A preliminary study on the feature representations of transfer learning and gradient-based meta-learning techniques

Huisman, M.; Rijn, J.N. van; Plaat, A.

A Preliminary Study on the Feature Representations of Transfer Learning and Gradient-Based Meta-Learning Techniques

Mike Huisman
LIACS, Leiden University
Niels Bohrweg 1, 2333CA, The Netherlands
m.huisman@liacs.leidenuniv.nl

Aske Plaat
LIACS, Leiden University
Niels Bohrweg 1, 2333CA, The Netherlands
a.plaat@liacs.leidenuniv.nl

Jan N. van Rijn
LIACS, Leiden University
Niels Bohrweg 1, 2333CA, The Netherlands
j.n.van.rijn@liacs.leidenuniv.nl

Abstract

Meta-learning receives considerable attention as an approach to enable deep neural networks to learn from a few data. Recent studies suggest that in specific cases, simply fine-tuning a pre-trained network may be more effective at learning new image classification tasks from limited data than more sophisticated meta-learning techniques such as MAML. This is surprising as the learning behaviour of MAML mimics that of fine-tuning. We investigate this phenomenon and show that the pre-trained features are more diverse and discriminative than those learned by MAML and Reptile, which specialize in adaptation in low-data regimes of similar data distributions as the one used for training. Due to this specialization, MAML and Reptile may be hampered in their ability to generalize to out-of-distribution tasks, whereas fine-tuning can fall back on the diversity of the learned features.

1 Introduction

Deep learning techniques have enabled breakthroughs in various areas such as game-playing (Silver et al., 2016; Mnih et al., 2015), image recognition (Krizhevsky et al., 2012; He et al., 2015), and machine translation (Wu et al., 2016). However, deep neural networks are notoriously *data-hungry* (LeCun et al., 2015), limiting their successes to domains where sufficient data and computing resources are available (Hospedales et al., 2021; Huisman et al., 2021b). *Meta-learning* (Schaul and Schmidhuber, 2010; Schmidhuber, 1987; Thrun, 1998) is one approach to reduce these limitations by learning how to learn from a few data.

While the field has been attracting much attention, recent results (Chen et al., 2019; Tian et al., 2020; Mangla et al., 2020) suggest that simply pre-training a network on a large data set and *fine-tuning* only the head of the network may be more effective at learning new image classification tasks quickly than more complicated meta-learning techniques such as MAML (Finn et al., 2017). This phenomenon is not well understood and surprising as Raghu et al. (2020) have shown that the adaptation behaviour of MAML resembles that of fine-tuning when learning new image classification tasks: most of the changes take place in the head of the network while the body of the network is mostly kept frozen.

In this work, we aim to find an explanation for the fact that simple fine-tuning can outperform gradient-based meta-learning techniques such as MAML (Finn et al., 2017) and Reptile (Nichol et al.,

5th Workshop on Meta-Learning at NeurIPS 2021, held virtually.

2018). For this, we analyse the learning objectives of the three techniques (fine-tuning, MAML, Reptile) and show that they can be interpreted as maximizing initial performance, post-adaptation performance (after a few training updates on a new task), and a combination of the two, respectively. Based on this interpretation, we hypothesize that MAML and Reptile settle for inferior initial features compared with fine-tuning due to their relative negligence of the initial performance.

The primary contribution of our work is that we show that the pre-trained features of the fine-tuning technique are more diverse and discriminative than those learned by MAML and Reptile.[1] We show this using experiments on the miniImageNet and CUB benchmarks. Nonetheless, MAML and Reptile can outperform fine-tuning when the backbone is shallow or the training and test data distributions are similar (Chen et al., 2019), indicating that they have specialized for adaptation in low-data regimes of the training distribution. However, when the test distribution diverges from the training distribution, the advantage of having a broad and diverse feature space may outweigh the quick adaptation specialism of the meta-learning techniques, allowing simple fine-tuning to bridge the performance gap to MAML and Reptile, or even outperform them (Tian et al., 2020; Mangla et al., 2020).

2 Related work

Meta-learning is a popular approach to enable deep neural networks to learn from a few data. Many architectures and model types have been proposed, such as MAML (Finn et al., 2017), the meta-learner LSTM (Ravi and Larochelle, 2017), TURTLE (Huisman et al., 2021a) and MetaOptNet (Lee et al., 2019). However, our understanding of newly proposed techniques remains limited in some cases. For example, different techniques use different backbones which raises the question of whether performance differences between techniques are due to new model-types or due to the difference in used backbones (Huisman et al., 2021b).

Chen et al. (2019) was one of the first that investigated this question by performing a fair comparison between popular meta-learning techniques, including MAML (Finn et al., 2017), on few-shot image classification benchmarks such as miniImageNet (Vinyals et al., 2016; Ravi and Larochelle, 2017) and CUB (Wah et al., 2011). Their results show that MAML often outperforms fine-tuning when the test tasks come from a similar data distribution as the training distribution when using shallow backbones. When the backbone becomes deeper and/or the domain differences between training and test tasks increase, however, this performance gap is reduced and, in some cases, fine-tuning outperforms MAML.

In addition to these findings by Chen et al. (2019), Tian et al. (2020) demonstrate that simply fine-tuning a pre-trained feature embedding module yields better performance than popular meta-learning techniques (including MAML) on few-shot benchmarks. Mangla et al. (2020) and Yang et al. (2021) further support this finding as they have proposed new few-shot learning techniques based on pre-training and fine-tuning which significantly outperform meta-learning techniques.

These performance differences between simple fine-tuning and more sophisticated techniques such as MAML may be surprising, as Raghu et al. (2020) found that the learning behaviour of MAML is similar to that of fine-tuning on image classification benchmarks. More specifically, they compared the feature representations of MAML before and after task-specific adaptation, and show that MAML relies mostly on feature re-use instead of quick adaptation because the body of the network is barely adjusted, which resembles the learning dynamics of fine-tuning (see Section 3.2). Collins et al. (2020) compared the feature representations of MAML and the finetuning method (expected risk minimization) in linear regression settings and found that MAML finds an initialization closer to the hard tasks, characterized by their gentle loss landscapes with small gradients. We demonstrate a similar property: MAML has greater flexibility in picking an initialization as long as the post-adaptation performance is good.

In this work, we aim to unite the findings of Raghu et al. (2020) and Chen et al. (2019) by finding an answer to the question of why fine-tuning can outperform meta-learning techniques such as MAML and Reptile (Nichol et al., 2018) in some image classification scenarios while it is outperformed in other scenarios (when using a shallow backbone or when train/test task distributions are similar).

[1] All code for reproducing our results can be found at `https://github.com/mikehuisman/transfer-meta-feature-representations`

3 Background

In this section, we explain fine-tuning, MAML, and Reptile in the context of supervised learning.

3.1 Supervised learning

In the *supervised learning* setting, we have a joint probability distribution over inputs $\mathbf{x}$ and corresponding outputs $\mathbf{y}$, i.e., $p(\mathbf{x}, \mathbf{y})$. In the context of deep learning, the goal is to build deep neural networks that can predict for any given input $\mathbf{x}$ the correct output $\mathbf{y}$. Throughout this paper, we assume that the neural network architecture f is fixed and that we only wish to find a set of parameters θ such that the network predictions $f_\theta(\mathbf{x})$ are as good as possible. This can be done by updating the parameters θ to minimize a loss function $\mathcal{L}$ that captures how well the network is performing. Thus, under the joint distribution $p(\mathbf{x}, \mathbf{y})$, we wish to find

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{\mathbf{x}_i, \mathbf{y}_i} \left[\mathcal{L}_{\mathbf{x}_i, \mathbf{y}_i}(\theta) \right], \tag{1}$$

where $\mathbf{x}_i, \mathbf{y}_i \sim p(\mathbf{x}, \mathbf{y})$.

The most common way to approximate these parameters is by performing gradient descent on that loss function, which means that we update the parameters in the direction of the steepest descent

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla_{\theta^{(t)}} \mathop{\mathbb{E}}_{\mathbf{x}_i, \mathbf{y}_i} \left[\mathcal{L}_{\mathbf{x}_i, \mathbf{y}_i}(\theta^{(t)}) \right]. \tag{2}$$

Here, $\nabla_{\theta^{(t)}}$ is the gradient with respect to $\theta^{(t)}$, t indicates the time step, and α the learning rate or step size.

3.2 Fine-tuning

Minimizing the objective in Equation 1 using gradient-based optimization often requires large amounts of data. This raises the question of how we can learn tasks for which only a few data are available. The transfer learning technique called *fine-tuning* tackles this problem as follows. In the *pre-training phase*, it minimizes Equation 1 on a given source distribution $p_s(\mathbf{x}, \mathbf{y})$ using gradient descent as shown in Equation 2. This leads to a sequence of updates that directly update the initialization parameters. Then, it freezes the feature extraction module of the network: all parameters of the network through the penultimate layer, i.e., $\theta^{(1:L-1)}$ where L is the number of layers. When presented with a target distribution $p_t(\mathbf{x}, \mathbf{y})$ from which we can sample fewer data, we can simply re-use the learned feature embedding module $f_{\theta^{(1:L-1)}}$ (all hidden layers of the network excluding the output layer) for this new problem. Then, in the *fine-tuning phase*, it only trains the parameters in the head of the network $\theta^{(L)}$ (the final layer).

By reducing the number of trainable parameters on the target problem, this technique effectively reduces the model complexity and prevents overfitting issues associated with the data scarcity in few-shot learning scenarios. This comes at the cost of not being able to adjust the feature representations of inputs. As a consequence, this approach fails when the pre-trained embedding module fails to produce informative representations of the target problem inputs.

3.3 Reptile

Instead of joint optimization on the source distribution, *Reptile* (Nichol et al., 2018) adheres to the idea of meta-learning and thus aims to learn how to learn. For this, it splits the source distribution $p_s(\mathbf{x}, \mathbf{y})$ into a number of smaller task distributions $p_1(\mathbf{x}, \mathbf{y}), p_2(\mathbf{x}, \mathbf{y}), \ldots, p_n(\mathbf{x}, \mathbf{y})$, corresponding to tasks $\mathcal{T}_1, \mathcal{T}_2, \ldots \mathcal{T}_n$. On a single task $\mathcal{T}_j$ for $j \in \{1, \ldots, n\}$, its objective is to minimize Equation 1 under the task distribution $p_j(\mathbf{x}, \mathbf{y})$ using T gradient descent update steps as shown in Equation 2. This results in a sequence of weight updates $\theta \to \theta_j^{(1)} \to \ldots \to \theta_j^{(T)}$. After task-specific adaptation, the initial parameters θ are moved into the direction of $\theta_j^{(T)}$

$$\theta = \theta + \epsilon \left(\theta_j^{(T)} - \theta \right), \tag{3}$$

where ϵ is the step size. Intuitively, this update interpolates between the current initialization parameters θ and the task-specific parameters $\theta_j^{(T)}$. The updated initialization θ is then used as

starting point when presented with new tasks, and the same process is repeated. It is easy to show that this update procedure corresponds to performing first-order optimization of the multi-step objective

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{\mathcal{T}_j \sim p(\mathcal{T})} \left(\sum_{t=0}^{T-1} \mathop{\mathbb{E}}_{\mathbf{x}_i, \mathbf{y}_i \sim p_j} \left[\mathcal{L}_{t+1}(\theta_{\mathbf{j}}^{(\mathbf{t})}) \right] \right), \tag{4}$$

where $\mathcal{L}_{t+1}$ is shorthand for the loss on a mini-batch sampled at time step t.

3.4 MAML

Another popular gradient-based meta-learning technique is *MAML* (Finn et al., 2017). Just like Reptile, MAML also learns a weight initialization θ from which new tasks can be learned more efficiently. However, instead of optimizing a multi-step loss function, MAML only optimizes the final performance after task-specific adaptation. More specifically, this means that MAML is only interested in the performance of the final weights $\theta_j^{(T)}$ on a task and not in intermediate performances of weights $\theta_j^{(t)}$ for $t < T$. In other words, MAML aims to find

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{\mathcal{T}_j \sim p(\mathcal{T})} \left(\mathop{\mathbb{E}}_{\mathbf{x}_i, \mathbf{y}_i \sim p_j} \left[\mathcal{L}_T(\theta_{\mathbf{j}}^{(\mathbf{T})}) \right] \right). \tag{5}$$

To find these parameters, MAML updates its initialization parameters as follows

$$\theta = \theta - \beta \nabla_\theta \mathcal{L}_{T+1}(\theta_j^{(T)}), \tag{6}$$

where β is the learning rate and $\nabla_\theta \mathcal{L}_{T+1}(\theta_j^{(T)}) = \nabla_{\theta_j^{(T)}} \mathcal{L}_{T+1}(\theta_j^{(T)}) \nabla_\theta \theta_j^{(T)}$. The factor $\nabla_\theta \theta_j^{(T)}$ contains second-order gradients and can be ignored by assuming that $\nabla_\theta \theta_j^{(T)} = I$ is the identity matrix, in similar fashion to what Reptile does. This assumption gives rise to *first-order* MAML (fo-MAML) and significantly increases the training efficiency in terms of running time and memory usage, whilst achieving roughly the same performance as the *second-order MAML* version (Finn et al., 2017). In short, first-order MAML updates its initialization in the gradient update direction of the final task-specific parameters.

4 A common framework and interpretation

The three discussed techniques can be seen as part of a general gradient-based optimization framework, as shown in Algorithm 1. All algorithms try to find a good set of initial parameters as specified by their objective functions. The parameters are initialized randomly in line 1. Then, these initial parameters are iteratively updated based on the learning objectives (the loop starting from line 2).

This iterative updating procedure continues as follows. First, the data distribution is selected to sample data from (line 3). That is, fine-tuning uses the full joint distribution $p_s(\mathbf{x}, \mathbf{y})$ of the source problem, whereas Reptile and MAML select task distributions $p_j(\mathbf{x}, \mathbf{y})$ (obtained by sub-sampling a set of labels from the full distribution p_s). Next, we make T task-specific updates on mini-batches sampled from the distribution p that was selected in the previous stage (lines 4–8). Lastly, the initial parameters θ are updated using the outcomes of the task-specific adaptation phase.

Note that in this general gradient-based optimization framework, all techniques update their initialization parameters based on a single distribution p at a time. One could also choose to use batches of distributions, or *meta-batches*, to update the initialization θ. This can be incorporated by using the average of the losses of the different distributions as an aggregated loss function.

Table 1 gives an overview of the three algorithms. As we can see, fine-tuning only optimizes for the initial performance and does not take into account the performance after adaptation. This means that its goal is to correctly classify any input $\mathbf{x}$ from the source problem distribution p_s. Reptile, on the other hand, optimizes both for initial performance, as well as the performance after every update step. This means that Reptile may settle for an initialization with somewhat worse initial performance compared with fine-tuning, as long as the performance during task-specific adaptation makes up for this initial deficit. MAML is the most extreme in the sense that it can settle for an initialization with poor initial performance, as long as the final performance is good.

Table 1: Overview of the loss functions and corresponding focus of fine-tuning, Reptile, and MAML.

Algorithm	Loss function	Focus
Fine-tuning	$\mathop{\mathbb{E}}_{\mathbf{x}_i,\mathbf{y}_i}\left[\mathcal{L}_{\mathbf{x}_i,\mathbf{y}_i}(\theta)\right]$	Initial performance
Reptile	$\mathop{\mathbb{E}}_{\mathcal{T}_j\sim p(\mathcal{T})}\left(\sum_{t=0}^{T-1}\mathop{\mathbb{E}}_{\mathbf{x}_i,\mathbf{y}_i\sim p_j}\left[\mathcal{L}_{t+1}(\theta_{\mathbf{j}}^{(\mathbf{t})})\right]\right)$	Initial, intermediate, and final performance
MAML	$\mathop{\mathbb{E}}_{\mathcal{T}_j\sim p(\mathcal{T})}\left(\mathop{\mathbb{E}}_{\mathbf{x}_i,\mathbf{y}_i\sim p_j}\left[\mathcal{L}_T(\theta_{\mathbf{j}}^{(\mathbf{T})})\right]\right)$	Final performance

Algorithm 1 General gradient-based optimization: fine-tuning reptile MAML

1: Randomly initialize θ
2: **while** not converged **do**
3: Select data distribution $p =$ p_s $p_j \sim p(\mathcal{T})$ $p_j \sim p(\mathcal{T})$
4: Set $\theta^{(0)} = \theta$
5: **for** $t = 0, ..., T-1$ **do**
6: Sample a batch of data $\mathbf{x}, \mathbf{y} \sim p$
7: Compute $\theta^{(t+1)} = \theta^{(t)} - \nabla_{\theta^{(t)}}\mathcal{L}_{t+1}(\theta^{(t)})$
8: **end for**
9: Update θ by $\theta = \theta^{(T)}$ Equation 3 Equation 6
10: **end while**

5 Experiments

In this section, we compare the learning behaviours of fine-tuning, MAML, and Reptile. All experiments are conducted using single PNY GeForce RTX 2080TI GPUs.

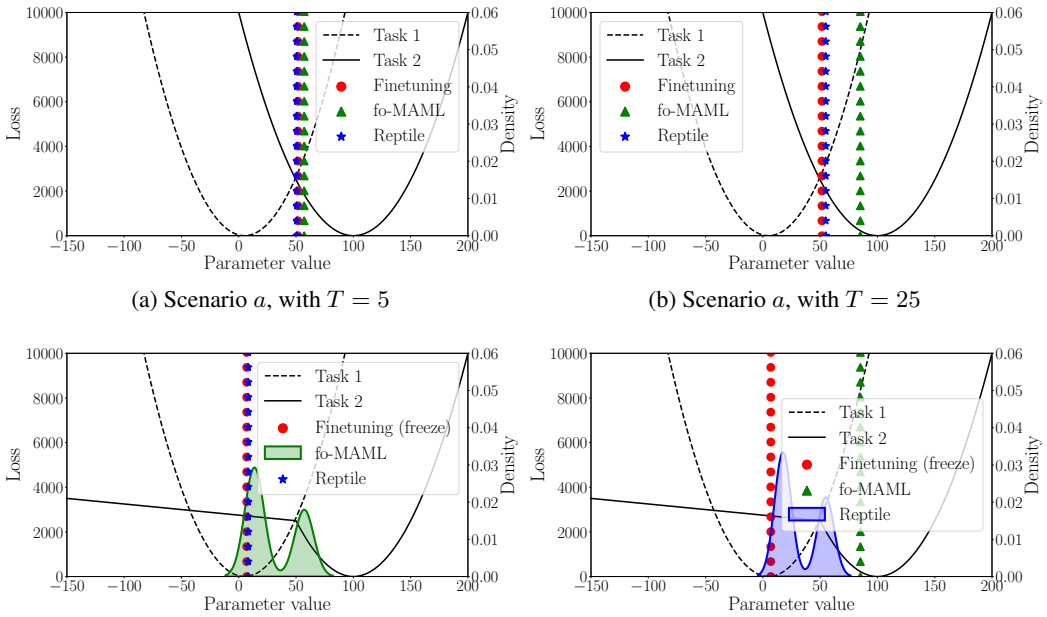
5.1 Toy example

First, we study the behaviour of fine-tuning, Reptile, and MAML in two synthetic scenarios a and b, consisting of two tasks each. Here, these tasks can be considered the meta-train set, and the goal of the algorithms is to find an initialization of a network that performs well on both tasks. We represent tasks by their loss landscape, which we have constructed by hand for illustrative purposes. In scenario a, the two task loss landscapes are quadratic functions of a single parameter x. More specifically, the losses for this scenario are given by $\ell_1^a(x) = 1.3(x-5)^2$ and $\ell_2^a(x) = (x-100)^2$. In scenario b, the first task loss landscape is the same $\ell_1^b = \ell_1^a$ while the second task represents a more complex function:

$$\ell_2^b(x) = \begin{cases} (x-100)^2 & x > 50 \\ -5x + 2750 & x \leq 50 \end{cases} \tag{7}$$

The respective algorithms train by sampling tasks in an interleaved fashion, and by adapting the parameter x based on the loss landscape of the sampled task. We investigate the behaviour of Reptile and MAML when they make $T = 5$ or $T = 25$ task-specific adaptation steps. For this, we average the found solutions of the techniques over 100 different runs with initial x values that are equally spaced in the interval $[-200, +200]$. We find that finetuning converges to the same point regardless of the initialization and are thus represented by a single vertical line. For Reptile and MAML, the found solution depends on the initialization, which is why we represent the found solution as a probability density. A Jupyter notebook for reproducing these results can be found on our Github page.

Based on the learning objectives of the techniques, we expect finetuning to settle for an initialization that has a good initial performance on both tasks (small loss values). Furthermore, we expect that MAML will pick any initialization point from which it can reach minimal loss on both tasks within T steps. Reptile is expected to find a mid-way solution between finetuning and MAML.



(a) Scenario a, with $T = 5$

(b) Scenario a, with $T = 25$

(c) Scenario b, with $T = 5$

(d) Scenario b, with $T = 25$

Figure 1: Average initialization that fine-tuning, Reptile, and MAML converge to when using $T = 5$ or $T = 25$ adaptation steps per task. In scenario a (top figures), fine-tuning and Reptile pick an initialization in the centre of the two optima where the initial loss is minimal. MAML neglects the initial performance and thus is freer to select an initialization point, especially when T is larger. In scenario b (bottom figures) the loss of task 2 is no longer convex and has a reasonably flat plateau. Fine-tuning and Reptile get stuck in the optimum of the first task and fail to learn the second task successfully, while MAML finds a location from which it can arrive at both optima.

The results of these experiments are displayed in Figure 1. In scenario a (top figures), we see that both fine-tuning and Reptile prefer an initialization at the intersection of the two loss curves, where the initial loss is minimal. MAML, on the other hand, neglects the initial performance when $T = 25$ and leans more to the right, whilst ensuring that it can reach the two optima within T steps. The reason that it prefers an initialization on the right of the intersection is that the loss landscape of task 1 is steeper, which means that task adaptation steps will be larger. Thus, a location at the right of the intersection ensures good learning of task 2 and yields comparatively fast learning on the first task.

In scenario b (bottom figures), the loss landscape of task 2 has a relatively flat plateau on the left-hand side. Because of this, fine-tuning and Reptile will be pulled towards the optimum (also the joint optimum) of the first task due to the larger gradients compared with the small gradients of the flat region of the second task. When $T = 25$, we see that Reptile either finds an initialization at $x = 50$ (when the starting point x_0 is on the right-hand side of the plateau) or at the joint optimum at $x = 0$ (when it starts with x_0 on the plateau). In the latter case, the post-adaptation performance of Reptile on both tasks is not optimal because it cannot reach the optimum of task 2. MAML, on the other hand, does not suffer from this suboptimality because it neglects the initial and intermediate performance and simply finds an initialization at $x \approx 85$ from which it can reach both the optima of tasks 1 and 2.

5.2 Information content in the learned initializations

Next, we investigate the relationship between the few-shot image classification performance and the discriminative power of the learned features by the three techniques.

To study this, we use the miniImageNet (Vinyals et al., 2016; Ravi and Larochelle, 2017) and CUB (Wah et al., 2011) benchmarks and three different backbones (Conv-4, ResNet-10, ResNet-18 (He et al., 2015)). We use Torchmeta for the implementation of the data loaders (Deleu et al., 2019). We note that a single run of MAML and finetuning finish within one day, while Reptile finished within 4

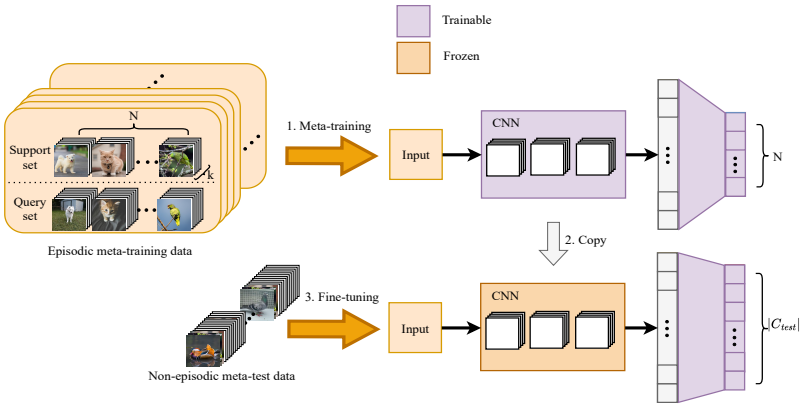


Figure 2: Flow chart for measuring the joint classification accuracy for meta-learning techniques. First, we train the techniques in an episodic manner on all data in the meta-train set. Second, we copy and freeze the learned initialization parameters and replace the output layer with a new one. Third, we fine-tune this new output layer on all meta-test data in a non-episodic manner. As such, the meta-test data is split into a non-episodic train and non-episodic test set. Finally, we evaluate the learned evaluation on the hold-out test split of the meta-test data. We refer to the resulting performance measure as the joint classification accuracy. Note that finetuning follows the same procedure, with the exception that it trains non-episodically (on batches instead of tasks) on the meta-training data.

days, perhaps due to the absence of parallelism in our implementation. We tune the hyperparameters of finetuning and MAML 5-way 1-shot image classification on the meta-validation tasks using random search with a budget of 30 function evaluations for every backbone and dataset. Due to the computational expenses, for Reptile, we use the best-reported hyperparameters on 5-way 1-shot miniImageNet as found by (Nichol et al., 2018).

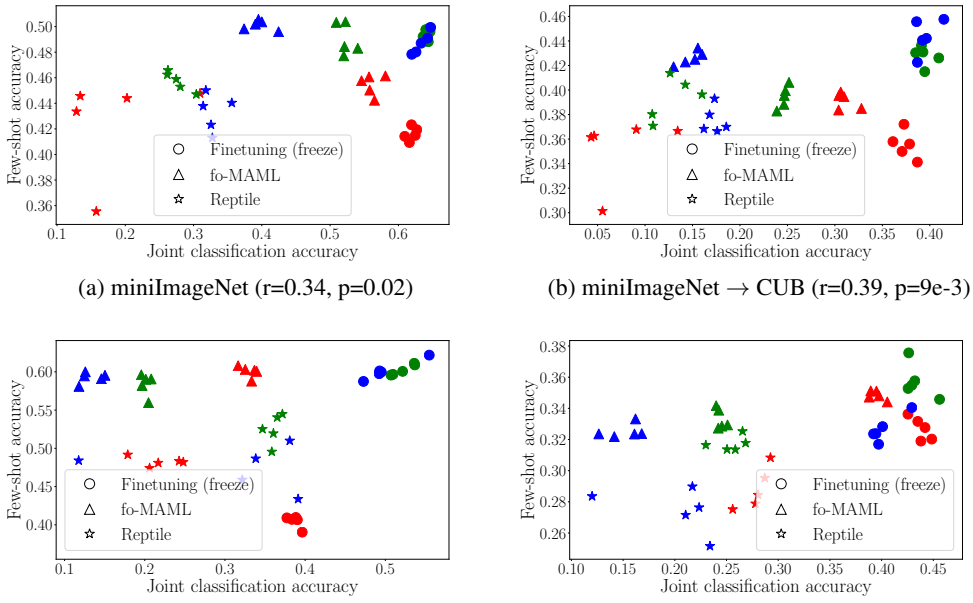
After deploying the three techniques on the data sets in a 5-way 1-shot manner, we measure the discriminative power of the learned initializations. Figure 2 visualizes this procedure for MAML and Reptile; finetuning follows a similar procedure. First, we extract the learned initialization parameters from the techniques. Second, we load these initializations into the base-learner network, freeze all hidden layers, and replace the output layer with a new one. The new output layer contains one node for every of the $|C_{test}|$ classes in the meta-test data. Third, we fine-tune this new output layer on the meta-test data in a *non-episodic* manner, which corresponds to regular supervised learning on the meta-test data set. We use a 60/40 train/test split and evaluate the final performance on the latter. We refer to the resulting performance measure as the *joint classification accuracy*, which aims to indicate the discriminative power of the learned initialization, evaluated on data from unseen classes.

The results of this experiment are shown in Figure 3. From this figure, we see that fine-tuning yields the best joint classification accuracy in all scenarios. This suggests that finetuning has learned a more discriminative feature space for direct joint classification on a large set of classes than both MAML and Reptile. However, we note that the joint classification performance either weakly correlates or does not correlate with the few-shot learning performance across the different techniques. We note that these correlation patterns may be affected by the fact that that we used the best-reported hyperparameters for Reptile for the Conv-4 backbone, while we also use ResNet-10 and ResNet-18 backbones (He et al., 2015) in different settings. For finetuning, however, we do observe strong and significant correlations between the joint classification accuracy and few-shot learning performance (see Appendix A).

Moreover, we see that MAML yields the best few-shot learning performance when using the shallow Conv-4 backbone. As the used backbone becomes deeper, the features learned by MAML become narrower, i.e., the joint classification accuracy decreases.

6 Discussion

We investigated observed performance differences between finetuning and gradient-based meta-learning techniques (MAML and Reptile), both in within-distribution and out-of-distribution settings.



(a) miniImageNet (r=0.34, p=0.02)

(b) miniImageNet → CUB (r=0.39, p=9e-3)

(c) CUB (r=0.07, p=0.63)

(d) CUB → miniImageNet (r=0.53, p=2e-4)

Figure 3: The joint classification accuracy (x-axes) plotted against the 5-way 1-shot performance (y-axis) on all test classes. For every technique, there are 15 results plotted, corresponding to 3 backbones (Conv-4=red, ResNet-10=green, ResNet-18=blue) and 5 runs per setting. The Pearson correlation coefficients (r) and p-values are displayed in the subcaptions. The general correlations between the few-shot learning performance and joint classification accuracy range from weak to mild. The correlations for individual techniques correlations can be found (see Appendix A).

The optimization objectives of the three techniques can be interpreted as maximizing the direct performance, post-adaptation performance, and a combination of the two, respectively. That is, fine-tuning aims to maximize the direct performance whereas MAML aims to maximize the performance *after* a few adaptation steps, making it a look-ahead objective. Reptile is a combination of the two as it focuses on both the initial performance as well as the performance after every update step on a given task. As a result, fine-tuning will favour an initialization that jointly minimizes the loss function, whereas MAML may settle for an inferior initialization that yields more promising results after a few gradient update steps. Reptile picks something in between the two extremes. Our synthetic example in Section 5.1 shows that these interpretations of the learning objectives allow us to understand the chosen initialization parameters.

Our empirical results show that these different objectives translate into different learned initializations. More specifically, we have found that fine-tuning learns a broad and diverse set of features that allows it to discriminate between many different classes. MAML and Reptile, in contrast, optimize a look-ahead objective and settle for a less diverse and broad feature space as long as it facilitates robust adaptation in low-data regimes of the *same* data distribution (as that is used to optimize the look-ahead objective). Whilst for finetuning a larger diversity does often lead to increased few-shot performance, a similar correlation is not found for MAML and Reptile.

Another result is that MAML yields the best few-shot learning performance when using the Conv-4 backbone in all settings. Interestingly, the features learned by MAML become less discriminative as the depth of the backbone increases. This may indicate an over-specialization, and it may be interesting to see whether adding a penalty for narrow features may prevent this and increase the few-shot learning performance on out-of-distribution tasks.

In summary, our work shows that the broadness of features does not correlate with the ability to learn new tasks quickly across techniques, although it is helpful when learning tasks that are outside of the training distribution as finetuning outperforms MAML and Reptile in these scenarios.

Acknowledgments

This work was carried out on i) the Dutch national e-infrastructure with the support of SURF Cooperative and ii) the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

References

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B. (2019). A closer look at few-shot classification. In *International Conference on Learning Representations*, ICLR'19.

Collins, L., Mokhtari, A., and Shakkottai, S. (2020). Why does maml outperform erm? an optimization perspective. *arXiv preprint arXiv:2010.14672*.

Deleu, T., Würfl, T., Samiei, M., Cohen, J. P., and Bengio, Y. (2019). Torchmeta: A Meta-Learning library for PyTorch. Available at: https://github.com/tristandeleu/pytorch-meta.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, page 1126–1135. PMLR.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

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

Huisman, M., Plaat, A., and van Rijn, J. N. (2021a). Stateless neural meta-learning using second-order gradients. *arXiv preprint arXiv:2104.10527*.

Huisman, M., van Rijn, J. N., and Plaat, A. (2021b). A survey of deep meta-learning. *Artificial Intelligence Review*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, NIPS'12, pages 1097–1105.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665.

Mangla, P., Kumari, N., Sinha, A., Singh, M., Krishnamurthy, B., and Balasubramanian, V. N. (2020). Charting the right manifold: Manifold mixup for few-shot learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2218–2227.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nichol, A., Achiam, J., and Schulman, J. (2018). On First-Order Meta-Learning Algorithms. *arXiv preprint arXiv:1803.02999*.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. (2020). Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML. In *International Conference on Learning Representations*, ICLR'20.

Ravi, S. and Larochelle, H. (2017). Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*, ICLR'17.

Schaul, T. and Schmidhuber, J. (2010). Metalearning. *Scholarpedia*, 5(6):4650.

Schmidhuber, J. (1987). Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Master's thesis, Technische Universität München.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Thrun, S. (1998). Lifelong Learning Algorithms. In *Learning to learn*, pages 181–209. Springer.

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? *arXiv preprint arXiv:2003.11539*.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems 29*, NIPS'16, pages 3637–3645.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*.

Yang, S., Liu, L., and Xu, M. (2021). Free lunch for few-shot learning: Distribution calibration. In *International Conference on Learning Representations*, ICLR'21.

A Additional experimental results

Table 2 displays the Pearson correlation and corresponding p-values for individual techniques for the experiment in Section 5.2. As we can see, there are strong and significant ($\alpha = 0.005$) correlations between the joint classification accuracy and the few-shot learning performance of finetuning in three settings. For MAML, there are strong negative correlations on miniImageNet and miniImageNet $\rightarrow$ CUB, indicating that a lower joint classification accuracy is often associated with better few-shot learning performance. For Reptile, the correlations are non-significant and mild to weak.

Table 2: Individual correlations between the joint classification accuracy and the few-shot learning performance. The Pearson correlation coefficients are indicated as *r* and corresponding p-values as *p*. We note that the results for each of the three few-shot learning techniques are produced with three different backbone networks. As such, correlations should be interpreted with utmost care. Significant correlations (using a threshold of $\alpha = 0.005$) are displayed in bold. "MIN": miniImageNet.

	MIN	MIN $\rightarrow$ CUB	CUB	CUB $\rightarrow$ MIN
Finetuning	**r=0.82, p=2e-4**	**r=0.71, p=3e-3**	**r=0.96, p=7e-9**	r=0.28, p=0.31
MAML	**r=-0.77, p=8e-4**	**r=-0.85, p=6e-5**	r=0.36, p=0.18	**r=0.90, p=4e-6**
Reptile	r=0.27, p=0.3	r=0.50, p=0.06	r=0.3, p=0.28	r=0.31, p=0.27