm is summarized in Algorithm 8.

Rotations are performed using a model with one learnable parameter ω (angle). The input that the model receives is rotated using matrix:

$$\begin{pmatrix} \cos(\omega) & -\sin(\omega) \\ \sin(\omega) & \cos(\omega) \end{pmatrix}.$$

Algorithm 8 Training with EvoGrad hypergradient – rotation transformation

```
1: **Input:** $\alpha$: learning rate; $\beta$: meta-learning rate; $\sigma$: noise parameter; $K$: number of model candidates; $\tau$: temperature
2: **Output:** $\theta$: trained model; $\omega$: rotation parameter
3: Initialize $\theta \sim p(\theta)$ and $\omega = 0$
4: while training do
5:     Sample minibatch of training $x_t, y_t$ (standard) and validation $x_v, y_v$ (rotated) examples
6:     Update $\theta \leftarrow \theta - \alpha\nabla_\theta \ell(f_\theta(f_\omega(x_t)), y_t)$
7:     Sample $K$ noise parameters $\epsilon_k \sim \sigma\text{sign}(\mathcal{N}(0,I))$ and use them to create $\theta_k = \theta + \epsilon_k$
8:     Calculate losses $\ell_k = \ell(f_{\theta_k}(f_\omega(x_t)), y_t)$ for $k$ between 1 and $K$
9:     Calculate weights $w_1, w_2, \dots, w_K = \text{softmax}([-\ell_1, -\ell_2, \dots, -\ell_K]/\tau)$
10:    Calculate $\theta^* = w_1\theta_1 + w_2\theta_2 + \dots + w_K\theta_K$
11:    Update $\omega \leftarrow \omega - \beta\nabla_\omega \ell(f_{\theta^*}(x_v), y_v)$
12: end while
```

We compare our meta-learning approach to simple standard training that does not use the rotation transformer. In such case we keep the same settings as before and update the model simply as $\theta \leftarrow \theta - \alpha\nabla_\theta \ell(f_\theta(x_t), y_t)$. The results prove EvoGrad is capable of meta-learning suitable values.

A.3.3 Cross-Domain Few-Shot Classification via Learned Feature-Wise Transformation

We extend the *Learning-to-Learn Feature-Wise Transformation* method from (Tseng et al., 2020) to show the practical impact that EvoGrad can make. The goal of the LFT method is to make metric-based few-shot learners robust to domain shift. A detailed description of the LFT method is provided in (Tseng et al., 2020), and here we describe the main changes that are needed to use EvoGrad for LFT. The key difference is that we do not backpropagate via standard model update that leads to higher memory and time consumption (we measure maximum allocated memory and time per epoch).

We summarize how EvoGrad is applied to LFT in Algorithm 9. A metric based model (we choose RelationNet (Sung et al., 2018)) includes feature encoder E_{θ_e} and metric function M_{θ_m}. Feature transformation layers parameterized by $\theta_f = \{\theta_\gamma, \theta_\beta\}$ are integrated into the feature encoder to form E_{θ_e,θ_f}. Similarly as (Tseng et al., 2020), we sample pseudo-seen $\mathcal{T}^{\text{ps}}$ and pseudo-unseen $\mathcal{T}^{\text{pu}}$ domains from the seen domains $\left\{\mathcal{T}_1^{\text{seen}}, \mathcal{T}_2^{\text{seen}}, \cdots, \mathcal{T}_n^{\text{seen}}\right\}$. In each step, we sample pseudo-seen and pseudo-unseen few-shot learning tasks that both include support and query examples. The pseudo-seen task is described as $T^{\text{ps}} = \left\{\left(\mathcal{X}_s^{\text{ps}}, \mathcal{Y}_s^{\text{ps}}\right), \left(\mathcal{X}_q^{\text{ps}}, \mathcal{Y}_q^{\text{ps}}\right)\right\} \in \mathcal{T}^{\text{ps}}$ and the pseudo-unseen task is $T^{\text{pu}} = \left\{\left(\mathcal{X}_s^{\text{pu}}, \mathcal{Y}_s^{\text{pu}}\right), \left(\mathcal{X}_q^{\text{pu}}, \mathcal{Y}_q^{\text{pu}}\right)\right\} \in \mathcal{T}^{\text{pu}}$, for task examples $\mathcal{X}$ with labels $\mathcal{Y}$.

We have used the exact same set-up as (Tseng et al., 2020) with their official implementation (for RelationNet), so we only describe the additional settings that are unique to us. In particular, EvoGrad-specific parameters are $\tau = 0.05, K = 2, \sigma = 0.001$ (we have used σ equal to the learning rate). We have used ResNet-10 (He et al., 2016) backbone for direct comparison with (Tseng et al., 2020). The datasets that we use are processed in the same way as done by (Tseng et al., 2020), and they are MiniImagenet (Ravi and Larochelle, 2017), CUB (Welinder et al., 2010), Cars (Krause et al., 2013), Places (Zhou et al., 2018) and Plantae (Horn et al., 2018).

In order to use ResNet-34, we have trained a new ResNet-34 baseline model on MiniImagenet (Ravi and Larochelle, 2017) per (Tseng et al., 2020) instructions. We use the same hyperparameters as were used for ResNet-10, which also means that when using fixed feature transformation layers, we use $\theta_\gamma = 0.3, \theta_\beta = 0.5$. Note that ResNet-34 ran out of memory for the original second-order LFT approach on 5-way 5-shot task with 16 query examples when using standard GPU with 12 GB GPU memory. If we wanted to use this model also for the second-order approach, we would need to decrease the number of examples in the task appropriately. However, with EvoGrad we do not need to make this compromise and overall it means that EvoGrad scales also to problems where the original second-order approach does not scale because of GPU memory limitations.

Algorithm 9 Learning-to-learn feature-wise transformation – with EvoGrad

```
1: Input: $\{\mathcal{T}_1^{\text{seen}}, \mathcal{T}_2^{\text{seen}}, \cdots, \mathcal{T}_n^{\text{seen}}\}$: seen domains; $\alpha$: learning rate; $\sigma$: noise parameter; $K$: number of model candidates; $\tau$: temperature
2: Output: $\theta_e$: feature extractor; $\theta_m$: metric learner; $\theta_f$: feature transformation layers
3: Initialize $\theta_e, \theta_m, \theta_f \sim p(\theta_e), p(\theta_m), p(\theta_f)$
4: while training do
5:     Randomly sample non-overlapping pseudo-seen $\mathcal{T}^{\text{ps}}$ and pseudo-unseen $\mathcal{T}^{\text{pu}}$ domains from the seen domains
6:     Sample a pseudo-seen task $T^{\text{ps}} \in \mathcal{T}^{\text{ps}}$ and a pseudo-unseen task $T^{\text{pu}} \in \mathcal{T}^{\text{pu}}$
7:     // Standard update of the metric-based model with pseudo-seen task:
8:     Update $\theta_e, \theta_m \leftarrow \theta_e, \theta_m - \alpha \nabla_{(\theta_e, \theta_m)} \ell\left(M_{\theta_m}\left(\mathcal{Y}_s^{\text{ps}}, E_{\theta_e, \theta_f}\left(\mathcal{X}_s^{\text{ps}}\right), E_{\theta_e, \theta_f}\left(\mathcal{X}_q^{\text{ps}}\right)\right), \mathcal{Y}_q^{\text{ps}}\right)$
9:     // EvoGrad computations:
10:    Sample $K$ noise parameters $\left\{\epsilon_e^{(k)}, \epsilon_m^{(k)}\right\}_{k=1}^{K} \sim \sigma \text{sign}(\mathcal{N}(0, I))$
11:    Create $\theta_e^{(k)} = \theta_e + \epsilon_e^{(k)}$ and $\theta_m^{(k)} = \theta_m + \epsilon_m^{(k)}$ for $k$ between 1 and $K$
12:    Calculate losses $\ell_k = \ell\left(M_{\theta_m^{(k)}}\left(\mathcal{Y}_s^{\text{ps}}, E_{\theta_e^{(k)}, \theta_f}\left(\mathcal{X}_s^{\text{ps}}\right), E_{\theta_e^{(k)}, \theta_f}\left(\mathcal{X}_q^{\text{ps}}\right)\right), \mathcal{Y}_q^{\text{ps}}\right)$
13:    Calculate weights $w_1, w_2, \ldots, w_K = \text{softmax}([-\ell_1, -\ell_2, \ldots, -\ell_K]/\tau)$
14:    Calculate $\theta_e^* = w_1 \theta_e^{(1)} + w_2 \theta_e^{(2)} + \cdots + w_K \theta_e^{(K)}$
15:    Calculate $\theta_m^* = w_1 \theta_m^{(1)} + w_2 \theta_m^{(2)} + \cdots + w_K \theta_m^{(K)}$
16:    // Update feature-wise transformation layers with pseudo-unseen task:
17:    Update $\theta_f \leftarrow \theta_f - \alpha \nabla_{\theta_f} \ell\left(M_{\theta_m^*}\left(\mathcal{Y}_s^{\text{pu}}, E_{\theta_e^*}\left(\mathcal{X}_s^{\text{pu}}\right), E_{\theta_e^*}\left(\mathcal{X}_q^{\text{pu}}\right)\right), \mathcal{Y}_q^{\text{pu}}\right)$
18: end while
```

A.3.4 Label Noise with Meta-Weight-Net

We use the experimental set-up from (Shu et al., 2019) for the label noise experiments, together with their official implementation. The label noise experiments use ResNet-32 model and 60 epochs, each of which has 500 iterations. CIFAR-10 and CIFAR-100 (Krizhevsky, 2009) datasets are used. Meta-Weight-Net is represented by a neural network with two linear layers with hidden size of 300 units, ReLU nonlinearity in between and sigmoid output unit. Meta-Weight-Net weights instance-wise losses for each example in the minibatch, which are then combined together by taking their sum.

EvoGrad specific parameters are $\tau = 0.05, K = 2, \sigma = 0.001$. The level of label noise depends on the specific scenario considered – 40%, 20% or 0%.

We provide an overview of the EvoGrad approach applied to the label noise with Meta-Weight-Net problem in Algorithm 10. Even though we do the standard update using noisy examples after the meta-update, the order could be swapped and we simply follow the order chosen by (Shu et al., 2019). Detailed explanations are provided in (Shu et al., 2019), we only explain how we modify the method to use EvoGrad. Note that we do not rerun the baseline experiments and we directly take the reported values from the Meta-Weight-Net paper (Shu et al., 2019). However, we do our own rerun of standard second-order Meta-Weight-Net to get memory and runtime statistics.

Algorithm 10 Meta-Weight-Net for label noise – with EvoGrad

```
1:  Input: $\alpha$: learning rate; $\sigma$: noise parameter; $K$: number of model candidates; $\tau$: temperature
2:  Output: $\theta$: trained model; $\omega$: Meta-Weight-Net parameters
3:  Initialize $\theta, \omega \sim p(\theta), p(\omega)$
4:  while training do
5:      Sample minibatch of training $x_t, y_t$ (noisy) and validation $x_v, y_v$ (clean) examples
6:      // EvoGrad update:
7:      Sample $K$ noise parameters $\epsilon_k \sim \sigma \text{sign}(\mathcal{N}(0, I))$ and use them to create $\theta_k = \theta + \epsilon_k$
8:      Calculate losses $\ell_k = f_\omega\left(\ell(f_{\theta_k}(x_t), y_t)\right)$ for $k$ between 1 and $K$
9:      Calculate weights $w_1, w_2, \ldots, w_K = \text{softmax}([-\ell_1, -\ell_2, \ldots, -\ell_K]/\tau)$
10:     Calculate $\theta^* = w_1\theta_1 + w_2\theta_2 + \cdots + w_K\theta_K$
11:     Update $\omega \leftarrow \omega - \alpha\nabla_\omega \ell(f_{\theta^*}(x_v), y_v)$
12:     // Standard update using noisy examples and MWN:
13:     Update $\theta \leftarrow \theta - \alpha\nabla_\theta f_\omega\left(\ell(f_\theta(x_t), y_t)\right)$
14: end while
```

In addition, we provide further details about Meta-Weight-Net scalability analyses. We have chosen MWN to conduct these analyses because it represents a real problem where meta-learning is helpful, yet the memory consumption and time requirements are small enough to allow us to easily evaluate scaling up of the numbers of parameters. All Meta-Weight-Net scalability experiments are repeated 5 times, but we do not run them fully – we only do 10 epochs to get estimates of the time per epoch.

We have provided the main results that evaluate the impact of using a model with significantly more parameters in the main chapter. Here we provide additional figures. Figure A.1 shows the impact of variable number of meta-parameters (number of hidden units in MWN). We can see the number of meta-parameters does not significantly impact the memory usage or runtime. This is likely because we use reverse-mode backpropagation that becomes more expensive with more model parameters and not hyperparameters (Micaelli and Storkey, 2019). Further, the number of meta-parameters still remains small compared to the size of the model. Figure A.2 shows the number of model copies does not lead to increased memory consumption, perhaps because we only keep the model weights in memory and not also many intermediate variables like activations that are needed for backpropagation – backpropagation is significantly more expensive in terms of memory than forward propagation (Rajeswaran et al., 2019). The runtime increases slightly with additional model copies, which comes from the need to calculate additional forward propagations.

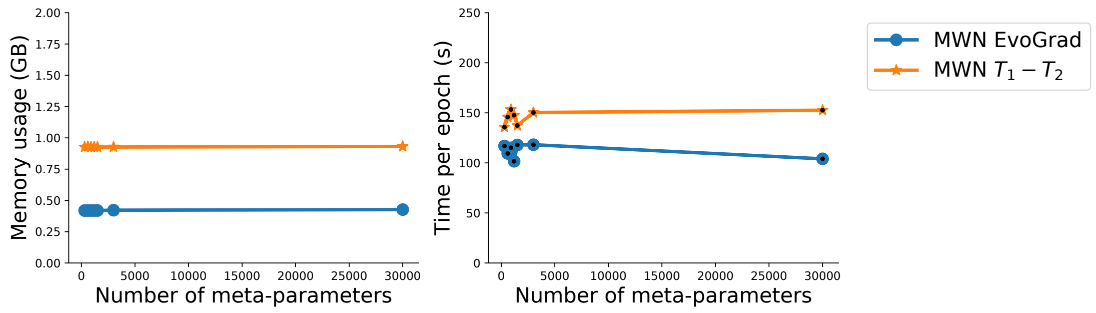


Figure A.1: Memory and time scaling of MWN EvoGrad vs original second-order Meta-Weight-Net – when changing the number of learnable hyperparameters (meta-parameters). The number of meta-parameters does not noticeably influence the memory usage and time per epoch in this case.

A.3.5 Low-Resource Cross-Lingual Learning with MetaXL

MetaXL (Xia et al., 2021) is an approach that meta-learns meta representation transformation to improve transfer in low-resource cross-lingual learning. We show how EvoGrad is applied to MetaXL in Algorithm 11

In order to do experiments, we have taken the official code provided by (Xia et al., 2021) and tried to replicate their experiments as closely as possible. We used the named entity recognition (NER) task with English source language. The only change we made is a smaller batch size: 12 instead of 16 to fit into the memory of the largest GPUs that we have currently available. All details are described in (Xia et al., 2021). For EvoGrad

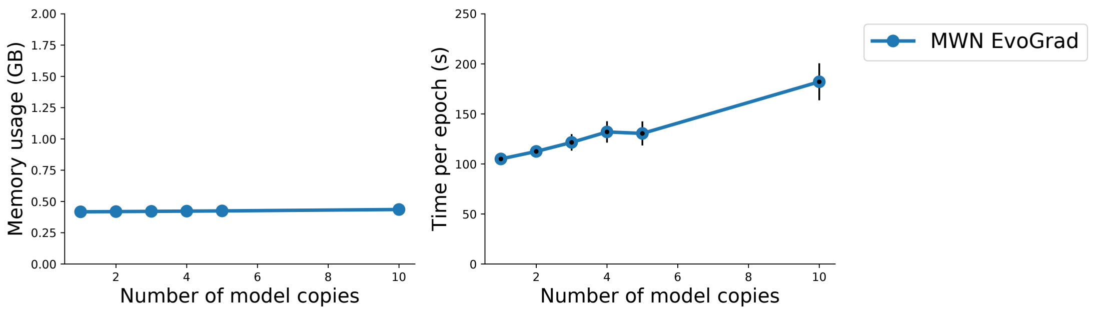


Figure A.2: Memory and time scaling of MWN EvoGrad – when using different numbers of model copies. A larger number of model copies does not increase the memory usage in this case, but it leads to a larger time per epoch.

we have selected the same hyperparameters as for the other tasks (two model candidates, $\sigma = 0.001$ and $\tau = 0.05$). In order to make the implementation of EvoGrad on MetaXL simple, we have only applied noise perturbation on the top layer of the model. It is likely that in practice it is enough to only apply the noise to the top layer, which can make using EvoGrad very simple in most cases.

A.3.6 Datasets Availability

All datasets that we use are freely available and their details are described in (Tseng et al., 2020), (Shu et al., 2019) and (Xia et al., 2021) – including how to download them.

A.3.7 Computational Resources

Illustration using a 1-dimensional problem and rotation transformation can be easily run on a laptop GPU. For cross-domain few-shot learning with LFT and label noise with MWN, we have used an internal cluster with NVIDIA GPUs - Titan X or P100 (all with 12GB GPU memory). For MetaXL we have used NVIDIA 3090 Ti GPUs with 24GB memory. When reporting the time and memory statistics we made sure to use the same model of GPU so that the comparisons are accurate. The experiments were allocated 14 GB RAM memory and 6 CPUs to allow for faster data loading (fewer resources would also be suitable).

Algorithm 11 MetaXL for cross-lingual learning – with EvoGrad

```
1: Input: $\alpha, \beta$: learning rates; $\sigma$: noise parameter; $K$: number of model candidates; $\tau$: temperature; $D_t, D_s$: input data from the target and source language
2: Output: $\theta$: trained model; $\omega$: representation transformation network
3: Initialize base model parameters $\theta$ with pretrained XLM-R weights, initialize parameters of the representation transformation network $\omega$ randomly
4: while training do
5:     Sample a source batch $(x_s, y_s)$ from $D_s$ and a target batch $(x_t, y_t)$ from $D_t$
6:     // EvoGrad update:
7:     Sample $K$ noise parameters $\epsilon_k \sim \sigma\text{sign}(\mathcal{N}(0, I))$ and use them to create $\theta_k = \theta + \epsilon_k$
8:     Calculate losses $\ell_k = \ell(f_{\omega \circ \theta_k}(x_s), y_s)$ for $k$ between 1 and $K$
9:     Calculate weights $w_1, w_2, \dots, w_K = \text{softmax}([-\ell_1, -\ell_2, \dots, -\ell_K]/\tau)$
10:    Calculate $\theta^* = w_1\theta_1 + w_2\theta_2 + \cdots + w_K\theta_K$
11:    Update $\omega \leftarrow \omega - \beta\nabla_\omega \ell(f_{\theta^*}(x_t), y_t)$
12:    // Standard update using representation transformation network:
13:    Update $\theta \leftarrow \theta - \alpha\nabla_\theta \ell(f_{\omega \circ \theta}(x_s), y_s)$
14: end while
```

A.4 Evaluation of Hypernetworks

We have evaluated hypernetworks (Lorraine and Duvenaud, 2018) for cross-domain few-shot classification via learned feature-wise transformation, to find if the approach can be useful for recent meta-learning applications. To make the approach computationally viable, we have used hypernetworks with a bottleneck. For H hyperparameters, P model parameters and bottleneck size of B, our hypernetwork ϕ consists of two layers, one with a weight matrix of $H \times B$, followed by $B \times P$ weight matrix, with sigmoid non-linearity in between. Note that B needs to be relatively small and directly using one layer with a weight matrix of $H \times P$ would require far more memory than normally available – for the considered problem. Following (Lorraine and Duvenaud, 2018), we have used bottleneck size $B = 10$. We have used the exact same experimental set-up as in our other experiments.

Our results in Table A.1 show hypernetworks fail to discover a good solution within the standard number of iterations used throughout, and their performance is poor. The results highlight that generating model parameters based on the hyperparameters may not be sufficient in more challenging and more realistic meta-learning problems. It also explains why hypernetworks are not commonly used in meta-learning applications.

Table A.1: RelationNet test accuracies (%) and 95% confidence intervals across test tasks on various unseen datasets. 5-way 1-shot learning at the top and 5-way 5-shot learning at the bottom. Hypernetworks lead to significantly worse accuracies than $T_1 - T_2$ and EvoGrad, showing they fail to generate well-performing model parameters.

Scenario	CUB	Cars	Places	Plantae
LFT with hypernetworks	38.94 ± 0.57	30.10 ± 0.48	38.07 ± 0.58	33.83 ± 0.58
LFT with $T_1 - T_2$	46.03 ± 0.60	31.50 ± 0.49	49.29 ± 0.65	36.34 ± 0.59
LFT with EvoGrad	47.39 ± 0.61	32.51 ± 0.56	50.70 ± 0.66	36.00 ± 0.56
LFT with hypernetworks	56.91 ± 0.57	40.64 ± 0.56	56.08 ± 0.58	44.73 ± 0.57
LFT with $T_1 - T_2$	65.94 ± 0.56	43.88 ± 0.56	65.57 ± 0.57	51.43 ± 0.55
LFT with EvoGrad	64.63 ± 0.56	42.64 ± 0.58	66.54 ± 0.57	52.92 ± 0.57

A.5 Comparison to More Meta-Learning Approaches

In this section we provide an extended comparison of hypergradient approximations by various gradient-based meta-learners, similar to the analysis done in (Lorraine et al., 2020). The approximations themselves are provided in Table A.2, while the time and memory requirements are given in Table A.3.

Table A.2: Comparison of hypergradient approximations of different gradient-based meta-learning methods. Number of inner-loop steps is denoted by i. Note that also one-step approximation methods can be used once per i steps. θ^* describes the optimal model parameters given ω, while $\widehat{\theta^*}$ represents their approximation.

Method	Hypergradient approximation
Unrolled diff. (Maclaurin et al., 2015)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times \sum_{j \leq i} \left[\prod_{k<j} I - \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \theta^T} \right\rvert_{\theta_{i-k}} \right] \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T} \right\rvert_{\theta_{i-j}}$
K-step truncated unrolled diff. (Shaban et al., 2019)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times \sum_{K \leq j \leq i} \left[\prod_{k<j} I - \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \theta^T} \right\rvert_{\theta_{i-k}} \right] \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T} \right\rvert_{\theta_{i-j}}$
$T_1 - T_2$ (Luketina et al., 2016)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times [I]^{-1} \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T} \right\rvert_{\widehat{\theta^*}(\omega)}$
Hypernetworks (Lorraine and Duvenaud, 2018)	$\frac{\partial \ell_V}{\partial \omega} + \frac{\partial \ell_V}{\partial \theta} \times \frac{\partial \theta^*_\phi}{\partial \omega}$ where $\theta^*_\phi(\omega) = \arg\min_\phi \ell_T \left(\omega, \theta_\phi(\omega) \right)$
Exact IFT (Lorraine et al., 2020)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times \left[\frac{\partial^2 \ell_T}{\partial \theta \partial \theta^T} \right]^{-1} \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T} \right\rvert_{\theta^*(\omega)}$
Neumann IFT (Lorraine et al., 2020)	$\frac{\partial \ell_V}{\partial \omega} - \frac{\partial \ell_V}{\partial \theta} \times \left(\sum_{j<i} \left[I - \frac{\partial^2 \ell_T}{\partial \theta \partial \theta^T} \right]^j \right) \left. \frac{\partial^2 \ell_T}{\partial \theta \partial \omega^T} \right\rvert_{\widehat{\theta^*}(\omega)}$
EvoGrad (ours)	$\frac{\partial \ell_V}{\partial \omega} + \frac{\partial \ell_V}{\partial \theta} \times \mathcal{E} \frac{\partial w}{\partial \ell} \frac{\partial \ell}{\partial \omega} = \frac{\partial \ell_V}{\partial \omega} + \frac{\partial \ell_V}{\partial \theta} \times \mathcal{E} \left. \frac{\partial \operatorname{softmax}(-\ell)}{\partial \omega} \right\rvert_{\widehat{\theta^*}(\omega)}$

Table A.3: Comparison of asymptotic time and memory requirements of EvoGrad and other gradient-based meta-learners. P is the number of model parameters, H is the number of hyperparameters, I is the number of inner-loop steps, N is the number of model copies in EvoGrad. Note this is a first-principles analysis, so the time requirements are different when using e.g. reverse-mode backpropagation that uses parallelization.

Method	Time requirements	Memory requirements
Unrolled diff. (Maclaurin et al., 2015)	$O(IP^2 + PH)$	$O(PI + H)$
K-step truncated unrolled diff. (Shaban et al., 2019)	$O(KP^2 + PH)$	$O(PK + H)$
$T_1 - T_2$ (Luketina et al., 2016)	$O(PH)$	$O(P + H)$
Linear hypernetworks (Lorraine and Duvenaud, 2018)	$O(PH)$	$O(PH)$
Neumann IFT (Lorraine et al., 2020)	$O(P^2 + PH)$	$O(P + H)$
EvoGrad (ours)	$O(NP + H)$	$O(P + H)$

Appendix B

Appendix: PASHA

B.1 Additional Baselines

We consider additional baselines that evaluate how good two, three and five-epoch baselines are compared to PASHA. From Table B.1 and B.2 we see that while these usually get closer to the performance of ASHA and PASHA than the one-epoch baseline, they are still relatively far compared to PASHA. Moreover, it is crucial to observe that such baselines cannot dynamically allocate resources and decide when to stop, and as a result PASHA can outperform them both in terms of speedup and the quality of the found configuration.

Table B.1: NASBench201 results. PASHA leads to large improvements in runtime, while achieving similar accuracy as ASHA.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
	ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
	PASHA	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
	One-epoch baseline	93.30 ± 0.61	0.3h ± 0.0h	8.5x	1.0 ± 0.0
CIFAR-10	Two epoch baseline	92.75 ± 0.91	0.7h ± 0.0h	4.2x	2.0 ± 0.0
	Three epoch baseline	93.47 ± 0.71	1.0h ± 0.0h	2.8x	3.0 ± 0.0
	Five epoch baseline	93.38 ± 0.90	1.7h ± 0.0h	1.7x	5.0 ± 0.0
	Random baseline	72.88 ± 19.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
	ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
	PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
	One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
CIFAR-100	Two-epoch baseline	68.28 ± 4.25	0.7h ± 0.0h	4.6x	2.0 ± 0.0
	Three-epoch baseline	70.47 ± 1.60	1.0h ± 0.0h	3.1x	3.0 ± 0.0
	Five-epoch baseline	70.95 ± 0.95	1.7h ± 0.0h	1.8x	5.0 ± 0.0
	Random baseline	42.83 ± 18.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
	ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
	PASHA	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
	One-epoch baseline	41.42 ± 4.98	1.0h ± 0.0h	8.8x	1.0 ± 0.0
ImageNet16-120	Two-epoch baseline	42.99 ± 1.89	2.0h ± 0.0h	4.4x	2.0 ± 0.0
	Three-epoch baseline	44.65 ± 0.95	3.0h ± 0.0h	2.9x	3.0 ± 0.0
	Five-epoch baseline	44.75 ± 1.03	5.0h ± 0.1h	1.8x	5.0 ± 0.0
	Random baseline	20.75 ± 9.97	0.0h ± 0.0h	N/A	0.0 ± 0.0

Table B.2: Results of the HPO experiments on WMT and ImageNet tasks from the PD1 benchmark. Mean and std of the best validation accuracy (or its equivalent as given in the PD1 benchmark).

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
	ASHA	62.72 ± 1.41	43.7h ± 37.2h	1.0x	1357.4 ± 80.4
	PASHA	62.04 ± 2.05	2.8h ± 0.6h	15.5x	37.8 ± 21.6
	One-epoch baseline	62.36 ± 1.40	0.6h ± 0.0h	67.3x	1.0 ± 0.0
WMT	Two-epoch baseline	60.16 ± 3.58	1.1h ± 0.0h	39.1x	2.0 ± 0.0
	Three-epoch baseline	61.12 ± 3.47	1.6h ± 0.0h	27.6x	3.0 ± 0.0
	Five-epoch baseline	57.89 ± 5.33	2.5h ± 0.0h	17.3x	5.0 ± 0.0
	Random baseline	33.93 ± 21.96	0.0h ± 0.0h	N/A	0.0 ± 0.0
	ASHA	75.10 ± 2.03	7.3h ± 1.2h	1.0x	251.0 ± 0.0
	PASHA	73.37 ± 2.71	3.8h ± 1.0h	1.9x	45.0 ± 30.1
	One-epoch baseline	63.40 ± 9.91	1.1h ± 0.0h	6.7x	1.0 ± 0.0
ImageNet	Two-epoch baseline	64.61 ± 10.81	1.7h ± 0.0h	4.2x	2.0 ± 0.0
	Three-epoch baseline	68.74 ± 3.79	2.3h ± 0.1h	3.2x	3.0 ± 0.0
	Five-epoch baseline	65.91 ± 3.99	3.7h ± 0.1h	2.0x	5.0 ± 0.0
	Random baseline	36.94 ± 31.05	0.0h ± 0.0h	N/A	0.0 ± 0.0

B.2 Reduction Factor

Table B.3 shows the full set of results for our experiments with different reduction factors. The behaviour is the same across all cases.

Table B.3: NASBench201 results with various reduction factors η.

Dataset	Reduction factor	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	$\eta = 2$	ASHA	93.88 ± 0.27	3.6h ± 1.1h	1.0x	200.0 ± 0.0
		PASHA	93.53 ± 0.76	1.0h ± 0.3h	3.5x	9.1 ± 8.1
	$\eta = 4$	ASHA	93.75 ± 0.28	2.4h ± 0.6h	1.0x	200.0 ± 0.0
		PASHA	93.65 ± 0.65	1.1h ± 0.5h	2.3x	32.3 ± 50.2
CIFAR-100	$\eta = 2$	ASHA	71.67 ± 0.84	3.8h ± 1.0h	1.0x	200.0 ± 0.0
		PASHA	71.65 ± 1.42	0.9h ± 0.1h	4.2x	5.9 ± 2.0
	$\eta = 4$	ASHA	71.43 ± 1.13	2.7h ± 0.9h	1.0x	200.0 ± 0.0
		PASHA	72.09 ± 1.22	1.0h ± 0.4h	2.8x	25.1 ± 49.0
ImageNet16-120	$\eta = 2$	ASHA	46.09 ± 0.68	11.9h ± 4.0h	1.0x	200.0 ± 0.0
		PASHA	45.35 ± 1.52	2.8h ± 0.6h	4.2x	9.3 ± 7.1
	$\eta = 4$	ASHA	45.43 ± 0.98	7.9h ± 3.0h	1.0x	200.0 ± 0.0
		PASHA	45.52 ± 1.30	2.9h ± 1.1h	2.8x	18.4 ± 18.7

B.3 Experiments with Alternative Ranking Functions

B.3.1 Description

PASHA employs a ranking function whose choice is completely arbitrary. In our main set of experiments we used soft ranking that automatically estimates the value of ε by measuring noise in rankings. In this set of experiments we would like to evaluate different criteria to define the ranking of the candidates. We describe the functions considered next.

Direct Ranking As a baseline, we study if we can use the simple ranking of configurations by predictive performance (e.g., sorting from the ones with the highest accuracy to the ones with the lowest). If any of the configurations change their order, we consider the ranking unstable and increase the resources.

Soft Ranking Variations We consider several variations of soft ranking. The first variation is to fix the value of the ε parameter. We have considered values 0.01, 0.02, 0.025, 0.03, 0.05. The second set of variations aim to estimate the value of ε

automatically, using various heuristics. The heuristics we have evaluated include:

- Standard deviation: calculate the standard deviation of the considered performance measure (e.g. accuracy) of the configurations in the previous rung and set a multiple of it as the value of ε – we tried multiples of 1, 2 and 3.
- Mean distance: value of ε is set as the mean distance between the score of the configurations in the previous rung.
- Median distance: similar to the mean distance, but using the median distance.

There are various benefits for estimating the value of ε by measuring noise in rankings, as we have presented earlier:

- There is no need to set the value of ε manually.
- Estimation of ε has an intuitive motivation that makes sense.
- The value of ε can dynamically adapt to the different stages of hyperparameter optimization.
- The approach works well in practice.

Rank Biased Overlap (RBO) (Webber et al., 2010) A score that can be broadly interpreted as a weighted correlation between rankings. We can specify how much we want to prioritize the top of the ranking using parameter p that is between 0.0 and 1.0, with a smaller value giving more priority to the top of the ranking. The best value is 1.0, while it gives value of 0.0 for rankings that are completely the opposite. We compute the RBO value and then compare it to the selected threshold t, increasing the resources if the value is less than the threshold.

Reciprocal Rank Regret (RRR) A key insight is that configurations can be very similar to each other and differences in their rankings will not affect the quality of the found solution significantly. To account for this we look at the objective values of the configurations (e.g. accuracy) and compute the relative regret that we would pay at the current rung if we would have assumed the ranking at the previous rung correct.

We define reciprocal rank regret (RRR) as:

$$\text{RRR} = \sum_{i=0}^{n-1} \frac{(f_i - f_i')}{f_i} w^i,$$

where f represents the ordered scores in the top rung, f' represents the reordered scores from the top rung according to the second rung, n is the number of configurations in the top rung and p is the parameter that says how much attention to give to the top of the ranking. The weights w_i sum to 1 and can be selected in different ways to e.g. give more priority to the top of the ranking. For example, we could use the following weights:

$$w_i = \frac{p^i}{\sum_{i=0}^{n-1} p^i}$$

The metric has an intuitive interpretation: it is the average relative regret with priority on top of the ranking. The best value of RRR is 0.0, while the worst possible value is 1.0.

We also consider a version of RRR which considers the absolute values of the differences in the objectives - Absolute RRR (ARRR).

We have evaluated these additional ranking functions using NASBench201 benchmark.

B.3.2 Results

We report the results in Table B.4, B.5 and B.6. We see there are also several other variations that achieve strong results across a variety of datasets within NASBench201, most notably soft ranking 2σ and variations based on RRR. In these cases we obtain similar performance as ASHA, but at a significantly shorter time. We additionally also give a similar analysis in Table B.7 (analogous to Table 4.4), where we analyse a selection of the most interesting ranking functions for the PD1 benchmark.

Table B.4: NASBench201 – CIFAR-10 results for a variety of ranking functions.

Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
PASHA	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
PASHA direct ranking	93.79 ± 0.26	2.7h ± 0.6h	1.1x	198.4 ± 6.0
PASHA soft ranking $\varepsilon = 0.01$	93.79 ± 0.26	2.6h ± 0.5h	1.1x	194.3 ± 21.2
PASHA soft ranking $\varepsilon = 0.02$	93.78 ± 0.31	2.4h ± 0.5h	1.2x	152.4 ± 58.3
PASHA soft ranking $\varepsilon = 0.025$	93.78 ± 0.31	2.3h ± 0.5h	1.3x	144.5 ± 59.4
PASHA soft ranking $\varepsilon = 0.03$	93.78 ± 0.32	2.2h ± 0.6h	1.3x	128.6 ± 58.3
PASHA soft ranking $\varepsilon = 0.05$	93.79 ± 0.49	1.8h ± 0.7h	1.6x	76.0 ± 66.0
PASHA soft ranking 1σ	93.75 ± 0.32	2.4h ± 0.5h	1.2x	186.4 ± 35.2
PASHA soft ranking 2σ	93.88 ± 0.28	1.9h ± 0.5h	1.5x	132.7 ± 68.7
PASHA soft ranking 3σ	93.56 ± 0.69	0.9h ± 0.3h	3.1x	16.2 ± 19.9
PASHA soft ranking mean distance	93.73 ± 0.52	2.3h ± 0.4h	1.3x	184.1 ± 40.5
PASHA soft ranking median distance	93.82 ± 0.26	2.3h ± 0.5h	1.3x	169.2 ± 51.2
PASHA RBO p=1.0, t=0.5	93.49 ± 0.78	0.7h ± 0.1h	4.2x	4.6 ± 6.0
PASHA RBO p=0.5, t=0.5	93.77 ± 0.35	2.2h ± 0.6h	1.3x	144.0 ± 71.2
PASHA RRR p=1.0, t=0.05	93.49 ± 0.78	0.7h ± 0.0h	4.4x	3.0 ± 0.0
PASHA RRR p=0.5, t=0.05	93.76 ± 0.31	2.1h ± 0.6h	1.4x	140.9 ± 69.7
PASHA ARRR p=1.0, t=0.05	93.71 ± 0.35	2.4h ± 0.4h	1.2x	179.0 ± 42.9
PASHA ARRR p=0.5, t=0.05	93.81 ± 0.30	2.5h ± 0.4h	1.2x	181.0 ± 40.9
One-epoch baseline	93.30 ± 0.61	0.3h ± 0.0h	8.5x	1.0 ± 0.0
Random baseline	72.88 ± 19.20	0.0h ± 0.0h	N/A	0.0 ± 0.0

Table B.5: NASBench201 – CIFAR-100 results for a variety of ranking functions.

Approach	Accuracy (%)	Runtime (s)	Speedup factor	Max resources
ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
PASHA direct ranking	71.69 ± 1.05	2.8h ± 0.7h	1.1x	200.0 ± 0.0
PASHA soft ranking $\epsilon = 0.01$	71.55 ± 1.04	2.5h ± 0.7h	1.3x	198.3 ± 6.5
PASHA soft ranking $\epsilon = 0.02$	70.94 ± 0.85	2.0h ± 0.5h	1.6x	160.5 ± 62.9
PASHA soft ranking $\epsilon = 0.025$	71.41 ± 1.15	1.5h ± 0.7h	2.1x	88.3 ± 74.4
PASHA soft ranking $\epsilon = 0.03$	71.00 ± 1.38	1.0h ± 0.5h	3.2x	39.4 ± 63.4
PASHA soft ranking $\epsilon = 0.05$	70.71 ± 1.66	0.7h ± 0.0h	4.9x	3.0 ± 0.0
PASHA soft ranking 1σ	71.56 ± 1.03	2.5h ± 0.6h	1.3x	184.1 ± 40.5
PASHA soft ranking 2σ	71.14 ± 0.97	1.9h ± 0.7h	1.7x	136.4 ± 75.8
PASHA soft ranking 3σ	71.63 ± 1.60	1.0h ± 0.3h	3.3x	20.2 ± 25.3
PASHA soft ranking mean distance	71.51 ± 0.99	2.4h ± 0.5h	1.4x	189.8 ± 30.3
PASHA soft ranking median distance	71.52 ± 0.98	2.4h ± 0.6h	1.3x	189.5 ± 30.6
PASHA RBO p=1.0, t=0.5	70.69 ± 1.67	0.7h ± 0.1h	4.6x	3.8 ± 2.0
PASHA RBO p=0.5, t=0.5	71.51 ± 0.93	2.4h ± 0.7h	1.3x	180.5 ± 50.6
PASHA RRR p=1.0, t=0.05	70.71 ± 1.66	0.7h ± 0.0h	4.9x	3.0 ± 0.0
PASHA RRR p=0.5, t=0.05	71.42 ± 1.51	1.2h ± 0.5h	2.6x	39.3 ± 51.4
PASHA ARRR p=1.0, t=0.05	70.80 ± 1.70	0.8h ± 0.4h	3.8x	22.9 ± 51.3
PASHA ARRR p=0.5, t=0.05	71.41 ± 1.05	1.8h ± 0.6h	1.7x	110.0 ± 68.7
One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
Random baseline	42.83 ± 18.20	0.0h ± 0.0h	N/A	0.0 ± 0.0

Table B.6: NASBench201 – ImageNet16-120 results for a variety of ranking functions.

Approach	Accuracy (%)	Runtime (s)	Speedup factor	Max resources
ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
PASHA	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
PASHA direct ranking	45.63 ± 0.81	8.3h ± 2.5h	1.1x	200.0 ± 0.0
PASHA soft ranking $\epsilon = 0.01$	45.52 ± 0.89	7.0h ± 1.5h	1.3x	185.7 ± 36.1
PASHA soft ranking $\epsilon = 0.02$	45.79 ± 1.16	4.4h ± 1.4h	2.0x	71.4 ± 50.8
PASHA soft ranking $\epsilon = 0.025$	46.01 ± 1.00	3.2h ± 1.0h	2.8x	28.6 ± 27.7
PASHA soft ranking $\epsilon = 0.03$	45.62 ± 1.48	2.4h ± 0.7h	3.6x	11.0 ± 10.0
PASHA soft ranking $\epsilon = 0.05$	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA soft ranking 1σ	45.63 ± 0.89	6.5h ± 1.3h	1.4x	177.1 ± 44.2
PASHA soft ranking 2σ	45.39 ± 1.22	4.5h ± 1.4h	1.9x	91.2 ± 58.0
PASHA soft ranking 3σ	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA soft ranking mean distance	45.50 ± 1.12	6.2h ± 1.5h	1.4x	157.7 ± 54.7
PASHA soft ranking median distance	45.67 ± 0.95	6.3h ± 1.6h	1.4x	156.3 ± 52.2
PASHA RBO p=1.0, t=0.5	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA RBO p=0.5, t=0.5	45.24 ± 1.13	6.4h ± 1.3h	1.4x	148.3 ± 56.9
PASHA RRR p=1.0, t=0.05	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA RRR p=0.5, t=0.05	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA ARRR p=1.0, t=0.05	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
PASHA ARRR p=0.5, t=0.05	44.90 ± 1.42	1.8h ± 0.0h	5.0x	3.0 ± 0.0
One-epoch baseline	41.42 ± 4.98	1.0h ± 0.0h	8.8x	1.0 ± 0.0
Random baseline	20.75 ± 9.97	0.0h ± 0.0h	N/A	0.0 ± 0.0

Table B.7: Results of the HPO experiments on WMT and ImageNet tasks from the PD1 benchmark, using a selection of the most interesting candidates for ranking functions. Mean and std of the best validation accuracy (or its equivalent as given in the PD1 benchmark).

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
WMT	ASHA	62.72 ± 1.41	43.7h ± 37.2h	1.0x	1357.4 ± 80.4
	PASHA	62.04 ± 2.05	2.8h ± 0.6h	15.5x	37.8 ± 21.6
	PASHA direct ranking	62.16 ± 1.75	39.3h ± 38.3h	1.1x	1024.0 ± 466.6
	PASHA soft ranking $\varepsilon = 2.5\%$	62.09 ± 2.04	1.3h ± 0.4h	33.4x	4.2 ± 2.4
	PASHA soft ranking 2σ	62.52 ± 2.18	1.1h ± 0.1h	38.8x	3.0 ± 0.0
	PASHA RBO p=0.5, t=0.5	61.44 ± 1.23	6.7h ± 7.8h	6.5x	147.6 ± 113.2
	PASHA RRR p=0.5, t=0.05	62.52 ± 2.18	1.1h ± 0.1h	38.8x	3.0 ± 0.0
	One-epoch baseline	62.36 ± 1.40	0.6h ± 0.0h	67.3x	1.0 ± 0.0
	Random baseline	33.93 ± 21.96	0.0h ± 0.0h	N/A	0.0 ± 0.0
ImageNet	ASHA	75.10 ± 2.03	7.3h ± 1.2h	1.0x	251.0 ± 0.0
	PASHA	73.37 ± 2.71	3.8h ± 1.0h	1.9x	45.0 ± 30.1
	PASHA direct ranking	75.10 ± 2.03	6.8h ± 0.7h	1.1x	247.8 ± 3.9
	PASHA soft ranking $\varepsilon = 2.5\%$	74.73 ± 1.99	4.3h ± 2.5h	1.7x	140.4 ± 112.8
	PASHA soft ranking 2σ	75.82 ± 0.82	5.0h ± 1.6h	1.5x	133.0 ± 96.8
	PASHA RBO p=0.5, t=0.5	74.80 ± 2.19	4.4h ± 2.1h	1.6x	117.4 ± 109.4
	PASHA RRR p=0.5, t=0.05	74.98 ± 2.12	1.6h ± 0.0h	4.7x	3.0 ± 0.0
	One-epoch baseline	63.40 ± 9.91	1.1h ± 0.0h	6.7x	1.0 ± 0.0
	Random baseline	36.94 ± 31.05	0.0h ± 0.0h	N/A	0.0 ± 0.0

B.4 Investigation with Variable Maximum Resources

We analyse the impact of variable maximum resources (number of epochs) on how large speedup PASHA provides over ASHA. More specifically, we change the maximum resources available for ASHA and also the upper boundary on maximum resources for PASHA. We utilize NASBench201 benchmark for these experiments and set the number of epochs to 200 (default) or 50 (other details are the same as earlier). The results in Table B.8 confirm that PASHA leads to larger speedups when there are more epochs (and rung levels) available. This analysis also explains the modest speedups on LCBench analysed earlier.

If the model is trained for a small number of epochs, it is worth redesigning the HPO so that there are more rung levels available, enabling PASHA to give larger speedups. This can be achieved by using sub-epoch resource levels – specifying the rung levels and the minimum resources in terms of the number of iterations (neural network weights updates). Based on the results observed across various benchmarks, we would recommend having at least 5 rung levels in ASHA, with more rung levels leading to larger speedups from PASHA over ASHA.

Table B.8: NASBench201 results. PASHA leads to larger speedups if the models are trained with more epochs.

Dataset	Number of epochs	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	200	ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
		PASHA	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
	50	ASHA	93.78 ± 0.39	1.8h ± 0.2h	1.0x	50.0 ± 0.0
		PASHA	93.58 ± 0.75	1.2h ± 0.4h	1.5x	22.0 ± 16.8
CIFAR-100	200	ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
		PASHA	71.84 ± 1.41	0.9h ± 0.4h	3.4x	20.5 ± 48.3
	50	ASHA	72.24 ± 0.87	1.8h ± 0.3h	1.0x	50.0 ± 0.0
		PASHA	71.91 ± 1.32	0.9h ± 0.3h	2.0x	10.5 ± 12.1
ImageNet16-120	200	ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
		PASHA	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
	50	ASHA	45.97 ± 0.99	5.2h ± 0.7h	1.0x	50.0 ± 0.0
		PASHA	45.09 ± 1.52	2.7h ± 1.0h	1.9x	11.3 ± 11.7

B.5 Analysis of Learning Curves

We analyse the NASBench201 learning curves in Figure B.1 and B.2. To make the analysis realistic and easier to grasp, we first sample a random subset of 256 configurations, similarly as we do for our NAS experiments. Figure B.1 shows the learning curves of the top three configurations (selected in terms of their final performance). We see that these learning curves are very close to each other and frequently cross due to noise in the training, allowing us to estimate a meaningful value of ε parameter (configurations that repeatedly swap their order are very likely to be similarly good, so we can select any of them because the goal is to find a strong configuration quickly rather than the very best one). Figure B.2 shows all learning curves from the same random sample of 256 configurations. In this case we can see that the learning curves are relatively well-behaved (especially the ones at the top), and any exceptions are rare.

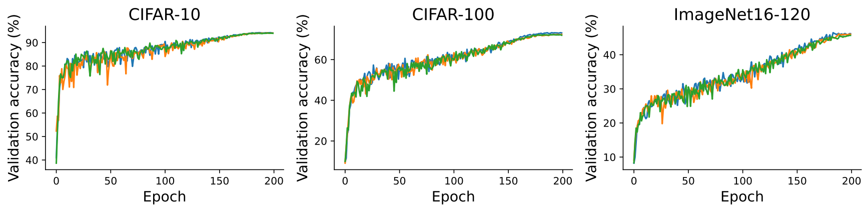


Figure B.1: Analysis of how the learning curves of the top three configurations (in terms of final validation accuracy; from a random sample of 256 configurations) evolve across epochs. We see that such similar configurations frequently change their ranks, enabling us to calculate a meaningful value of ε parameter.

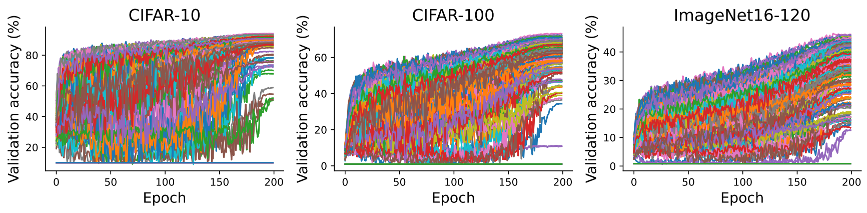


Figure B.2: Analysis of what the learning curves look like for a random sample of 256 configurations. We see that the learning curves are relatively well-behaved (especially the ones at the top), and any exceptions are rare.

B.6 Investigation of How Value ε Evolves

We analyse how the value of ε that is used for calculating soft ranking develops during the HPO process. We show the results in Figure B.3 for the three different datasets available in NASBench201 (taking one seed). The results show the obtained values of ε are relatively small.

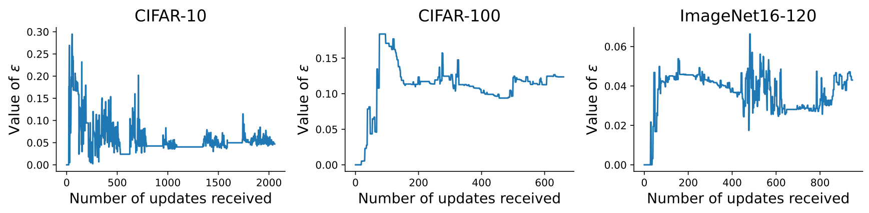


Figure B.3: Analysis of how the value of ε evolves as we receive additional updates about the performances of candidate configurations. Note that most of the updates are obtained in the top rung due to how multi-fidelity methods work.

B.7 Investigation of Percentile Value N

We investigate the impact of using various percentile values N used for estimating the value of ε in Table B.9. The intuition is that we want to take some value on the top end rather than the maximum distance in case there are some outliers. We see that the results are relatively stable, even though larger value of N can lead to further speedups. However, from the point of view of a practitioner we would still take $N = 90$ in case there are any outliers in the specific new use-case.

Table B.9: NASBench201 results. PASHA leads to large improvements in runtime, while achieving similar accuracy as ASHA. Investigation of various percentile values (N) to use for calculating parameter ε.

Dataset	Approach	Accuracy (%)	Runtime	Speedup factor	Max resources
CIFAR-10	ASHA	93.85 ± 0.25	3.0h ± 0.6h	1.0x	200.0 ± 0.0
	PASHA $N = 100\%$	93.70 ± 0.61	1.0h ± 0.4h	3.0x	13.8 ± 19.5
	PASHA $N = 95\%$	93.64 ± 0.59	1.0h ± 0.4h	2.8x	15.4 ± 19.5
	PASHA $N = 90\%$	93.57 ± 0.75	1.3h ± 0.6h	2.3x	36.1 ± 50.0
	PASHA $N = 80\%$	93.86 ± 0.53	1.5h ± 0.6h	1.9x	60.9 ± 60.7
	One-epoch baseline	93.30 ± 0.61	0.3h ± 0.0h	8.5x	1.0 ± 0.0
	Random baseline	72.88 ± 19.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
CIFAR-100	ASHA	71.69 ± 1.05	3.2h ± 0.9h	1.0x	200.0 ± 0.0
	PASHA $N = 100\%$	71.84 ± 1.41	0.8h ± 0.1h	3.9x	6.6 ± 2.9
	PASHA $N = 95\%$	71.84 ± 1.41	0.8h ± 0.1h	3.9x	6.6 ± 2.9
	PASHA $N = 90\%$	71.91 ± 1.32	0.9h ± 0.3h	3.5x	12.6 ± 19.2
	PASHA $N = 80\%$	71.78 ± 1.31	1.2h ± 0.6h	2.6x	56.0 ± 76.2
	One-epoch baseline	65.57 ± 5.53	0.3h ± 0.0h	9.2x	1.0 ± 0.0
	Random baseline	42.83 ± 18.20	0.0h ± 0.0h	N/A	0.0 ± 0.0
ImageNet16-120	ASHA	45.63 ± 0.81	8.8h ± 2.2h	1.0x	200.0 ± 0.0
	PASHA $N = 100\%$	45.09 ± 1.61	2.3h ± 0.4h	3.7x	7.0 ± 2.8
	PASHA $N = 95\%$	45.26 ± 1.58	2.4h ± 0.4h	3.7x	7.4 ± 2.7
	PASHA $N = 90\%$	45.13 ± 1.51	2.9h ± 1.7h	3.1x	21.3 ± 48.1
	PASHA $N = 80\%$	45.36 ± 1.38	3.6h ± 1.2h	2.5x	40.5 ± 47.7
	One-epoch baseline	41.42 ± 4.98	1.0h ± 0.0h	8.8x	1.0 ± 0.0
	Random baseline	20.75 ± 9.97	0.0h ± 0.0h	N/A	0.0 ± 0.0

***

Initial work on PASHA was done during an internship at AWS. However, significant amount of work was done outside of the internship as part of the PhD studies, including paper writing, extension of the initial algorithm, running experiments.

Appendix C

Appendix: Meta-Calibration

C.1 Learned L2 Regularization Values

We show the learned unit-wise L2 classifier regularization values in Figure C.1. The results show there is a large variability in the coefficients and they take both positive and negative values.

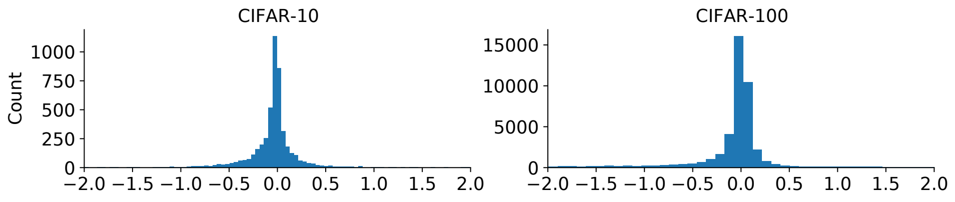


Figure C.1: Histograms of the learned unit-wise L2 classifier regularization values.

C.2 Training Times Analysis

We report the training times of the different approaches in Table C.1 (for simplicity we only include CE baseline as all baselines have similar training time). We see that even if training with Meta-Calibration takes longer, the overall training time remains manageable. If excellent calibration is key, longer training time is acceptable.

Table C.1: Training times in hours. We used one NVIDIA Titan X for each experiment. Meta-Calibration takes longer, but if excellent calibration is a priority, the additional time is acceptable.

Dataset	Model	CE	MC (Ours)
CIFAR-10	ResNet18	2.8h ± 0.1h	7.0h ± 0.2h
	ResNet50	8.9h ± 0.1h	21.9h ± 0.1h
	ResNet110	17.3h ± 0.3h	44.6h ± 7.1h
	WideResNet26-10	9.2h ± 0.3h	29.7h ± 0.2h
CIFAR-100	ResNet18	2.7h ± 0.1h	12.4h ± 0.2h
	ResNet50	8.9h ± 0.1h	26.4h ± 0.2h
	ResNet110	17.3h ± 0.2h	50.5h ± 1.0h
	WideResNet26-10	9.3h ± 0.4h	39.4h ± 6.7h
SVHN	ResNet18	4.4h ± 0.6h	9.6h ± 0.1h
20 Newsgroups	Global Pooling CNN	0.1h ± 0.0h	0.3h ± 0.1h

C.3 Cross-Domain Evaluation

C.3.1 Setup

To evaluate our framework's calibration performance under distribution shift, we use the CIFAR-C benchmark (Hendrycks and Dietterich, 2019). CIFAR-C contains a relatively large variety of image corruptions, such as adding fog, pixelation, changes to the brightness and many others (overall 19 corruption types). There are 5 levels of severity for each of them, which can be interpreted as providing 95 domains. We select 22 of these as test domains following (Zhang et al., 2021). These include impulse noise, motion blur, fog, and elastic transform at all levels of severity, and spatter and JPEG compression at the maximum level of severity. All other domains from (Hendrycks and Dietterich, 2019) are used during training and validation.

All approaches are trained with augmented (corrupted) data so that they can better generalize across domains. In each step we sample a corruption to use from the set of training corruptions. Meta-Calibration in particular samples a separate corruption for the inner and outer loop so that it can train meta-parameters that are more likely to generalize to new domains.

C.3.2 Results

We have evaluated our approach using CIFAR-C benchmark and ResNet18 model, and we show the results in Table C.2 and C.3. We include both the average across all test domains as well as the result on the most challenging domain (worst case). We report the mean and standard deviation across 3 repetitions. Meta-Calibration is still helpful for obtaining better calibration in the cross-domain case, but not to as large an extent as for clean data. While the efficacy of this algorithm likely depends on whether the domain-shifts seen during meta-training are representative in strength of those seen during meta-testing, we still view these as encouraging initial results that tools from multi-domain meta-learning can be adapted to address model calibration under domain shift. Studying how to more successfully exploit meta-learning for calibration in cross-domain scenarios is an interesting research question that the ML community can focus on in the future.

Table C.2: Test ECE (%, ↓) for our cross-domain experiments on CIFAR-C.

Dataset	Case	CE	LS	Brier	MMCE	FL	FLSD	MC (Ours)
CIFAR-10-C	Average	7.28 ± 0.03	3.13 ± 0.19	4.20 ± 0.81	4.12 ± 0.23	4.02 ± 0.26	4.13 ± 0.01	3.00 ± 0.28
	Worst case	15.55 ± 0.32	9.43 ± 0.65	9.19 ± 1.84	5.47 ± 0.78	4.97 ± 0.64	9.05 ± 0.88	10.97 ± 0.99
CIFAR-100-C	Average	7.77 ± 0.09	5.13 ± 0.05	4.89 ± 0.06	6.07 ± 0.59	6.33 ± 0.23	7.06 ± 0.54	5.26 ± 2.03
	Worst case	14.75 ± 1.09	9.42 ± 0.71	7.40 ± 0.32	8.30 ± 0.75	8.53 ± 0.16	8.94 ± 0.61	6.79 ± 2.37

Table C.3: Test error (%, ↓) for our cross-domain experiments on CIFAR-C.

Dataset	Case	CE	LS	Brier	MMCE	FL	FLSD	MC (Ours)
CIFAR-10-C	Average	10.60 ± 0.05	10.55 ± 0.19	10.67 ± 0.11	11.08 ± 0.18	11.05 ± 0.15	10.13 ± 0.04	11.22 ± 0.22
	Worst case	21.83 ± 0.49	21.78 ± 0.18	21.32 ± 0.42	23.52 ± 0.29	23.34 ± 0.79	21.03 ± 0.80	23.57 ± 0.78
CIFAR-100-C	Average	34.20 ± 0.09	35.26 ± 0.09	34.27 ± 0.23	34.26 ± 0.22	34.14 ± 0.38	33.32 ± 0.10	34.68 ± 0.13
	Worst case	57.10 ± 0.98	56.56 ± 0.91	56.95 ± 0.27	56.00 ± 0.20	55.87 ± 1.47	54.69 ± 0.44	55.76 ± 0.51

C.4 Comparison to SB-ECE

Soft-binned ECE (SB-ECE) approximates the binning operation in ECE so that it is differentiable and can be used as an auxiliary loss during training (Karandikar et al., 2021). Comparing DECE with SB-ECE, our DECE has both conceptual and empirical advantages. Conceptual advantages are as follows: 1) We also make the accuracy component of ECE differentiable, 2) SB-ECE binning estimate for the left-most and right-most bin can be very inaccurate as a result of using bin's center value, while our binning approach does not suffer from this. The empirical advantages are:

- DECE provides a closer approximation to ECE than SB-ECE as empirically evaluated in Figure C.2 and C.3. The quality of binning in SB-ECE is controlled by temperature parameter T, and we try both lower $T = 0.0001$ and higher temperatures $T = 0.01$. The results show DECE provides a significantly better approximation to ECE than SB-ECE regardless the value of the temperature. In fact, we see that the quality of SB-ECE approximation is relatively insensitive to the value of the temperature parameter T.

- Our Meta-Calibration with DECE leads to better calibration. Karandikar et al. (2021) propose SB-ECE and SAvUC (soft version of accuracy versus uncertainty calibration loss (Krishnan and Tickoo, 2020)) to be used as auxiliary losses to encourage better calibration. We evaluate SB-ECE and SAvUC on our benchmark setup, both in their original form (as a regularizer added to CE or Focal Loss) and in our meta-learning framework (as part of meta-objective for LS learning) in Table C.4. The results confirm the benefits of using Meta-Calibration with meta-objective that includes DECE. We additionally include comparison of error rates in Table C.5. The error rates remain similar to Meta-Calibration and the other baselines, although instabilities can occur that lead to noticeably worse error rates.

Karandikar et al. (2021) also introduce an alternative way of training called interleaved training. In such training each epoch is split into two and a separate set is used for training with respect to calibration. We have implemented and evaluated interleaved training as described in (Karandikar et al., 2021), and we report test ECE and error (%) in Table C.6 and C.7 respectively. The results suggest interleaved training generally leads to worse ECE than our approach, in most cases significantly worse. Interleaved training also leads to significantly worse error compared to our Meta-Calibration and the other baselines. We attribute the empirical benefits of Meta-Calibration to using the calibration objective for training only the label smoothing meta-parameters and also the specialised meta-training that uses an inner and outer loop.

Table C.4: Test ECE (%, ↓) – comparison of Meta-Calibration (MC) that uses DECE vs SB-ECE and SAvUC.

Dataset	Model	CE+SBECE	CE+SAvUC	FL3+SBECE	FL3+SAvUC	MC-SBECE	MC-SAvUC	MC (Ours)
CIFAR-10	ResNet18	5.37 ± 0.16	3.26 ± 0.02	1.31 ± 0.01	1.91 ± 0.09	2.63 ± 0.66	4.18 ± 0.49	1.17 ± 0.26
	ResNet50	3.29 ± 0.37	3.36 ± 0.10	1.85 ± 0.03	1.54 ± 0.17	2.51 ± 0.90	2.50 ± 0.42	1.09 ± 0.09
CIFAR-100	ResNet18	5.21 ± 1.15	5.68 ± 0.26	2.77 ± 0.08	2.58 ± 0.10	5.72 ± 0.21	5.49 ± 0.54	2.52 ± 0.35
	ResNet50	5.72 ± 0.39	12.34 ± 0.09	5.86 ± 0.01	4.93 ± 0.18	3.10 ± 0.13	7.53 ± 0.63	3.07 ± 0.18

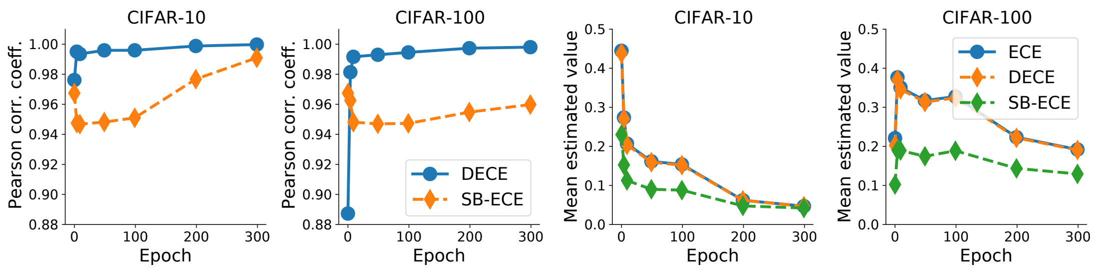


Figure C.2: $T = 0.01$: Pearson correlation coefficient between ECE/DECE and ECE/SB-ECE, and the mean estimated value of ECE, DECE and SB-ECE for CIFAR-10 and CIFAR-100 using ResNet18.

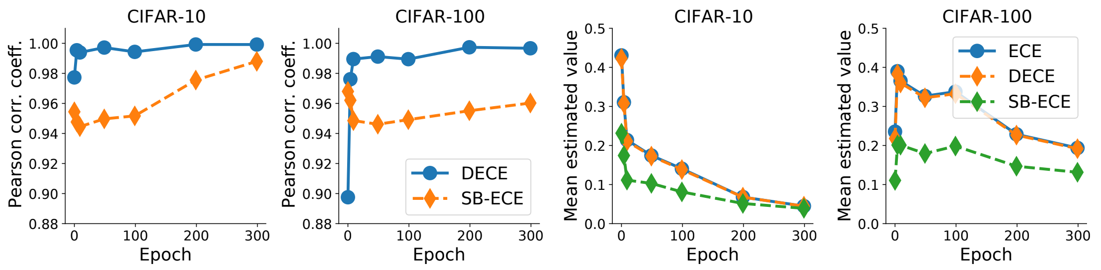


Figure C.3: $T = 0.0001$: Pearson correlation coefficient between ECE/DECE and ECE/SB-ECE, and the mean estimated value of ECE, DECE and SB-ECE for CIFAR-10 and CIFAR-100 using ResNet18.

Table C.5: Test error (%, $\downarrow$) – comparison of Meta-Calibration (MC) that uses DECE vs SB-ECE and SAvUC.

Dataset	Model	CE+SBECE	CE+SAvUC	FL3+SBECE	FL3+SAvUC	MC-SBECE	MC-SAvUC	MC (Ours)
CIFAR-10	ResNet18	8.97 ± 0.04	5.07 ± 0.06	5.16 ± 0.10	5.20 ± 0.11	5.10 ± 0.12	5.17 ± 0.25	5.22 ± 0.06
	ResNet50	12.39 ± 0.39	5.03 ± 0.10	5.04 ± 0.10	5.26 ± 0.23	5.30 ± 0.05	5.16 ± 0.21	5.46 ± 0.05
CIFAR-100	ResNet18	27.55 ± 0.18	22.64 ± 0.32	22.87 ± 0.34	22.88 ± 0.26	23.98 ± 0.14	23.83 ± 0.25	23.88 ± 0.20
	ResNet50	28.33 ± 0.39	22.09 ± 0.13	22.40 ± 0.21	22.02 ± 0.41	23.51 ± 0.45	23.18 ± 0.26	23.22 ± 0.48

Table C.6: Test ECE (%, ↓) – comparison of interleaved training and Meta-Calibration (MC).

Dataset	Model	CE+SBECE	CE+SAvUC	FL3+SBECE	FL3+SAvUC	MC (Ours)
CIFAR-10	ResNet18	4.81 ± 1.25	4.67 ± 0.12	6.90 ± 0.18	1.52 ± 0.08	1.17 ± 0.26
	ResNet50	3.44 ± 0.37	4.55 ± 0.15	6.38 ± 0.82	1.38 ± 0.31	1.09 ± 0.09
CIFAR-100	ResNet18	5.17 ± 1.23	10.04 ± 0.65	4.81 ± 1.05	1.73 ± 0.24	2.52 ± 0.35
	ResNet50	6.74 ± 0.29	11.56 ± 2.98	4.35 ± 0.48	2.86 ± 0.27	3.07 ± 0.18

Table C.7: Test error (%, ↓) – comparison of interleaved training and Meta-Calibration (MC).

Dataset	Model	CE+SBECE	CE+SAvUC	FL3+SBECE	FL3+SAvUC	MC (Ours)
CIFAR-10	ResNet18	19.09 ± 5.56	5.66 ± 0.13	14.22 ± 0.63	5.96 ± 0.13	5.22 ± 0.06
	ResNet50	13.51 ± 1.01	5.56 ± 0.13	12.63 ± 0.56	6.15 ± 0.45	5.46 ± 0.05
CIFAR-100	ResNet18	37.66 ± 2.85	26.94 ± 1.62	39.97 ± 5.71	26.64 ± 0.47	23.88 ± 0.20
	ResNet50	35.18 ± 1.24	24.75 ± 0.39	35.61 ± 2.22	25.45 ± 0.97	23.22 ± 0.48

Appendix D

Appendix: Meta Omnium

D.1 Full Dataset Details

We describe the full details of our multi-task meta-dataset in Table D.1 and provide further high-level details in this section.

- Classification datasets: We reuse datasets selected in the initial release of Meta-Album (Ullah et al., 2022). We split BCT (microscopy – bacteria) (Zieliński et al., 2017), BRD (large-animals – birds) (Piosenka, b) and CRS (vehicles – cars) (Krause et al., 2013) datasets into meta-training, in-domain meta-validation and in-domain meta-testing splits. We perform the splits randomly and in terms of classes – 70% for training, 15% validation and testing each. FLW (plants – flowers) (Nilsback and Zisserman, 2008), MD-Mix (OCR) (Sun et al., 2021) and PLK (small animals – plankton) (Heidi M. Sosik, 2015) datasets are used for out-domain meta-validation. PLT-VIL (plant diseases) (Geetharamani and Pandian, 2019), RESISC (remote sensing) (Cheng et al., 2017), SPT (human actions – sports) (Piosenka, a) and TEX (manufacturing – textures) (Fritz et al., 2004; Mallikarjuna et al., 2006; Kylberg, 2011; Lazebnik et al., 2005) for out-domain meta-testing. We use the middle version ("Mini") of these datasets as processed by the authors of Meta-Album (Ullah et al., 2022), which allows us to keep the overall size of Meta Omnium sufficiently small.
- Segmentation datasets: We first split FSS1000 (Li et al., 2020c) dataset into in-domain train, validation, and test sets, i.e. FSS1000-Trn, FSS1000-Val, FSS1000-Test. We use Vizwiz (Tseng et al., 2022) dataset for out-of-domain validation, and a modified version of Pascal 5i (Shaban et al., 2017) and PH2 (Mendonça

et al., 2013) datasets for out-of-domain testing. We exclude the object classes from the out-of-domain datasets that overlap with FSS1000 to ensure the classes during validation and testing are never seen during training.

- Keypoint estimation datasets: We use three keypoint datasets in our work, including animal pose (Cao et al., 2019), synthetic animal pose (Mu et al., 2020) and human pose (Andriluka et al., 2014). A single animal/human image is cropped from the original picture according to absolute maximum and minimum keypoint coordinates. The boundary is extended with 5 more pixels to avoid losing important information at object edges. Different keypoint datasets would have various target keypoints, so we cannot have a trivial solution like classification with a N-way K-shot setting, which stands for sampling K samples from N categories. Instead, we sample each keypoint task from one object category with only a fixed number of keypoints. In detail, we randomly select 5 keypoints per task, and train and fit the model to predict only 5 keypoints. This method leads to a general meta-learning keypoint prediction model that learns to predict corresponding keypoints from the limited support labels, which makes it possible for an arbitrary number of keypoint prediction tasks when conducted on more complex keypoint datasets.

- Regression datasets: We use regression datasets only for out-of-task (OOT) meta-test evaluation, so they are not used during meta-training. More specifically we use ShapeNet1D (Gao et al., 2022b), ShapeNet2D (Gao et al., 2022b), Distractor (Gao et al., 2022b) and Pascal1D datasets (Yin et al., 2020). Because regression problems typically require larger number of examples for adaptation, we use 5-times as many support examples compared to the other cases (e.g. instead of 5-shot we have 25-shot case). For our analysis experiments we consider the equivalent of variable 1-to-5-shot setting: variable 5-to-25-shot setting.

D.2 Additional Analysis

How Do Gradient-Based Meta-Learners Adapt Their Layers? A recent debate in few-shot meta-learning has been around whether gradient-based meta-learners really learn to adapt, or simply reuse features without adaptation. (Raghu et al., 2020) claimed that feature reuse was the dominant effect after measuring the representational change

Table D.1: Details of all task families included in Meta Omnium.

Task Family	Dataset Name	Domain	# Classes	# Images	Cardinality	Role	Size (MB)
Classification	BCT-Trn	Microscopy	23	920	(5)	Meta-train	8
	BRD-Trn	Bird	220	8800	(5)	Meta-train	72
	CRS-Trn	Car	137	5480	(5)	Meta-train	44
	BCT-Val	Microscopy	5	200	(5)	ID Meta-val	1.7
	BRD-Val	Bird	47	1880	(5)	ID Meta-val	15
	CRS-Val	Car	29	1160	(5)	ID Meta-val	9
	FLW	Flowers	102	4080	(5)	OD Meta-val	39
	MD-MIX	OCR	706	28240	(5)	OD Meta-val	479
	PLK	Plankton	86	3440	(5)	OD Meta-val	36
	BCT-Test	Microscopy	5	200	(5)	ID Meta-test	1.7
	BRD-Test	Bird	48	1920	(5)	ID Meta-test	16
	CRS-Test	Car	30	1200	(5)	ID Meta-test	10
	PLT-VIL	Plant Disease	38	1520	(5)	OD Meta-test	14
	RESISC	Remote Sensing	45	1800	(5)	OD Meta-test	17
	SPT	Sports	73	2920	(5)	OD Meta-test	27
	TEX	Textures	64	2560	(5)	OD Meta-test	26
Segmentation	FSS1000-Trn	Natural Image	520	5200	(2)	Meta-train	331
	FSS1000-Val	Natural Image	240	2400	(2)	ID Meta-val	150
	FSS1000-Test	Natural Image	240	2400	(2)	ID Meta-test	53
	Pascal 5i	Natural Image	6	7247	(2)	OD Meta-test	563
	Vizwiz	Natural Image	22	862	(2)	OD Meta-val	24
	PH2 (Skin)	Medical Image	3	200	(2)	OD Meta-test	114
Keypoint Regression	Animal pose - Trn	Animal	2	3237	(20, 2)	Meta-train	112
	Animal pose - Val	Animal	2	2038	(20, 2)	ID Meta-val	54
	Animal pose - Test	Animal	1	842	(20, 2)	ID Meta-test	18
	Synthetic Animal Pose	Synthetic Animal	2	20000	(18, 2)	OD Meta-val	627
	MPII	Human	1	28882	(16, 2)	OD Meta-test	265
Regression	ShapeNet1D-Test	Synthetic Image	60	3000	(2)	OOT Meta-test	8
	ShapeNet2D-Test	Synthetic Image	300	9000	(4)	OOT Meta-test	29
	Distractor-Test	Synthetic Image	200	7200	(2)	OOT Meta-test	93
	Pascal1D-Test	Synthetic Image	15	1500	(1)	OOT Meta-test	4

pre- and post-adaptation and finding that representational change was primarily in the output layer. We analyze this using Canonical Correlation Analysis (CCA) (Raghu et al., 2017; Morcos et al., 2018) for Meta Omnium, reporting the representational change of multi-task MAML by layer for each task family during meta-testing. From the results in Figure D.1, we observe that: 1) The degree of representational change varies substantially with tasks, 2) Similar to (Raghu et al., 2020), there is greater representational change at the later layers, especially the final output layer. However, significant amount of adaptation is done also in the earlier layers, which we attribute to the greater diversity of tasks and visual domains in Meta Omnium compared to the simple recognition episodes in miniImageNet studied by (Raghu et al., 2020).

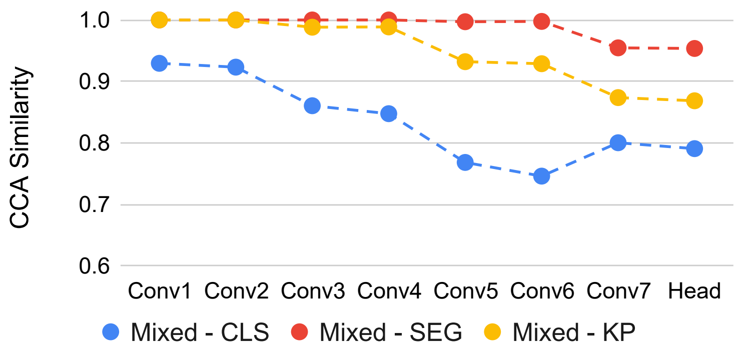


Figure D.1: Analysis of the layer adaptation by MAML in Meta Omnium.

D.3 Additional Experimental Details

D.3.1 Hyperparameter Optimization

The details of how we perform hyperparameter optimization (HPO) are described in Section 6.3.6, and in this section we provide additional details. The search space for HPO is as follows (note that momentum is only used if SGD optimizer is selected):

- MAML and Meta-Curvature: meta-learning rate $\in (10^{-4}, 10^{-1})$ (log scale), meta optimizer $\in \{\text{Adam}, \text{SGD}\}$, momentum $\in \{0.0, 0.9, 0.99\}$, inner-loop learning rate $\in (10^{-3}, 0.5)$ (log scale)
- Proto-MAML: same as MAML and also parameter $\lambda \in (0.01, 100)$ (log scale) that influences the prototype calculation in the case of keypoint estimation
- ProtoNet: meta-learning rate $\in (10^{-4}, 10^{-1})$ (log scale), meta optimizer $\in \{\text{Adam}, \text{SGD}\}$, momentum $\in \{0.0, 0.9, 0.99\}$ and distance temperature $\in (0.1, 10.0)$ (log scale) that is used for keypoint estimation
- DDRR: meta-learning rate $\in (10^{-4}, 10^{-1})$ (log scale), meta optimizer $\in \{\text{Adam}, \text{SGD}\}$, momentum $\in \{0.0, 0.9, 0.99\}$ and $\lambda \in (0.01, 100)$ (log scale)
- Proto-FineTuning: learning rate $\in (10^{-4}, 10^{-1})$ (log scale), optimizer $\in \{\text{Adam}, \text{SGD}\}$, momentum $\in \{0.0, 0.9, 0.99\}$ and $\lambda \in (0.01, 100)$ (log scale)
- FineTuning: learning rate $\in (10^{-4}, 10^{-1})$ (log scale), optimizer $\in \{\text{Adam}, \text{SGD}\}$ and momentum $\in \{0.0, 0.9, 0.99\}$
- Linear-Readout and TFS: same as FineTuning

After training a model with the candidate configuration for 5,000 iterations, we evaluate its validation performance. We use 100 tasks for evaluating the in-domain validation performance, and additional 100 tasks for evaluation of out-domain performance. As part of our multi-objective HPO, we minimize the validation error rates (or appropriate equivalent) and use each dataset as a separate objective. We perform HPO on the primary variable 1-to-5 shot setting. We use the same hyperparameters also for the 1-shot and 5-shot settings.

Our HPO is reasonably fast, and it generally takes between a few hours up to two days in the slowest cases (using a single NVIDIA 1080 Ti GPU with 12GB memory and using 4 CPUs). As a result, it is feasible to run the HPO even with modest resources when designing new approaches for our multi-task scenario. We provide the found hyperparameters within the released code.

D.3.2 Experimental Settings

Many of our experimental settings follow Meta-Album (Ullah et al., 2022)), whose code-base we have also used as the starting point. All approaches use one task in a meta-batch. We use 5 inner-loop steps during meta-training and 10 inner-loop steps during evaluation for MAML, Proto-MAML and Meta-Curvature. We use gradient-clipping of 5. DDRR uses an adjustment layer, the scale of which is initialized to 5.0 (with the adjust base set to 1.0). Proto-FineTuning, FineTuning, Linear-Readout and TFS use 20 fine-tuning steps during evaluation. The training minibatch size for these approaches is 16, while the testing minibatch size is 4. We use standard ImageNet normalization for segmentation tasks, but we do not use normalization in the other cases, following earlier work (Ullah et al., 2022).

We train each model for 30,000 iterations and evaluate the model on validation data after every 2,500 tasks, including at the beginning and the end (used for early stopping – model selection). We use 5-way tasks during both training and evaluation. The number of shots is between 1 and 5 during meta-training, and we consider 3 setups for evaluation: variable 1-to-5-shot (primary), 1-shot and 5-shot (presented in the appendix). The query size is 5 examples per category and this has been selected to be consistent across the different datasets. Validation uses 600 tasks for each of in-domain and out-domain evaluation. Testing uses 600 tasks per dataset to provide more rigorous evaluation.

During evaluation, we randomly initialize the top layer weights (classifier) to enable

Table D.2: In-domain single-task classification results. Mean test accuracy (%) and 95% confidence interval across test tasks.

Method	BCT-Test	BRD-Test	CRS-Test
MAML	78.09 ± 0.75	64.14 ± 1.17	33.87 ± 0.94
Proto-MAML	74.29 ± 0.83	51.99 ± 1.16	25.25 ± 0.61
Meta-Curvature	85.31 ± 0.66	71.99 ± 1.09	37.19 ± 0.95
ProtoNet	81.53 ± 0.66	75.39 ± 1.07	54.35 ± 1.12
DDRR	76.49 ± 0.80	69.25 ± 1.15	43.52 ± 1.03
Proto-FineTuning	42.92 ± 0.89	68.69 ± 1.17	40.81 ± 1.03
FineTuning	41.48 ± 0.85	52.51 ± 1.15	32.97 ± 0.78
Linear-Readout	45.44 ± 0.87	64.33 ± 1.12	36.11 ± 0.80
TFS	34.19 ± 0.86	35.14 ± 0.93	25.25 ± 0.71

any-way predictions, in line with previous literature (Ullah et al., 2022). We do this for the approaches that perform fine-tuning (e.g. MAML or Fine-Tuning baseline). Note that in approaches such as Proto-MAML the top layer is initialized using weights derived from the prototypes or ridge regression solution.

D.4 Detailed Per-Dataset Results

We include detailed per-dataset results (various-shot evaluation), first showing the single-task learning results for classification, segmentation and keypoint estimation, followed by multi-task learning results. In each case, we separately report the results for in-domain and out-of-domain evaluation. We also include detailed results for our out-of-task evaluation using regression datasets. Summary 1-shot and 5-shot results are included for the single and multi-task settings.

Table D.3: Out-of-domain single-task classification results. Mean test accuracy (%) and 95% confidence interval across test tasks.

Method	PLT_VIL	RESISC	SPT	TEX
MAML	62.69 ± 1.14	51.83 ± 1.06	46.24 ± 1.05	85.49 ± 1.02
Proto-MAML	46.59 ± 1.00	39.79 ± 0.94	35.24 ± 0.91	77.11 ± 1.14
Meta-Curvature	61.88 ± 1.07	52.00 ± 1.13	45.11 ± 1.06	86.55 ± 0.98
ProtoNet	59.68 ± 1.15	51.17 ± 1.04	43.65 ± 1.06	83.02 ± 1.03
DDRR	60.28 ± 1.19	48.70 ± 1.04	42.83 ± 1.04	83.17 ± 1.10
Proto-FineTuning	51.50 ± 1.27	41.92 ± 1.06	39.54 ± 1.03	69.77 ± 1.39
FineTuning	46.83 ± 1.13	41.37 ± 0.96	36.03 ± 0.90	68.39 ± 1.23
Linear-Readout	52.68 ± 1.02	46.24 ± 1.00	41.39 ± 0.94	73.45 ± 1.13
TFS	43.87 ± 1.03	36.36 ± 0.90	35.07 ± 0.91	52.54 ± 1.33

Table D.4: In-domain single-task segmentation results. Mean test mIoU (%) and 95% confidence interval across test tasks. Larger mIoU is better.

Method	FSS1000-Test
MAML	54.70 ± 1.68
Proto-MAML	46.40 ± 1.62
Meta-Curvature	65.57 ± 1.21
ProtoNet	75.84 ± 0.98
DDRR	66.71 ± 1.20
Proto-FineTuning	59.96 ± 1.55
FineTuning	50.52 ± 1.59
Linear-Readout	34.00 ± 1.85
TFS	42.80 ± 1.52

Table D.5: Out-of-domain single-task segmentation results. Mean test mIoU (%) and 95% confidence interval across test tasks. Larger mIoU is better.

Method	Pascal 5i	PH2
MAML	15.27 ± 1.29	68.88 ± 1.25
Proto-MAML	22.80 ± 1.19	65.46 ± 1.19
Meta-Curvature	27.66 ± 1.22	71.92 ± 0.80
ProtoNet	36.49 ± 1.39	77.82 ± 0.79
DDRR	29.07 ± 1.12	66.95 ± 0.77
Proto-FineTuning	21.03 ± 1.24	65.79 ± 1.21
FineTuning	16.23 ± 1.24	63.68 ± 1.11
Linear-Readout	5.67 ± 0.80	39.67 ± 1.91
TFS	15.45 ± 1.03	59.77 ± 1.18

Table D.6: In-domain single-task keypoint estimation results. Mean test PCK (%) and 95% confidence interval across test tasks. Larger PCK is better.

Method	Animal pose - Test
MAML	25.36 ± 0.93
Proto-MAML	23.63 ± 0.84
Meta-Curvature	43.47 ± 0.99
ProtoNet	27.79 ± 0.89
DDRR	20.53 ± 0.72
Proto-FineTuning	21.27 ± 0.74
FineTuning	25.69 ± 0.90
Linear-Readout	22.09 ± 0.74
TFS	20.98 ± 0.63

Table D.7: Out-of-domain single-task keypoint estimation results. Mean test PCK (%) and 95% confidence interval across test tasks. Larger PCK is better.

Method	MPII
MAML	33.04 ± 0.64
Proto-MAML	22.48 ± 0.64
Meta-Curvature	16.00 ± 0.39
ProtoNet	33.33 ± 0.71
DDRR	31.88 ± 0.63
Proto-FineTuning	33.10 ± 0.71
FineTuning	30.03 ± 0.53
Linear-Readout	26.86 ± 0.46
TFS	25.95 ± 0.52

Table D.8: In-domain multi-task learning results. Mean test score (%) and 95% confidence interval across test tasks. Larger score is better in all cases.

Method	FSS1000-Test	BCT-Test	BRD-Test	CRS-Test	Animal pose - Test
MAML	43.31 ± 1.60	89.05 ± 0.61	59.94 ± 0.99	28.19 ± 0.75	24.25 ± 0.79
Proto-MAML	53.03 ± 1.51	84.71 ± 0.70	59.79 ± 1.12	30.87 ± 0.87	21.63 ± 0.76
Meta-Curvature	42.60 ± 1.74	85.43 ± 0.66	76.85 ± 1.06	48.97 ± 1.09	18.21 ± 0.47
ProtoNet	63.32 ± 1.09	81.95 ± 0.68	72.31 ± 1.05	43.58 ± 1.04	20.10 ± 0.75
DDRR	40.39 ± 1.11	77.19 ± 0.73	51.47 ± 1.11	29.59 ± 0.83	22.77 ± 0.73
Proto-FineTuning	44.80 ± 1.62	41.11 ± 0.91	71.21 ± 1.21	44.85 ± 1.06	21.16 ± 0.74
FineTuning	41.31 ± 1.74	42.84 ± 0.89	55.41 ± 1.22	34.05 ± 0.81	18.05 ± 0.49
Linear-Readout	41.53 ± 1.66	39.07 ± 0.78	63.60 ± 1.04	35.18 ± 0.81	19.89 ± 0.52
TFS	38.66 ± 1.56	22.45 ± 0.54	22.74 ± 0.49	20.39 ± 0.39	14.09 ± 0.75

Table D.9: Out-of-domain multi-task learning results. Mean test score (%) and 95% confidence interval across test tasks. Larger score is better in all cases.

Method	PLT_VIL	RESISC	SPT	TEX	Pascal 5i	PH2	MPII
MAML	60.81 ± 1.11	48.19 ± 1.04	39.23 ± 0.91	85.59 ± 1.02	15.57 ± 1.11	59.28 ± 1.32	23.85 ± 0.47
Proto-MAML	65.18 ± 1.18	54.04 ± 1.10	49.84 ± 1.09	85.85 ± 0.95	21.90 ± 1.16	64.51 ± 1.04	33.34 ± 0.68
Meta-Curvature	70.98 ± 1.09	56.05 ± 1.15	51.09 ± 1.18	89.63 ± 0.80	13.29 ± 1.12	55.78 ± 1.52	25.29 ± 0.39
ProtoNet	60.55 ± 1.10	50.13 ± 1.04	41.92 ± 1.05	82.71 ± 1.00	30.46 ± 1.11	68.95 ± 0.87	33.00 ± 0.69
DDRR	53.19 ± 1.13	41.49 ± 0.98	35.73 ± 0.98	77.05 ± 1.19	20.19 ± 0.68	54.35 ± 0.78	30.08 ± 0.59
Proto-FineTuning	55.05 ± 1.21	44.17 ± 1.05	41.79 ± 1.04	71.64 ± 1.33	15.19 ± 1.06	60.47 ± 1.30	30.04 ± 0.60
FineTuning	50.88 ± 1.16	43.59 ± 1.00	38.07 ± 0.94	72.41 ± 1.18	11.68 ± 1.02	60.58 ± 1.39	20.46 ± 0.33
Linear-Readout	49.78 ± 1.00	44.57 ± 0.98	40.83 ± 0.97	68.58 ± 1.12	14.37 ± 1.07	50.73 ± 1.96	23.47 ± 0.35
TFS	23.15 ± 0.51	23.01 ± 0.47	22.45 ± 0.49	26.63 ± 0.71	13.12 ± 0.96	58.41 ± 1.25	11.04 ± 0.29

Table D.10: Evaluation of multi-task models on out-of-task regression datasets, using variable 5-to-25-shot episodes. Lower value is better.

Method	ShapeNet2D-Test	Distractor-Test	ShapeNet1D-Test	Pascal1D-Test
MAML	95.44 ± 2.82	38.68 ± 0.60	54.20 ± 2.10	2.88 ± 0.10
Proto-MAML	69.94 ± 1.50	39.58 ± 0.69	63.17 ± 2.33	51.74 ± 1.15
Meta-Curvature	70.55 ± 2.32	38.73 ± 0.66	43.17 ± 2.03	12.04 ± 0.61
ProtoNet	63.50 ± 1.15	38.53 ± 0.58	84.53 ± 1.92	2.53 ± 0.06
DDRR	64.41 ± 1.64	41.67 ± 0.68	46.08 ± 1.90	2.11 ± 0.07
Proto-FineTuning	61.94 ± 2.25	39.44 ± 0.69	58.33 ± 2.54	4.17 ± 0.18
FineTuning	63.36 ± 1.54	51.52 ± 1.44	81.36 ± 2.12	6.54 ± 0.31
Linear-Readout	68.66 ± 1.94	40.53 ± 0.71	46.93 ± 2.05	2.62 ± 0.09
TFS	133.67 ± 2.67	95.36 ± 0.91	88.25 ± 1.88	6.63 ± 0.31

Table D.11: 5-way 1-shot results, reporting the same metrics as in our primary table with variable-shot results.

		Classification		**Segmentation**		**Keypoints**		**Average Rank**		
		ID	OOD	ID	OOD	ID	OOD	ID	OOD	AVG
Single-Task	MAML	50.8	50.8	44.1	33.6	34.7	33.4	4.3	4.0	4.2
	Proto-MAML	53.5	52.4	46.0	39.2	23.5	14.8	4.7	4.7	4.7
	Meta-Curvature	58.0	51.6	60.5	40.1	38.1	16.1	**1.7**	4.3	**3.0**
	ProtoNet	61.7	50.2	73.7	52.5	22.5	31.9	2.7	**3.7**	3.2
	DDRR	54.7	48.9	60.1	42.2	22.1	32.2	4.7	4.0	4.3
	Proto-FineTuning	47.0	50.4	50.5	36.6	22.4	32.8	5.7	4.3	5.0
	FineTuning	35.5	43.2	42.4	36.6	34.6	33.8	6.0	4.7	5.3
	Linear-Readout	46.2	48.0	30.3	18.3	26.5	26.7	6.7	7.3	7.0
	TFS	27.2	36.4	31.5	30.3	19.5	20.0	8.7	8.0	8.3
Multi-Task	MAML	56.4	54.0	35.0	28.8	29.1	29.0	3.0	4.3	3.7
	Proto-MAML	50.5	51.7	43.6	34.2	22.5	32.7	3.0	**2.3**	2.7
	Meta-Curvature	62.4	56.1	29.2	25.6	16.0	22.3	5.7	5.7	5.7
	ProtoNet	60.8	50.8	59.2	42.2	22.5	31.9	**2.3**	2.7	**2.5**
	DDRR	47.3	46.2	36.4	33.2	19.6	29.3	5.0	4.7	4.8
	Proto-FineTuning	47.2	52.1	33.6	33.9	19.2	28.1	6.3	3.7	5.0
	FineTuning	37.2	43.9	38.2	33.4	23.5	23.8	4.3	5.7	5.0
	Linear-Readout	40.3	43.7	24.5	22.3	21.3	22.8	7.0	8.0	7.5
	TFS	22.0	24.2	30.2	30.2	9.4	9.3	8.3	8.0	8.2

Table D.12: 5-way 5-shot results, reporting the same metrics as in our primary table with variable-shot results.

		Classification		**Segmentation**		**Keypoints**		**Average Rank**		
		ID	OOD	ID	OOD	ID	OOD	ID	OOD	AVG
Single-Task	MAML	63.2	67.7	57.6	45.0	22.2	33.6	4.7	3.3	4.0
	Proto-MAML	57.0	52.3	49.9	46.7	22.3	29.6	5.0	5.7	5.3
	Meta-Curvature	69.0	67.2	73.5	53.5	43.7	16.3	1.7	4.3	3.0
	ProtoNet	74.3	64.0	75.9	55.9	29.4	33.9	**1.3**	**2.0**	**1.7**
	DDRR	68.0	65.7	69.4	50.5	22.0	32.0	4.3	3.7	4.0
	Proto-FineTuning	52.9	52.0	65.9	48.7	22.2	33.9	5.0	4.3	4.7
	FineTuning	43.8	50.0	55.3	42.7	22.1	33.1	6.7	6.3	6.5
	Linear-Readout	53.6	55.0	32.3	32.8	20.0	27.1	8.0	7.3	7.7
	TFS	33.8	44.4	47.4	40.5	20.9	28.2	8.3	8.0	8.2
Multi-Task	MAML	68.3	72.1	52.0	42.1	20.7	31.4	3.3	3.0	3.2
	Proto-MAML	67.0	71.2	63.0	48.5	23.2	34.0	**2.7**	2.3	**2.5**
	Meta-Curvature	76.7	73.8	49.7	38.1	19.6	27.7	4.3	5.3	4.8
	ProtoNet	71.0	63.4	64.7	52.4	19.7	34.5	3.0	**2.0**	**2.5**
	DDRR	58.0	59.2	42.5	38.3	23.5	30.0	4.7	6.0	5.3
	Proto-FineTuning	52.7	51.3	50.4	40.6	21.3	32.5	4.3	5.0	4.7
	FineTuning	47.8	54.2	46.6	41.5	18.1	22.3	7.3	6.0	6.7
	Linear-Readout	48.2	50.9	45.8	38.4	19.7	25.5	6.3	7.0	6.7
	TFS	22.4	23.9	40.7	38.3	15.8	12.1	9.0	8.3	8.7

Bibliography

Abdollahzadeh, M., Malekzadeh, T., and Cheung, N.-M. M. (2021). Revisit multimodal meta-learning through the lens of multi-task learning. In *NeurIPS*.

Afrasiyabi, A., Larochelle, H., Lalonde, J.-F., and Gagné, C. (2022). Matching feature sets for few-shot image classification. In *CVPR*.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *KDD*.

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. (2018). Continuous adaptation via meta-learning in nonstationary and competitive environments. In *ICLR*.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. (2022). Flamingo: a visual language model for few-shot learning. In *NeurIPS*.

Alet, F., Schneider, M. F., Lozano-Perez, T., and Kaelbling, L. P. (2020). Meta-learning curiosity algorithms. In *ICLR*.

Algan, G. and Ulusoy, I. (2022). Metalabelnet: Learning to generate soft-labels from noisy-labels. *IEEE Transactions on Image Processing*.

Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 2d human pose estimation: New benchmark and state of the art analysis. In *CVPR*.

Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and Freitas, N. d. (2016). Learning to learn by gradient descent by gradient descent. In *NIPS*.

Antoniou, A., Edwards, H., and Storkey, A. (2019). How to train your MAML. In *ICLR*.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Annual Foundations of Computer Science*.

Baevski, A., Hsu, W.-N., Xu, Q., Babu, A., Gu, J., and Auli, M. (2022). Data2vec: A general framework for self-supervised learning in speech, vision and language. In *ICML*.

Balaji, Y., Sankaranarayanan, S., and Chellappa, R. (2018). MetaReg: towards domain generalization using meta-regularization. In *NeurIPS*.

Baldwin, J. M. (1896). A new factor in evolution. *The American Naturalist*.

Baydin, A. G., Pearlmutter, B. A., and Siskind, J. M. (2018). Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*.

Bechtle, S., Molchanov, A., Chebotar, Y., Grefenstette, E., Righetti, L., Sukhatme, G., and Meier, F. (2021). Meta learning via learned loss. *ICPR*.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. (1997). On the optimization of a synaptic learning rule. In *Optimality in Biological and Artificial Networks?*

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural Computation*.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *NIPS*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2019). Meta-learning with differentiable closed-form solvers. In *ICLR*.

Biggs, J. B. (1985). The role of metalearning in study processes. *British Journal of Educational Psychology*.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J. (2020). Fast differentiable sorting and ranking. In *ICML*.

Bohdal, O., Balles, L., Wistuba, M., Ermis, B., Archambeau, C., and Zappella, G. (2023a). PASHA: Efficient HPO and NAS with progressive resource allocation. In *ICLR*.

Bohdal, O., Li, D., and Hospedales, T. (2022). Feed-forward source-free domain adaptation via class prototypes. In *ECCV OOD-CV Workshop*.

Bohdal, O., Li, D., Hu, S. X., and Hospedales, T. (2024). Feed-forward latent domain adaptation. In *WACV*.

Bohdal, O., Tian, Y., Zong, Y., Chavhan, R., Li, D., Gouk, H., Guo, L., and Hospedales, T. (2023b). Meta omnium: A benchmark for general-purpose learning-to-learn. In *CVPR*.

Bohdal, O., Yang, Y., and Hospedales, T. (2020). Flexible dataset distillation: learn labels instead of images. In *NeurIPS Workshop on Meta-Learning*.

Bohdal, O., Yang, Y., and Hospedales, T. (2021). EvoGrad: efficient gradient-based meta-learning and hyperparameter optimization. In *NeurIPS*.

Bohdal, O., Yang, Y., and Hospedales, T. (2023c). Meta-Calibration: Learning of model calibration using differentiable expected calibration error. In *Transactions on Machine Learning Research*.

Bojar, O., Chatterjee, R., Federmann, C., Haddow, B., Huck, M., Hokamp, C., Koehn, P., Logacheva, V., Monz, C., Negri, M., Post, M., Scarton, C., Specia, L., and Turchi, M. (2015). Findings of the 2015 workshop on statistical machine translation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to end learning for self-driving cars. In *arXiv*.

Bornschein, J., Visin, F. V., and Osindero, S. (2020). Small data, big decisions: Model selection in the small-data regime. In *ICML*.

Bragg, J., Cohan, A., Lo, K., and Beltagy, I. (2021). Flex: Unifying evaluation for few-shot nlp. In *NeurIPS*.

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*.

Brown, A. L. and Kane, M. J. (1988). Preschool children can learn to transfer: Learning to learn and learning from example. *Cognitive Psychology*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *NeurIPS*.

Cao, J., Tang, H., Fang, H.-S., Shen, X., Lu, C., and Tai, Y.-W. (2019). Cross-domain adaptation for animal pose estimation. In *ICCV*.

Carrell, A., Mallinar, N., Lucas, J., and Nakkiran, P. (2022). The calibration generalization gap. In *ICML 2022 Workshop on Distribution-Free Uncertainty Quantification*.

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., and Elhadad, N. (2015). Intelligible models for healthcare. In *KDD*.

Chan, S. C., Santoro, A., Lampinen, A. K., Wang, J. X., Singh, A., Richemond, P. H., McClelland, J., and Hill, F. (2022). Data distributional properties drive emergent few-shot learning in transformers. In *NeurIPS*.

Chavhan, R., Stuehmer, J., Heggan, C., Yaghoobi, M., and Hospedales, T. (2023). Amortised invariance learning for contrastive self-supervision. In *ICLR*.

Chen, C., Zhang, Y., Liu, X., and Coates, M. (2023a). Bidirectional learning for offline model-based biological sequence design. In *ICML*.

Chen, C. S., Chen, X., Ma, C., Liu, Z., and Liu, X. (2022). Gradient-based bi-level optimization for deep learning: A survey. In *arXiv*.

Chen, S., Du, Y., Mettes, P., and Snoek, C. G. (2023b). Multi-label meta weighting for long-tailed dynamic scene graph generation. In *ACM International Conference on Multimedia Retrieval*.

Chen, W., Tripp, A., and Hernández-Lobato, J. M. (2023c). Meta-learning adaptive deep kernel gaussian processes for molecular property prediction. In *ICLR*.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Tech, G., Wang, Y.-C. F., Huang, J.-B., and Tech, V. (2019). A closer look at few-shot classification. In *ICLR*.

Cheng, G., Han, J., and Lu, X. (2017). Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*.

Choe, S. K., Mehta, S. V., Ahn, H., Neiswanger, W., Xie, P., Strubell, E., and Xing, E. (2023a). Making scalable meta learning practical. In *NeurIPS*.

Choe, S. K., Neiswanger, W., Xie, P., and Xing, E. (2023b). Betty: An automatic differentiation library for multilevel oimization. In *ICLR*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2023). Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*.

Clune, J. (2019). Ai-gas: Ai-generating algorithms, an alternate paradigm for producing general artificial intelligence. In *arXiv*.

Csurka, G. (2017). *A comprehensive survey on domain adaptation for visual applications*. Springer International Publishing.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation policies from data. In *CVPR*.

Dai, D., Sun, Y., Dong, L., Hao, Y., Sui, Z., and Wei, F. (2023). Why can GPT learn in-context? Language models secretly perform gradient descent as meta optimizers. In *ACL Findings*.

Deng, J., Dong, W., Socher, R., Li, L.-j., Li, K., and Fei-fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *CVPR*.

Dery, L. M., Michel, P., Talwalkar, A., and Neubig, G. (2022). Should we be pre-training? an argument for end-task aware training as an alternative. In *ICLR*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *ACL*.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., and Sui, Z. (2023). A survey for in-context learning. In *arXiv*.

Dong, X. and Yang, Y. (2020). NAS-Bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*.

Dumoulin, V., Houlsby, N., Evci, U., Zhai, X., Goroshin, R., Gelly, S., and Larochelle, H. (2021). A unified few-shot classification benchmark to compare transfer and meta learning approaches. In *NeurIPS Datasets and Benchmarks*.

Dutt, R., Bohdal, O., Tsaftaris, S. A., and Hospedales, T. (2024). Fairtune: Optimizing parameter efficient fine tuning for fairness in medical image analysis. In *ICLR*.

Dvornik, N., Schmid, C., and Mairal, J. (2020). Selecting relevant features from a universal representation for few-shot classification. In *ECCV*.

Egele, R., Guyon, I., Sun, Y., and Balaprakash, P. (2023). Is one epoch all you need for multi-fidelity hyperparameter optimization? In *arXiv*.

El-Sappagh, S., Alonso-Moral, J. M., Abuhmed, T., Ali, F., and Bugarín-Diz, A. (2023). Trustworthy artificial intelligence in alzheimer's disease: state of the art, opportunities, and challenges. *Artificial Intelligence Review*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*.

Elsken, T., Staffler, B., Zela, A., Metzen, J. H., and Hutter, F. (2021). Bag of tricks for neural architecture search. In *CVPR Workshop on Neural Architecture Search*.

Emde, C., Pinto, F., Lukasiewicz, T., Torr, P., and Bibi, A. (2023). Certified calibration: Bounding worst-case calibration under adversarial attacks. In *ICML Workshop on New Frontiers in Adversarial Machine Learning*.

Ericsson, L., Gouk, H., and Hospedales, T. M. (2022). Why do self-supervised models transfer? Investigating the impact of invariance on downstream tasks. In *BMVC*.

Eustratiadis, P., Dudziak, Ł., Li, D., and Hospedales, T. (2024). Neural fine-tuning search for few-shot learning. In *ICLR*.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *ICML*.

Fang, Z., Wang, X., Li, H., Liu, J., Hu, Q., and Xiao, J. (2023). Fastrecon: Few-shot industrial anomaly detection via fast feature reconstruction. In *ICCV*.

Feurer, M. and Hutter, F. (2019). *Hyperparameter optimization*. Springer International Publishing.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*.

Flennerhag, S., Schroecker, Y., Zahavy, T., van Hasselt, H., Silver, D., and Singh, S. (2022). Bootstrapped meta-learning. In *ICLR*.

Fritz, M., Hayman, E., Caputo, B., and Eklundh, J. (2004). THE KTH-TIPS database.

Gao, B., Gouk, H., and Hospedales, T. M. (2021). Searching for robustness: Loss learning for noisy classification tasks. In *ICCV*.

Gao, B., Gouk, H., Yang, Y., and Hospedales, T. (2022a). Loss function learning for domain generalization by implicit gradient. In *ICML*.

Gao, N., Ziesche, H., Vien, N. A., Volpp, M., and Neumann, G. (2022b). What matters for meta-learning vision regression tasks? In *CVPR*.

Garcia, F. and Thomas, P. S. (2019). A meta-mdp approach to exploration for lifelong reinforcement learning. In *NeurIPS*.

Gawlikowski, J., Tassi, C. R. N., Ali, M., Lee, J., Humt, M., Feng, J., Kruspe, A. M., Triebel, R., Jung, P., Roscher, R., Shahzad, M., Yang, W., Bamler, R., and Zhu, X. (2021). A survey of uncertainty in deep neural networks. In *arXiv*.

Geetharamani, G. and Pandian, J. A. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers and Electrical Engineering*.

Gemini Team Google (2023). Gemini: a family of highly capable multimodal models. In *arXiv*.

Ghiasi, G., Zoph, B., Cubuk, E. D., Le, Q. V., and Lin, T.-Y. (2021). Multi-task self-training for learning general representations. In *ICCV*.

Gonzalez, S. and Miikkulainen, R. (2020). Improved training speed, accuracy, and data utilization through loss function optimization. In *IEEE Congress on Evolutionary Computation*.

Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. (2019). Generalized inner loop meta-learning. In *arXiv*.

Griewank, A. (1993). Some bounds on the complexity of gradients, Jacobians, and Hessians. In *Complexity in Numerical Optimization*.

Gulrajani, I. and Lopez-Paz, D. (2021). In search of lost domain generalization. In *ICLR*.

Guo, C., Pleiss, G., Sun, Y., and Weinberger, K. Q. (2017). On calibration of modern neural networks. In *ICML*.

Guo, Y., Codella, N. C., Karlinsky, L., Codella, J. V., Smith, J. R., Saenko, K., Rosing, T., and Feris, R. (2020). A broader study of cross-domain few-shot learning. In *ECCV*.

Guo, Y., Du, R., Dong, Y., Hospedales, T., Song, Y.-Z., and Ma, Z. (2023). Task-aware adaptive learning for cross-domain few-shot learning. In *ICCV*.

Harlow, H. F. (1949). The formation of learning sets. *Psychological Review*.

Hataya, R. and Yamada, M. (2023). Nyström method for accurate and scalable implicit differentiation. In *AISTATS*.

Hataya, R., Zdenek, J., Yoshizoe, K., and Nakayama, H. (2022). Meta approach to data augmentation optimization. In *WACV*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

Heggan, C., Budgett, S., Hospedales, T., and Yaghoobi, M. (2022). Metaaudio: A few-shot audio classification benchmark. In *ICANN*.

Heidi M. Sosik, Emily E. Peacock, E. F. B. (2015). Annotated plankton images - data set for developing and evaluating classification methods.

Hellan, S. P., Shen, H., Aubet, F.-X., Salinas, D., and Klein, A. (2023). Obeying the order: Introducing ordered transfer hyperparameter optimisation. In *AutoML Conference Workshop Track*.

Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. In *ICLR*.

Hendrycks, D. and Gimpel, K. (2017). A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *ICLR*.

Hong, S., Cho, S., Nam, J., Lin, S., and Kim, S. (2022). Cost aggregation with 4d convolutional swin transformer for few-shot segmentation. In *ECCV*.

Horn, G. V., Aodha, O. M., Song, Y., Cui, Y., Sun, C., Shepard, A., Adam, H., Perona, P., and Belongie, S. (2018). The iNaturalist species classification and detection dataset. In *CVPR*.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2021). Meta-learning in neural networks: a survey. *Transactions on Pattern Analysis and Machine Intelligence*.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Jonathan Ho, O., and Abbeel, P. (2018). Evolved policy gradients. In *NeurIPS*.

Hu, S., Li, D., Stuehmer, J., Kim, M., and Hospedales, T. (2022). Pushing the limits of simple pipelines for few-shot learning. In *CVPR*.

Huang, C., Zhai, S., Talbott, W. A., Bautista, M. Á., Sun, S., Guestrin, C., and Susskind, J. M. (2019). Addressing the loss-metric mismatch with adaptive loss alignment. In *ICML*.

Huang, J., Park, S., and Simeone, O. (2023). Calibration-aware bayesian learning. In *arXiv*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*.

Hutter, F., Kotthoff, L., and Vanschoren, J. (2019). *Automated machine learning*. Springer.

Ivkin, N., Karnin, Z., Perrone, V., and Zappella, G. (2021). Cost-aware adversarial best arm identification. In *ICLR NAS Workshop*.

Iwata, T. and Kumagai, A. (2023). Meta-learning to calibrate gaussian processes with deep kernels for regression uncertainty estimation. In *arXiv*.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castañeda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., and Graepel, T. (2019). Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*.

Jiang, X. and Deng, X. (2023). Knowledge reverse distillation based confidence calibration for deep neural networks. *Neural Processing Letters*.

Jiang, X., Osl, M., Kim, J., and Ohno-Machado, L. (2012). Calibrating predictive model estimates to support personalized medicine. *Journal of the American Medical Informatics Association*.

Judge, T., Bernard, O., Porumb, M., Chartsias, A., Beqiri, A., and Jodoin, P.-M. (2022). Crisp - reliable uncertainty estimation for medical image segmentation. In *MICCAI*.

Kadra, A., Janowski, M., Wistuba, M., and Grabocka, J. (2023). Scaling laws for hyperparameter optimization. In *NeurIPS*.

Kaiser, L., Gomez, A. N., Shazeer, N., Vaswani, A., Parmar, N., Jones, L., and Uszkoreit, J. (2017). One model to learn them all. In *arXiv*.

Kamani, M. M., Farhang, S., Mahdavi, M., and Wang, J. Z. (2019). Targeted meta-learning for critical incident detection in weather data. In *ICML Workshop on Climate Change: How Can AI Help?*

Karandikar, A., Cain, N., Tran, D., Lakshminarayanan, B., Shlens, J., Mozer, M. C., and Roelofs, B. (2021). Soft calibration objectives for neural networks. In *NeurIPS*.

Karnin, Z., Koren, T., and Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. In *ICML*.

Kim, S. and Yun, S.-Y. (2020). Task calibration for distributional uncertainty in few-shot classification. In *OpenReview*.

Kingma, D. P. and Ba, J. (2015). Adam: a method for stochastic optimization. In *ICLR*.

Kirsch, L., van Steenkiste, S., and Schmidhuber, J. (2020). Improving generalization in meta reinforcement learning using learned objectives. In *ICLR*.

Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization. In *arXiv*.

Klein, A., Tiao, L. C., Lienart, T., Archambeau, C., and Seeger, M. (2020). Model-based asynchronous hyperparameter and neural architecture search. In *arXiv*.

Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3D object representations for fine-grained categorization. In *ICCV Workshops*.

Krishnan, R. and Tickoo, O. (2020). Improving model calibration with accuracy versus uncertainty optimization. In *NeurIPS*.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Technical report.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS*.

Kull, M., Perello Nieto, M., Kängsepp, M., Silva Filho, T., Song, H., and Flach, P. (2019). Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In *NeurIPS*.

Kumar, A., Liang, P., and Ma, T. (2019). Verified uncertainty calibration. In *NeurIPS*.

Kumar, A., Sarawagi, S., and Jain, U. (2018). Trainable calibration measures for neural networks from kernel mean embeddings. In *ICML*.

Kwon, J., Kwon, D., Wright, S., and Nowak, R. D. (2023). A fully first-order method for stochastic bilevel optimization. In *ICML*.

Kylberg, G. (2011). The Kylberg texture dataset v. 1.0. Technical report.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*.

Lang, K. (1995). Newsweeder: Learning to filter netnews. In *ICML*.

Larsen, J., Hansen, L. K., Svarer, C., and Ohlsson, M. (1996). Design and regularization of neural networks: the optimal use of a validation set. In *Neural Networks for Signal Processing – Proceedings of the IEEE Workshop*.

Lazebnik, S., Schmid, C., and Ponce, J. (2005). A sparse texture representation using local affine regions. *Transactions on Pattern Analysis and Machine Intelligence*.

LeCun, Y., Cortes, C., and Burges, C. (1998). MNIST handwritten digit database.

LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: applications of neural network chips and automatic learning. *IEEE Communications Magazine*.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex pptimization. In *CVPR*.

Lefaudeux, B., Massa, F., Liskovich, D., Xiong, W., Caggiano, V., Naren, S., Xu, M., Hu, J., Tintore, M., and Zhang, S. (2021). xformers: A modular and hackable transformer modelling library.

Lei, S. and Tao, D. (2023). A comprehensive survey to dataset distillation. In *arXiv*.

Levine, W., Pikus, B., Raja, P. V., and Amat, F. (2023). Enabling calibration in the zero-shot inference of large vision-language models. In *ICLR 2023 Workshop on Pitfalls of limited data and computation for Trustworthy ML*.

Li, D., Gouk, H., and Hospedales, T. (2022). Finding lost DG: Explaining domain generalization via model complexity. In *arXiv*.

Li, D. and Hospedales, T. (2020). Online meta-learning for multi-source and semi-supervised domain adaptation. In *ECCV*.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. (2018a). Learning to Generalize: Meta-Learning for Domain Generalization. In *AAAI*.

Li, J., Wong, Y., Zhao, Q., and Kankanhalli, M. S. (2019a). Learning to Learn from Noisy Labeled Data. In *CVPR*.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Hardt, M. H., Recht, B., and Talwalkar, A. (2020a). A system for massively parallel hyperparameter tuning. In *MLSys*.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018b). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*.

Li, L. and Talwalkar, A. (2020). Random search and reproducibility for neural architecture search. In *UAI*.

Li, L., Zhang, Y., and Wang, S. (2023). The euclidean space is evil: hyperbolic attribute editing for few-shot image generation. In *ICCV*.

Li, R., Bohdal, O., Mishra, R. K., Kim, H., Li, D., Lane, N. D., and Hospedales, T. M. (2021a). A channel coding benchmark for meta-learning. In *NeurIPS Datasets and Benchmarks*.

Li, W.-H., Foo, C.-S., and Bilen, H. (2020b). Learning to impute: a general framework for semi-supervised learning. In *arXiv*.

Li, W.-H., Liu, X., and Bilen, H. (2021b). Universal representation learning from multiple domains for few-shot classification. In *ICCV*.

Li, X., Wei, T., Chen, Y. P., Tai, Y.-W., and Tang, C.-K. (2020c). Fss-1000: A 1000-class dataset for few-shot segmentation. In *CVPR*.

Li, Y., Hu, G., Wang, Y., Hospedales, T. M., Robertson, N. M., and Yang, Y. (2020d). DADA: differentiable automatic data augmentation. In *ECCV*.

Li, Y., Yang, Y., Zhou, W., and Hospedales, T. M. (2019b). Feature-critic networks for heterogeneous domain generalization. In *ICML*.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-SGD: learning to learn quickly for few-shot learning. In *arXiv*.

Liang, S., Li, Y., and Srikant, R. (2018). Enhancing the reliability of out-of-distribution image detection in neural networks. In *ICLR*.

Lin, M., Chen, Q., and Yan, S. (2014). Network in network. In *ICLR*.

Lin, S., Zhou, P., Liang, X., Tang, J., Zhao, R., Chen, Z., and Lin, L. (2021). Graph-evolving meta-learning for low-resource medical dialogue generation. In *AAAI*.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *ICCV*.

Liu, B., Ye, M., Wright, S., Stone, P., and Liu, Q. (2022). Bome! bilevel optimization made easy: A simple first-order approach. In *NeurIPS*.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2023). Visual instruction tuning. In *NeurIPS*.

Liu, H., Simonyan, K., and Yang, Y. (2019a). DARTS: differentiable architecture search. In *ICLR*.

Liu, S., Borovykh, A., Grzelak, L. A., and Oosterlee, C. W. (2019b). A neural network-based framework for financial model calibration. *Journal of Mathematics in Industry*, 9.

Liu, S., Davison, A. J., and Johns, E. (2019c). Self-supervised generalisation with meta auxiliary learning. In *NeurIPS*.

Liu, Y., Lee, J., Zhu, L., Chen, L., Shi, H., and Yang, Y. (2021). A multi-mode modulator for multi-domain few-shot classification. In *ICCV*.

Lorraine, J. and Duvenaud, D. (2018). Stochastic hyperparameter optimization through hypernetworks. In *arXiv*.

Lorraine, J., Vicol, P., and Duvenaud, D. (2020). Optimizing millions of hyperparameters by implicit differentiation. In *AISTATS*.

Lu, C. and Koniusz, P. (2022). Few-shot keypoint detection with uncertainty learning for unseen species. In *CVPR*.

Luketina, J., Berglund, M., Klaus Greff, A., and Raiko, T. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *ICML*.

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. In *ICML*.

Mallik, N., Bergman, E., Hvarfner, C., Stoll, D., Janowski, M., Lindauer, M., Nardi, L., and Hutter, F. (2023). Priorband: Practical hyperparameter optimization in the age of deep learning. In *NeurIPS*.

Mallikarjuna, P., Targhi, A. T., Fritz, M., Hayman, E., Caputo, B., and Eklundh, J. (2006). The KTH-TIPS 2 database.

Marcus, G. F. (2018). Deep learning: A critical appraisal. In *arXiv*.

Marques, J. M. C., Zhai, A., Wang, S., and Hauser, K. (2023). On the overconfidence problem in semantic 3d mapping. In *arXiv*.

Mehrabi, N., Morstatter, F., Saxena, N. A., Lerman, K., and Galstyan, A. G. (2019). A survey on bias and fairness in machine learning. *ACM Computing Surveys*.

Mehta, Y., White, C., Zela, A., Krishnakumar, A., Zabergja, G., Moradian, S., Safari, M., Yu, K., and Hutter, F. (2022). NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy. In *ICLR*.

Mendonça, T., Ferreira, P. M., Marques, J. S., Marçal, A. R. S., and Rozeira, J. (2013). PH2 - a dermoscopic image database for research and benchmarking. In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*.

Metz, L., Brain, G., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. (2019). Meta-learning update rules for unsupervised representation learning. In *ICLR*.

Metz, L., Harrison, J., Freeman, C. D., Merchant, A., Beyer, L., Bradbury, J., Agrawal, N., Poole, B., Mordatch, I., Roberts, A., and Sohl-Dickstein, J. N. (2022). Velo: Training versatile learned optimizers by scaling up. In *arXiv*.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, C. D., and Sohl-Dickstein, J. N. (2018). Understanding and correcting pathologies in the training of learned optimizers. In *ICML*.

Micaelli, P. and Storkey, A. (2019). Zero-shot knowledge transfer via adversarial belief matching. In *NeurIPS*.

Micaelli, P. and Storkey, A. (2020). Non-greedy gradient-based hyperparameter optimization over long horizons. In *arXiv*.

Min, J., Kang, D., and Cho, M. (2021). Hypercorrelation squeeze for few-shot segmentation. In *ICCV*.

Minderer, M., Djolonga, J., Romijnders, R., Hubis, F., Zhai, X., Houlsby, N., Tran, D., and Lucic, M. (2021). Revisiting the calibration of modern neural networks. In *NeurIPS*.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *ICLR*.

Mohr, F. and van Rijn, J. N. (2022). Learning curves for decision making in supervised machine learning - a survey. In *arXiv*.

Morcos, A., Raghu, M., and Bengio, S. (2018). Insights on representational similarity in neural networks with canonical correlation. In *NeurIPS*.

Mosser, L. and Naeini, E. Z. (2022). A comprehensive study of calibration and uncertainty quantification for bayesian convolutional neural networks — an application to seismic data. *Geophysics*.

Mu, J., Qiu, W., Hager, G. D., and Yuille, A. L. (2020). Learning from synthetic animals. In *CVPR*.

Muandet, K., Balduzzi, D., and Schölkopf, B. (2013). Domain generalization via invariant feature representation. In *ICML*.

Mukhoti, J., Kulharia, V., Sanyal, A., Golodetz, S., Torr, P. H. S., and Dokania, P. K. (2020). Calibrating deep neural networks using focal loss. In *NeurIPS*.

Müller, R., Kornblith, S., and Hinton, G. (2019). When does label smoothing help? In *NeurIPS*.

Munir, M. A., Khan, M. H., Khan, S., and Khan, F. S. (2023). Bridging precision and confidence: A train-time loss for calibrating object detection. In *CVPR*.

Musgrave, K., Belongie, S., and Lim, S.-N. (2021). Unsupervised domain adaptation: A reality check. In *arXiv*.

Naeini, P., Cooper, G. F., and Hauskrecht, M. (2015). Obtaining well calibrated probabilities using bayesian binning. In *AAAI*.

Naik, D. and Mammone, R. (1992). Meta-neural networks that learn by learning. In *IJCNN*.

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. In *arXiv*.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*.

Oh, C., Kim, M., Lim, H., Park, J., Jeong, E., Cheng, Z.-Q., and Song, K. (2023). Towards calibrated robust fine-tuning of vision-language models. In *NeurIPS 2023 Workshop on Distribution Shifts*.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. In *NeurIPS*.

Ozaki, Y., Tanigaki, Y., Watanabe, S., and Onishi, M. (2020). Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Genetic and Evolutionary Computation Conference*.

Pan, X., Zhang, B., May, J., Nothman, J., Knight, K., and Ji, H. (2017). Cross-lingual name tagging and linking for 282 languages. In *ACL*.

Park, E. and Oliva, J. B. (2019). Meta-Curvature. In *NeurIPS*.

Park, H., Noh, J., Oh, Y., Baek, D., and Ham, B. (2023). Acls: Adaptive and conditional label smoothing for network calibration. In *ICCV*.

Park, S., Simeone, O., and Kang, J. (2020). Meta-learning to communicate: Fast end-to-end training for fading channels. In *ICASSP*.

Patacchiola, M., Turner, J., Crowley, E. J., O'Boyle, M., and Storkey, A. J. (2020). Bayesian meta-learning for the few-shot setting via deep kernels. In *NeurIPS*.

Patra, R., Hebbalaguppe, R., Dash, T., Shroff, G., and Vig, L. (2023). Calibrating deep neural networks using explicit regularisation and dynamic data pruning. In *WACV*.

Piosenka, G. 100 sports image classification.

Piosenka, G. Birds 400 - species image classification.

Popordanoska, T., Tiulpin, A., and Blaschko, M. B. (2024). Beyond classification: Definition and density-based estimation of calibration in object detection. In *WACV*.

Prakash, U., Chollera, A., Khatwani, K., Bodas, T., et al. (2023). Practical first-order bayesian optimization algorithms. In *arXiv*.

Pratt, S. T. . L. (1997). Learning to learn: Introduction and overview. In *Learning to Learn*.

Qin, T., Liu, T.-Y., and Li, H. (2010). A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *ICML*.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. (2020). Rapid learning or feature reuse? Towards understanding the effectiveness of maml. In *ICLR*.

Raghu, M., Gilmer, J., Yosinski, J., and Sohl-Dickstein, J. (2017). Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *NeurIPS*.

Rajeswaran, A., Finn, C., Kakade, S., and Levine, S. (2019). Meta-learning with implicit gradients. In *NeurIPS*.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for machine learning*. The MIT Press.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *ICLR*.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *AAAI*.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., and de Freitas, N. (2022). A generalist agent.

Ren, J., Feng, X., Liu, B., Pan, X., Fu, Y., Mai, L., and Yang, Y. (2023). Torchopt: An efficient library for differentiable optimization. *Journal of Machine Learning Research*.

Ren, M., Triantafillou, E., Sachin Ravi, J. S., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. (2018a). Meta-learning for semi-supervised few-shot classification. In *ICLR*.

Ren, M., Zeng, W., Yang, B., and Urtasun, R. (2018b). Learning to reweight examples for robust deep learning. In *ICML*.

Ruder, S. (2017). An overview of multi-task learning in deep neural networks. In *arXiv*.

Ruhkopf, T., Mohan, A., Deng, D., Tornede, A., Hutter, F., and Lindauer, M. (2023). MASIF: Meta-learned algorithm selection using implicit fidelity information. *Transactions on Machine Learning Research*.

Sachdeva, N. and McAuley, J. (2023). Data distillation: A survey. In *arXiv*.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. In *arXiv*.

Salinas, D., Golebiowski, J., Klein, A., Seeger, M., and Archambeau, C. (2023). Optimizing hyperparameters with conformal quantile regression. In *ICML*.

Salinas, D., Seeger, M., Klein, A., Perrone, V., Wistuba, M., and Archambeau, C. (2022). Syne Tune: A library for large scale hyperparameter tuning and reproducible research. In *First Conference on Automated Machine Learning (Main Track)*.

Schmidhuber, J. (1987). *Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-meta...-hook*. Diploma thesis, Technische Universität München.

Schrodi, S., Stoll, D., Ru, B., Sukthanker, R. S., Brox, T., and Hutter, F. (2023). Construction of hierarchical neural architecture search spaces based on context-free grammars. In *NeurIPS*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. In *arXiv*.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2019). Green AI. In *arXiv*.

Segel, S., Graf, H., Tornede, A., Bischl, B., and Lindauer, M. (2023). Symbolic explanations for hyperparameter optimization. In *AutoML Conference*.

Seo, S., Chang, Y., and Kwak, N. (2023). Flipnerf: Flipped reflection rays for few-shot novel view synthesis. In *ICCV*.

Shaban, A., Bansal, S., Liu, Z., Essa, I., and Boots, B. (2017). One-shot learning for semantic segmentation. In *BMVC*.

Shaban, A., Cheng, C.-A., Hatch, N., and Boots, B. (2019). Truncated back-propagation for bilevel optimization. In *AISTATS*.

Shan, S., Li, Y., and Oliva, J. B. (2020). Meta-Neighborhoods. In *NeurIPS*.

Sharir, O., Peleg, B., and Shoham, Y. (2020). The cost of training nlp models: A concise overview. In *arXiv*.

Shim, J.-h., Kong, K., and Kang, S.-J. (2021). Core-set sampling for efficient neural architecture search. In *ICML Workshop on Subset Selection in ML*.

Shu, J., Xie, Q., Yi, L., Zhao, Q., Zhou, S., Xu, Z., and Meng, D. (2019). Meta-Weight-Net: learning an explicit mapping for sample weighting. In *NeurIPS*.

Siems, J. N., Zimmer, L., Zela, A., Lukasik, J., Keuper, M., and Hutter, F. (2020). NAS-Bench-301 and the case for surrogate benchmarks for neural architecture search. In *NeurIPS Workshop on Meta-Learning*.

Sigaud, O. and Stulp, F. (2019). Policy search in continuous action domains: an overview. *Neural Networks*.

Singh, R., Bharti, V., Purohit, V., Kumar, A., Singh, A. K., and Singh, S. K. (2021). Metamed: Few-shot medical image classification using gradient-based meta-learning. *Pattern Recognition*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *NeurIPS*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *NeurIPS*.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M. M. A., Prabhat, and Adams, R. P. (2015). Scalable bayesian optimization using deep neural networks. In *ICML*.

Song, X., Gao, W., Yang, Y., Choromanski, K., Pacchiano, A., and Tang, Y. (2020). ES-MAML: Simple hessian-free meta learning. In *ICLR*.

Song, Y., Wang, T., Cai, P., Mondal, S. K., and Sahoo, J. P. (2023). A comprehensive survey of few-shot learning: Evolution, applications, challenges, and opportunities. *ACM Computing Surveys*.

Stadie, B. C., Yang, G., Houthooft, R., Chen, X., Duan, Y., Wu, Y., Abbeel, P., and Sutskever, I. (2018). Some considerations on learning to explore via meta-reinforcement learning. In *NeurIPS*.

Stanley, K. O., Clune, J., Lehman, J., and Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*.

Sun, H., Tu, W.-W., and Guyon, I. M. (2021). Omniprint: A configurable printed character synthesizer. In *NeurIPS Datasets and Benchmarks Track*.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to compare: relation network for few-shot learning. In *CVPR*.

Szegedy, C., Vanhoucke, V., Ioffe, S., and Shlens, J. (2016). Rethinking the Inception architecture for computer vision. In *CVPR*.

Tang, W., YANG, B., Li, X., Liu, Y.-H., Heng, P.-A., and Fu, C.-W. (2023). Prototypical variational autoencoder for 3d few-shot object detection. In *NeurIPS*.

Tao, L., Dong, M., Liu, D., Sun, C., and Xu, C. (2023a). Calibrating a deep neural network with its predecessors. In *arXiv*.

Tao, L., Dong, M., and Xu, C. (2023b). Dual focal loss for calibration. In *ICML*.

Thrun, S. and Pratt, L., editors (1998). *Learning to Learn*. Kluwer Academic Publishers.

Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J. B., and Isola, P. (2020). Rethinking few-shot image classification: a good embedding is all you need? In *ECCV*.

Tomani, C., Gruber, S., Erdem, M. E., Cremers, D., and Buettner, F. (2021). Post-hoc uncertainty calibration for domain drift scenarios. In *CVPR*.

Tornede, A., Deng, D., Eimer, T., Giovanelli, J., Mohan, A., Ruhkopf, T., Segel, S., Theodorakopoulos, D., Tornede, T., Wachsmuth, H., et al. (2023). Automl in the age of large language models: Current challenges, future opportunities and risks. In *arXiv*.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. In *arXiv*.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., and Larochelle, H. (2020). Meta-dataset: A dataset of datasets for learning to learn from few examples. In *ICLR*.

Tseng, H.-Y., Lee, H.-Y., Huang, J.-B., and Yang, M.-H. (2020). Cross-domain few-shot classification via learned feature-wise transformation. In *ICLR*.

Tseng, Y.-Y., Bell, A., and Gurari, D. (2022). Vizwiz-fewshot: Locating objects in images taken by people with visual impairments. In *ECCV*.

Tsoumplekas, G., Li, V., Argyriou, V., Lytos, A., Fountoukidis, E., Goudos, S. K., Moscholios, I. D., and Sarigiannidis, P. (2024). Toward green and human-like artificial intelligence: A complete survey on contemporary few-shot learning approaches. In *arXiv*.

Ullah, I., Carrion, D., Escalera, S., Guyon, I., Huisman, M., Mohr, F., van Rijn, J. N., Sun, H., Vanschoren, J., and Vu, P. A. (2022). Meta-album: Multi-domain meta-dataset for few-shot image classification. In *NeurIPS (Datasets and Benchmarks Track)*.

Vaicenavicius, J., Widmann, D., Andersson, C., Lindsten, F., Roll, J., and Schön, T. (2019). Evaluating model calibration in classification. In *AISTATS*.

Viering, T. and Loog, M. (2021). The shape of learning curves: a review. In *arXiv*.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *NIPS*.

Visalpara, S., Killamsetty, K., and Iyer, R. (2021). A data subset selection framework for efficient hyper-parameter tuning and automatic machine learning. In *ICML Workshop on Subset Selection in ML*.

Von Oswald, J., Niklasson, E., Randazzo, E., Sacramento, J., Mordvintsev, A., Zhmoginov, A., and Vladymyrov, M. (2023). Transformers learn in-context by gradient descent. In *ICML*.

Vuorio, R., Sun, S.-H., Hu, H., and Lim, J. J. (2019). Multimodal model-agnostic meta-learning via task-aware modulation. In *NeurIPS*.

Wang, C. (2023). Calibration in deep learning: A survey of the state-of-the-art. In *arXiv*.

Wang, C., Balazs, J., Szarvas, G., Ernst, P., Poddar, L., and Danchenko, P. (2022a). Calibrating imbalanced classifiers with focal loss: An empirical study. In *EMNLP Industry Track*.

Wang, C. and Golebiowski, J. (2023). Meta-calibration regularized neural networks. In *arXiv*.

Wang, D.-B., Feng, L., and Zhang, M.-L. (2021a). Rethinking calibration of deep neural networks: do not be afraid of overconfidence. In *NeurIPS*.

Wang, J., Zhu, Y., Wang, Z., Zheng, Y., Hao, J., and Chen, C. (2023a). Bierl: A meta evolutionary reinforcement learning framework via bilevel optimization. In *arXiv*.

Wang, J. X. (2021). Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*. Computational cognitive neuroscience.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D., and Botvinick, M. (2018a). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*.

Wang, K., Liew, J. H., Zou, Y., Zhou, D., and Feng, J. (2019a). Panet: Few-shot image semantic segmentation with prototype alignment. In *ICCV*.

Wang, M., Gong, Q., Wan, Q., Leng, Z., Xu, Y., Yan, B., Zhang, H., Huang, H., and Sun, S. (2024). A fast interpretable adaptive meta-learning enhanced deep learning framework for diagnosis of diabetic retinopathy. *Expert Systems with Applications*.

Wang, Q., Ye, J., Liu, F., Dai, Q., Kalander, M., Liu, T., Hao, J., and Han, B. (2023b). Out-of-distribution detection with implicit outlier transformation. In *ICLR*.

Wang, T., Zhu, J.-Y., Torralba, A., and Efros, A. A. (2018b). Dataset distillation. In *arXiv*.

Wang, Y., Chao, W.-L., Weinberger, K. Q., and van der Maaten, L. (2019b). Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. In *arXiv*.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020). Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys*.

Wang, Z., Dahl, G. E., Swersky, K., Lee, C., Mariet, Z., Nado, Z., Gilmer, J., Snoek, J., and Ghahramani, Z. (2021b). Pre-trained Gaussian processes for Bayesian Optimization. In *arXiv*.

Wang, Z., Wang, X., Shen, L., Suo, Q., Song, K., Yu, D., Shen, Y., and Gao, M. (2022b). Meta-learning without data via wasserstein distributionally-robust model fusion. In *UAI*.

Watanabe, S., Awad, N., Onishi, M., and Hutter, F. (2023). Speeding up multi-objective hyperparameter optimization by task similarity-based meta-learning for the tree-structured parzen estimator. In *IJCAI*.

Webber, W., Moffat, A., and Zobel, J. (2010). A similarity measure for indefinite rankings. *ACM Transactions on Information Systems*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. (2010). Caltech-UCSD Birds 200. Technical report, California Institute of Technology.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. N. (2017). Learned optimizers that scale and generalize. In *ICML*.

Widmann, D., Lindsten, F., and Zachariah, D. (2019). Calibration tests in multi-class classification: A unifying framework. In *NeurIPS*.

Williams, R. J. (2004). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Wiseman, Y. (2022). Autonomous vehicles. In *Research Anthology on Cross-Disciplinary Designs and Applications of Automation*.

Wistuba, M., Ferianc, M., Balles, L., Archambeau, C., and Zappella, G. (2023). Renate: A library for real-world continual learning. In *CVPR Workshop on Continual Learning in Computer Vision*.

Wu, M., Goodman, N. D., Piech, C., and Finn, C. (2021). ProtoTransformer: A meta-learning approach to providing student feedback. In *arXiv*.

Wu, Y., Ren, M., Liao, R., and Grosse, R. B. (2018). Understanding short-horizon bias in stochastic meta-optimization. In *ICLR*.

Xia, H., Li, K., Min, M. R., and Ding, Z. (2023). Few-shot video classification via representation fusion and promotion learning. In *ICCV*.

Xia, M., Zheng, G., Mukherjee, S., Shokouhi, M., Neubig, G., and Awadallah, A. H. (2021). MetaXL: meta representation transformation for low-resource cross-lingual learning. In *NAACL*.

Xiao, T., Wang, X., Efros, A. A., and Darrell, T. (2021). What should not be contrastive in contrastive learning. In *ICLR*.

Xiao, Y. and Marlet, R. (2020). Few-shot object detection and viewpoint estimation for objects in the wild. In *ECCV*.

Xu, L., Jin, S., Zeng, W., Liu, W., Qian, C., Ouyang, W., Luo, P., and Wang, X. (2022). Pose for everything: Towards category-agnostic pose estimation. In *ECCV*.

Xu, M., Yin, W., Cai, D., Yi, R., Xu, D., Wang, Q., Wu, B., Zhao, Y., Yang, C., Wang, S., et al. (2024). A survey of resource-efficient llm and multimodal foundation models. In *arXiv*.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. (2020). PC-DARTS: Partial channel connections for memory-efficient architecture search. In *ICLR*.

Yang, A., Lu, C., Li, J., Huang, X., Ji, T., Li, X., and Sheng, Y. (2023). Application of meta-learning in cyberspace security: a survey. *Digital Communications and Networks*.

Yang, P., Ren, S., Zhao, Y., and Li, P. (2022). Calibrating CNNs for few-shot meta learning. In *WACV*.

Yang, Y. and Hospedales, T. (2015). A unified perspective on multi-domain and multi-task learning. In *ICLR*.

Yang, Y. and Hospedales, T. M. (2023). Partial index tracking: A meta-learning approach. In *CoLLAs*.

Yang, Y., Morillo, I. G., and Hospedales, T. M. (2018). Deep neural decision trees. In *ICML Workshop on Human Interpretability in Machine Learning*.

Yao, Q., Wang, M., Escalante, H. J., Guyon, I. M., Hu, Y.-Q., Li, Y.-F., Tu, W.-W., Yang, Q., and Yu, Y. (2018). Taking human out of learning applications: A survey on automated machine learning. In *arXiv*.

Ye, H.-J., Hu, H., Zhan, D.-C., and Sha, F. (2020). Few-shot learning via embedding adaptation with set-to-set functions. In *CVPR*.

Yin, M., Tucker, G., Zhou, M., Levine, S., and Finn, C. (2020). Meta-learning without memorization. In *ICLR*.

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., and Hutter, F. (2019). Nas-bench-101: Towards reproducible neural architecture search. In *ICML*.

Yu, R., Liu, S., and Wang, X. (2023). Dataset distillation: A comprehensive review. In *arXiv*.

Yu, T., Kumar, S., Gupta, A., Levine, S., Hausman, K., and Finn, C. (2020). Gradient surgery for multi-task learning. In *NeurIPS*.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *CORL*.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In *BMVC*.

Zamir, A. R., Sax, A., Shen, W., Guibas, L. J., Malik, J., and Savarese, S. (2018). Taskonomy: Disentangling task transfer learning. In *CVPR*.

Zappella, G., Salinas, D., and Archambeau, C. (2021). A resource-efficient method for repeated HPO and NAS problems. In *ICML AutoML Workshop*.

Zela, A., Elsken, T., Saikia, T., Marrakchi, Y., Brox, T., and Hutter, F. (2020). Understanding and robustifying differentiable architecture search. In *ICLR*.

Zhai, X., Puigcerver, J., Kolesnikov, A., Ruyssen, P., Riquelme, C., Lucic, M., Djolonga, J., Pinto, A. S., Neumann, M., Dosovitskiy, A., et al. (2019). A large-scale study of representation learning with the visual task adaptation benchmark. In *arXiv*.

Zhang, M., Marklund, H., Dhawan, N., Gupta, A., Levine, S., and Finn, C. (2021). Adaptive risk minimization: learning to adapt to domain shift. In *NeurIPS*.

Zhang, X. S., Tang, F., Dodge, H. H., Zhou, J., and Wang, F. (2019). Metapred: Meta-learning for clinical risk prediction with limited patient electronic health records. In *SIGKDD*.

Zhao, B., Mopuri, K. R., and Bilen, H. (2021). Dataset Condensation with Gradient Matching. In *ICLR*.

Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017). Pyramid scene parsing network. In *CVPR*.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023a). A survey of large language models. In *arXiv*.

Zhao, X., Gattoni, R., Kozlosky, A., Jacobs, A., Morrow, C., Lindo, S., and Spruston, N. (2023b). Meta-learning in head fixed mice navigating in virtual reality: A behavioral analysis. In *bioRxiv*.

Zheng, K., Zhang, H., and Huang, W. (2023). Diffkendall: A novel approach for few-shot learning with differentiable kendall's rank correlation. In *NeurIPS*.

Zheng, L., Tang, M., Chen, Y., Wang, J., and Lu, H. (2020a). Learning feature embeddings for discriminant model based tracking. In *ECCV*.

Zheng, W., Yan, L., Gou, C., and Wang, F.-Y. (2020b). Federated meta-learning for fraudulent credit card detection. In *IJCAI*.

Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., and Torralba, A. (2018). Places: a 10 million image database for scene recognition. *Transactions on Pattern Analysis and Machine Intelligence*.

Zhou, D., Zhou, X., Zhang, W., Loy, C. C., Yi, S., Zhang, X., and Ouyang, W. (2020a). EcoNAS: Finding proxies for economical neural architecture search. In *CVPR*.

Zhou, P., Yuan, X.-T., Xu, H., Yan, S., and Feng, J. (2019). Efficient meta learning via minibatch proximal update. In *NeurIPS*.

Zhou, W., Li, Y., Yang, Y., Wang, H., and Hospedales, T. (2020b). Online meta-critic learning for off-policy actor-critic methods. In *NeurIPS*.

Zieliński, B., Plichta, A., Misztal, K., Spurek, P., Brzychczy-Włoch, M., and Ochońska, D. (2017). Deep learning approach to bacterial colony classification. *PLOS ONE*.

Zimmer, L., Lindauer, M., and Hutter, F. (2021). Auto-PyTorch Tabular: Multi-fidelity metalearning for efficient and robust AutoDL. *Transactions on Pattern Analysis and Machine Intelligence*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *ICLR*.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *CVPR*.