Meta-Learning with label noise

A step towards label few-shot meta-learning with label noise.

Jeroen M. Galjaard

TU Delft

Meta-Learning with label noise

A step towards label few-shot meta-learning with label noise.

by

Jeroen M. Galjaard

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday May 4, 2023 at 10:00 AM.

Student number:	4654099	
Project duration:	February 1, 2022 – May 4, 2023	
Thesis committee:	Dr. L. Y. Chen,	TU Delft, supervisor
	Dr. A. Panichella,	TU Delft, committee

This thesis is confidential and cannot be made public until May 4, 2023.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Few-shot learning presents the challenging problem of learning a task with only a few provided examples. Gradient-Based Meta-Learners (GBML) offer a solution for learning such few-shot problems. These learners approach the few-shot problem by learning an initial parameterization that requires only a few adaptation steps for new tasks. Although these GMBLs are well-studied with correct training data, few have studied the impact of training them with noisy labels.

In this thesis, we show that GMBLs are negatively affected by label noise. We propose a training strategy (BatMan-CLR) leveraging a novel subsampling approach to address the impact of meta-training with label noise. To train and evaluate the different GMBLs, we implement `nmfw`, a novel framework for extensible training loop definition and few-shot data generation.

Our results show that BatMan-CLR is capable of learning few-shot classification models. We show that our approach can effectively mitigate the impact of meta-training label noise. Even with 60% wrong labels BatMan and Man can limit the meta- testing accuracy drop to 2.5, 9.4, and 1.1 percent points, respectively, with existing meta-learners across the Omniglot, CifarFS, and MiniImagenet datasets.

Acknowledgements

To my family, Theo, Ineke† and Helena, for your support, love, and encouragement that have helped me throughout this endeavor. Without you, none of this would have been possible.

I want to thank Dr. Pérez, Dr. Birke, and Masoud for your insights, dedication, and many fruitful discussions. You have provided countless insights and valuable tips that helped shape this work. I want to extend my thanks to Dr. Panichella, for reading my thesis and availability as a committee member.

Lastly, I want to express my heartfelt gratitude toward Lydia. Your positive mindset and solution-oriented stance have helped tremendously when encountering personal and thesis-related problems.

Jeroen M. Galjaard
Delft, April 2023

Contents

1

Introduction

Neural networks–especially *large and deep* ones–have shown that they can perform increasingly complex tasks [18, 12]. They show an impressive capability to learn tasks as their complexity increases [12, 28, 43]. On the one hand, this comes at the cost of a–sometimes extreme–data hunger to properly train such learners to achieve such impressive levels, requiring up to billions of samples [28, 43]. On the other hand, humans often require significantly fewer examples to adapt to new scenarios. For example, students require only a few instances a teacher provides to understand a new concept. Later, students can rapidly adapt and apply these concepts to new unseen (similar) problems. Students in this setting leverage their knowledge of related topics and apply it to new problems.

Few-shot learning poses this problem to learners, thus requiring a learner to quickly generalize *to adapt to new scenarios*. Each few-shot problem only provides a few samples (shots) per class, with which the learner must predict unknown samples. Ordinarily, in (supervised) classification learning–of *many*-shot learning–a learner has access to a sizable training set to train on. It then predicts labels of unseen samples of *the same classes* to estimate its prediction ability during the evaluation. Few-shot learning requires the learner to learn to generalize, similar to humans. Data in few-shot learning resembles 'mini' datasets containing a small 'training' and 'testing'–or support and query set. The support data provides the needed samples for the learner to generalize and the query to evaluate its ability. From a high level, this is equivalent to the initial supervised setting but with the data split up into mini-episodes.

Few-shot learning with label noise makes this problem even more challenging. Learning to generalize becomes difficult when the learner gets tasks with incorrectly labeled samples during training. Due to the low number of samples per class, a task can become ambiguous with even a few incorrectly provided examples. Tasks may become unclear as different classes may not contain samples from other classes. Moreover, provided samples may lay outside of the intended class (out-of-task label noise), making the concepts seemingly random. Due to the limited number of samples, overfitting to the noisy samples can thus quickly occur, making it more challenging to learn to generalize.

An extensive collection of work exists on 'learning to learn' for few-shot learning. The closest related to our work are metric learning and optimization-based learners. We provide a short overview of these works. **Metric Learners** approach this problem by learning a model that learns a shared representation space [53, 31, 57]. They approach the few-shot learning as a clustering problem, where related inputs should be similar in the learned representation space, represented by a deep neural network. Training these learners episodically incentivizes the learner to adapt rapidly, mimicking the few-shot *testing* scenario during training.

More recently, however, large-scale embedding models–the *many-shot* metric learner–have also shown to be performant few-shot learners [55, 35]. This latter approach treats the few-shot problem as a 'downstream task', i.e., using the model's representations as input to solve a related task. **Optimization Based Learners** takes a different route; rather than optimizing 'embeddings', these aim at optimizing parameterizations. [47] approaches this by modeling the 'learning to learn' with an LSTM that learns to predict a small network–or student. The LSTM then acts as the 'meta-learner', which learns to predict a task-specific model in an episodic fashion. By this approach, the goal is to find an initialization, such that it only requires a few adaptation steps–and thus samples–to generalize to unseen settings.

Model Agnostic Meta-Learning [15] (MAML) is a seminal work that proposes an end-to-end differentiable strategy. As a result, this allows a learner to learn its *own* initialization using gradient descent. Although its formulation allows for an end-to-end learning strategy, its exact formulation does not scale well as more samples, larger models, or long adaptation processes are used. Various works [41, 14, 40, 46, 45, 42] have been proposed to alleviate these issues.

However, limited work [34, 36, 38, 6, 26, 56] is done on few-shot learning with label noise, with most [34, 36] addressing label noise at test time. As such, they assume that clean training data is present to train the few-shot learner and label noise rejection mechanism.

This study concerns the impact of meta-training label noise on meta-learners' ability to generalize. To evaluate the experiments, we implement a flexible and extensible framework for meta-learning under different few-shot scenarios. This developed tool, `nmfw`[1], is part of the work towards this thesis, and is planned to be opened up to the public.
The research questions that this work aims to answer are as follows:

1. How does meta-training with label noise affects a meta-learned model to adapt to new tasks during meta-training?

2. Can existing meta-learners be adapted to mitigate the impact of label noise during meta-training?

3. What is the effect of label noise during meta-testing on meta-learned few-shot classifiers? Does label-noise robust meta-training improve meta-testing robustness against meta-testing label noise?

The remainder of this work is structured as follows. First, we provide the main findings of this work in the format of a pre-print of the written paper in chapter 2. Second, we give additional background information in chapter 3 on core concepts used throughout this work, leveraging visuals to provide intuition behind these concepts. Third, we introduce the developed framework `nmfw` in chapter 4. Fourth, in chapter 5, we discuss additional results from experiments not presented in the research paper. Lastly, we conclude in chapter 6 and discuss directions for future work.

[1] Noisy Meta-learning Framework.

2

Research Paper

BatMan-CLR: Making Few-shots Meta Learners Resilient Against Label Noise

No Author Given

No Institute Given

Abstract The negative impact of label noise is well studied in classical supervised learning yet remains an open research question for meta-learning. Meta-learners aim to adapt to unseen learning tasks by learning a good initial model in meta-training and consecutively fine-tuning it according to new tasks during meta-testing. In this paper, we present the first extensive analysis on state-of-the-art meta-learners, specifically gradient-based N-way K-shot learners, under different levels of label noise. We show that, in the presence of label noise in meta-training, the accuracy of Reptile, iMAML, foMAML, drops by up to 42% on the Omniglot and CifarFS datasets. To strengthen the resilience against label noise, we propose two sampling techniques, namely manifold (Man) and batch manifold (BatMan), which transform the noisy supervised learners into semi-supervised ones. We first construct manifold samples of N-way 2-contrastive-shot tasks through augmentation, learning the embedding through a contrastive loss in meta-training, and then perform classification through zeroing on the embedding in meta-testing. We show that our approach can effectively mitigate the impact of meta-training label noise. Even with 60% wrong labels BatMan and Man can limit the meta-testing accuracy drop to 2.5, 9.4, 1.1 percent points, respectively, with existing meta-learners across the Omniglot, CifarFS, and MiniImagenet datasets.

1 Introduction

Few-Shot Learning (FSL) poses the problem where learners need to quickly adapt to new unseen tasks by using only a low number of samples. Meta-learning [19,5] emerged as a promising solution to this problem. Like humans, meta-learners learn the information at a higher abstraction or meta-level, providing the inductive bias to quickly adapt to new tasks. Among existing meta-learners, gradient-based few-shot learners, e.g., iMAML [18] and foMAML(+ZO) [5,7], have been shown effective to solve N-way K-shot (N, K) problems, which need to learn N classes given only K samples each. Such few-shot learners are composed of two stages, meta-training, and meta-testing, each with their own labeled support and query data sets. Meta-training learns the initial meta-model using two optimization loops. An inner loop adapts the model to a specific task via supervised learning on the support set. Then, an outer loop updates the meta-model based on the task-specific model and the labeled query set. Using a similar structure,

meta-testing verifies how well the meta-model performs on new tasks. First, it uses supervised learning to adapt the meta-model to an unseen task given by a test support set. Then it compares the predicted and given labels of the testing query set to measure the learner accuracy. Class labels are thus critical in both meta-training and meta-testing.

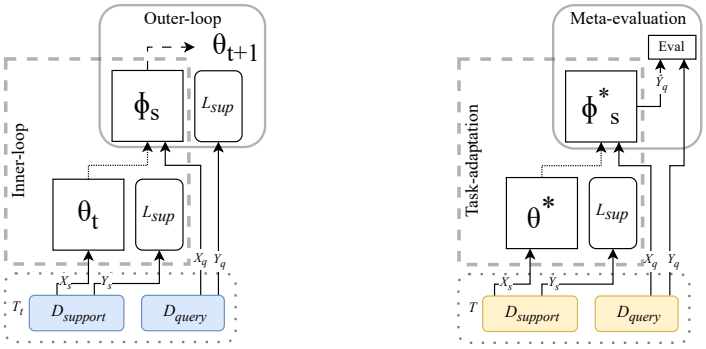
Label noise is more a norm than a rarity and can degrade the performance of supervised learners significantly [20]. Prior studies address label noise mainly in classical supervised learners. In this context, samples hold labels different from the underlying ground truth. In the context of FSL, label noise means that a shot (example) may not correspond to the way (class) it was provided with. This yields a degenerate N-way K-shot problem where ways become indistinguishable since they contain shots of the same ground truth. Such noise can appear in the support set of the meta-training and meta-testing phases, as well as in the query set of the meta-training.

Given the importance of labels in meta-training and meta-testing, only a limited number of studies [14,12] address the challenge of noisy labels in FSL, with a focus on label noise in the testing support set. As the number of samples per class is very limited, e.g., five to ten shots, the task adaptation can be over-parameterized by label noise and lead to significant degradation. Such approaches are specific to meta testing, under a strong assumption that there is no label noise in the training support and query sets. Unfortunately, label noise can appear in meta-testing as well as meta-training and little is known on its impact and resolution. Those methods of learning with noise require clean data to learn a meta-objective. This is the case of [9], where to distill the impact of label noise, it looks to learn which samples are noisy, by adding an additional meta-objective on clean validation data. Also, TADAM [22] proposes task-aware scheduling to learn to filter out *noisy tasks* but only considers corrupting a fraction of support sets during training.

In this paper, we first answer the question of whether state-of-the-art FSL methods are resilient to label noise present in the query and support sets in meta-training. We empirically show that Reptile [17], Eigen-Reptile [4] iMAML [18], and foMAML+ZO [7] are significantly affected by label noise in meta-training, overfitting to noise and degrading the efficacy of any randomly initialized models. To address the noisy labels in meta training, we propose BatMan, which turns any supervised few shot learner into a semi-supervised one by a novel batch manifold sampling and contrastive learning. We learn the embedding in meta-training and then apply a zeroing strategy in meta-testing. Specifically, we turn a noisy N-way K-shot problem into a self-cleansed N-way 2-*contrastive* shot problem. We first augment the original shots and construct contrastive pairs, ensuring the shots are from the same class. We then sample such pairs from the N ways, termed manifold (Man) samples. To lower the probability to get noisy N-ways, i.e., overlapping classes, we draw a batch of such Man samples, termed BatMan sampling. Combining with Decoupled Contrastive Loss (DCL) [23], we can effectively learn the embedding of the initial model which can then be adapted in meta-testing to a new N-way K-shot task.

The specific contributions of this paper are:

1. A first-of-its-kind study on the impact of label noise in meta-training for gradient-based meta-learners.
2. A generic and self-cleansing framework, BatMan-CLR, that turns meta-learners into semi-supervised ones by (batch) manifold sampling N-way 2-contrastive shots.
3. Extensive evaluation on four meta learners, Reptile, EigenReptile, iMAML, foMAML, showing nearly no performance degradation under the presence of up to 60% label noise.



(a) Meta-training (b) Meta-testing.

Figure 1: High-level comparison of meta-learning (left) and meta-testing (right) with Gradient-based Meta-Learners. During each meta-epochs' t inner-loop, the meta-model θ_t is adapted in few steps s to task-specific ϕ, leveraging $(X_s, Y_s) \in \mathcal{D}_{\text{support}}$. The meta-model θ_{t+1} is then learned by relating ϕ's loss on $(X_q, Y_q) \in \mathcal{D}_{\text{query}}$ back to θ_t. Meta-testing evaluates the capability of a trained meta-model θ^* to adapt to new tasks *using query data*. For simplicity, the details pertaining drawing tasks $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}}) = \mathcal{T}$ is left away.

2 Preliminary and Related work

Preliminary on FSL. FSL considers the setting where a learned model must adapt to new settings leveraging only few samples. In this setting meta-learning has emerged as a promising research direction. Gradient-based meta-learners aim to find a meta-model, with parameters θ, capable of quickly adapting into a task specific parameterization ϕ. We consider the N-way K-shot classification problem, which consists of a family of tasks, made up of N classes, each with K samples, simply labeled (N, K)-FSL. We explicitly focus on gradient-based meta-learners which aim to find θ iteratively using a two-step meta-training algorithm

(see Figure 1a). At the beginning of each meta-epoch, the learner selects a task $\mathcal{T}$ and samples two task-specific sets, support $\mathcal{D}_{\text{support}}$ and query $\mathcal{D}_{\text{query}}$, from the training data $\mathcal{D}_{\text{train}}$. More formally a task $\mathcal{T}$ is a tuple of support and query data $\mathcal{T} = (\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}})$, defined as sets of inputs x and targets (labels) y:

$$\mathcal{D}_{\text{support}} = \bigcup_{i=1}^{N} \{(x_j^i, y_j^i)\}_{j=1}^{K},\ \mathcal{D}_{\text{query}} = \bigcup_{i=1}^{N} \{(x_j^i, y_j^i)\}_{j=1}^{Q}.$$

Next, step one transforms θ_t using the features and labels $(X_s, Y_s) \in \mathcal{D}_{\text{support}}$ and a supervised loss function L_{sup} in task-specific parameters ϕ. Then, step two uses ϕ, the data $(X_q, Y_q) \in \mathcal{D}_{\text{query}}$, and L_{sup} to update θ_{t+1} for the next iteration. Note that while the support set $\mathcal{D}_{\text{support}}$ is used during an inner loop to train for a specific task, the query set $\mathcal{D}_{\text{query}}$ is used in an outer loop to learn the meta-model. During meta-training, the support set for a single task is built by randomly selecting N classes, each with K samples, from the training set D_{tr}. For the query set we select Q additional samples for each class included in a task.

Figure 1b shows the steps performed during meta-testing. Meta-testing aims to evaluate the ability of the learned meta-model θ^* to adapt to new tasks. Analogous to meta-training, first a support set $(X_s, Y_s) \in \mathcal{D}_{\text{support}}$ is sampled from test data $\mathcal{D}_{\text{test}}$ to perform an adaptation step. The adaptation step transforms the generic meta-model with parameters θ^* into the task-specific model with parameters ϕ^*, which is then tested on the query set $(X_q, Y_q) \in \mathcal{D}_{\text{query}}$ by comparing its predictions $\hat{Y}_q$ against known labels Y_q.

Examples of gradient-based meta learners include MAML [5], iMAML [18], foMAML(+ZO) [7] and Reptile [17]. MAML updates the meta-model via gradient descent through gradient descent [5]. As this operation is both computationally and memory intensive [5,18,16] many works proposed approximations. iMAML [18] and foMAML (With the zero-ing trick as proposed in [7]) approximate $\nabla_{\theta_t} \mathcal{L}_{\mathcal{D}_{\text{query}}}(\phi_s)$ by leveraging $\nabla_{\phi_s} \mathcal{L}_{\mathcal{D}_{\text{query}}}(\phi_s)$, dropping the need for gradient descent through gradient descent altogether. Whereas foMAML directly assumes that the higher-order components of the meta-gradient can be ignored altogether, iMAML enforces that it can exactly calculate the meta-gradient through more adaptation steps s and weight regularization. Reptile [17] learners drops $\mathcal{D}_{\text{query}}$ completely during meta-training and approximate the meta-gradient as $\theta_t - \phi_s$, effectively stepping towards the inner-loop parameterization. Eigen Reptile [4] builds on this by decomposing the inner-optimization path $[\theta_t, \phi_1, \ldots, \phi_s]$ and stepping towards the direction with the largest variance.

Label noise in FSL. Considering label noise poses a major challenge to meta-learners, especially in the absence of a clean data set that can be used as ground truth during the training phase. Although a large collection of work exists on robust supervised learning [21,11,24], these are not directly applicable to meta-learners due to the limited number of samples available during each adaptation process. Recognizing the presence of label noise, related studies [22,13,9,12,15,12] mainly focus on distilling the label noise appearing in *meta-testing* by explicitly

studying the noise patterns [22,12,13], using soft-relabeling [15] through clustering or re-weighting suspicious samples [9] based on additional ground truth of *clean* data. To our best knowledge, Eigen Reptile [4] is the only study that addresses noisy *training data* in FSL by updating the inner loop only along the direction of highest variance. However such an approach can only be applied on Reptile, lacking generalization to other state-of-the-art meta learners.

Table 1: Meta-test accuracies on clean meta-test data, following training on varying levels of meta-train noise. Each learner was validated with 2048 (5, 5)-FSL new meta-test tasks with transductive inference, using a query size of 15. '+ZO' indicates trained with Zero-ing trick [7].

Alg.	$\epsilon = 0.0$	$\epsilon = 0.3$	$\epsilon = 0.6$
Reptile	$65.5_{\pm 0.241}$	$58.6_{\pm 0.258}$	$51.8_{\pm 0.253}$
foMAML+ZO	$69.5_{\pm 0.255}$	$65.2_{\pm 0.265}$	$40.3_{\pm 0.207}$
iMAML	$64.0_{\pm 0.242}$	$55.9_{\pm 0.257}$	$46.3_{\pm 0.243}$
Eigen Reptile	$65.3_{\pm 0.243}$	$58.1_{\pm 0.272}$	$52.7_{\pm 0.241}$

(a) CifarFS results.

$\epsilon = 0.0$	$\epsilon = 0.3$	$\epsilon = 0.6$
$92.5_{\pm 0.125}$	$79.7_{\pm 0.214}$	$71.5_{\pm 0.240}$
$99.3_{\pm 0.037}$	$97.7_{\pm 0.067}$	$90.3_{\pm 0.148}$
$96.9_{\pm 0.110}$	$91.0_{\pm 0.176}$	$82.6_{\pm 0.192}$
$93.6_{\pm 0.122}$	$83.6_{\pm 0.189}$	$73.4_{\pm 0.235}$

(b) Omniglot results.

The impact of label noise Here we start with an empirical study to motivate the need of noise resilience in FSL. We investigate the effect of label noise on two representative datasets, CifarFS [2] and Omniglot [10], in a (5, 5)-FSL setting with a query set of 15 samples per class. The details of the datasets and experiments can be found in Section 4.1. We train re-implementations of four different meta-learners, Eigen-Reptile (ER) [4], Reptile [17], first order MAML with Zero Out (foMAML+ZO) [7], and implicit MAML (iMAML) [18], using comparable hyper-parameters. We consider a symmetric label noise setting, where for each class i, a fraction ϵ of its samples are corrupted with a label $j \neq i$ with uniform probability across all other classes.

Table 1 shows the meta-test accuracy obtained by each meta-learner under varying degrees of corrupted training labels, $\epsilon = [0.0, 0.3, 0.6]$. Note that $\epsilon = 0.0$ means no noise, i.e., all clean labels. Reported results are the average across three runs with the standard deviation. For both datasets and all meta-learners, one can clearly see a significant performance degradation as the noise ratio increases. On the CifarFS dataset, accuracy drops across all meta-learners on average by 10.1% and 27.4% under 30% and 60% corrupted labels, respectively. The Omniglot dataset shows similar trends but more limited in amplitude, with 8.1% and 17.0% average degradation. This is due to the fact that the Omniglot dataset is easier to learn. Indeed, all meta-learners obtain an accuracy score above 90% under zero noise. Interestingly, although ER is the sole meta-learner that explicitly tries to counter noise, it is not always the most

robust one. Under moderate noise, i.e. 30%, foMAML+ZO is the least affected with accuracy drops of 6.2% and 1.6% for CifarFS and Omniglot, respectively. Only under heavy noise, i.e. 60%, on the CifarFS dataset ER is the least affected (accuracy drop of 19.3%) and able to beat the others by 0.9 percentage points in higher accuracy. More in general, Reptile, ER and iMAML show a higher but almost linear impact of noise, while foMAML+ZO degrades less with 30% noise but gets much worse under 60% noise. Overall, the results underline the need for better noise resilience across all meta-learners.

Algorithm 1 Pseudocode for Man Sampling. BatMan is achieved through multiple Man samples.

function ManSampling($\mathcal{D}$, N)
Require: `Augment` feature augment function.
  $\mathcal{D}_{\text{samp}} = \{\}$
  for $i \in N$ **do**
    $x \leftarrow rand\,((x, y) \in \mathcal{D} : y = i)$
    $a_1 \leftarrow \texttt{Augment}(x)$
    $a_2 \leftarrow \texttt{Augment}(x)$
    $\mathcal{D}_{\text{samp}} \leftarrow \mathcal{D}_{\text{samp}} \cup \{(a_1, i), (a_2, i)\}$
  end for
  return $\mathcal{D}_{\text{samp}}$
end function

Algorithm 2 General Inner-loop structure with BatMan-CLR for MAML style learners.

Require: α inner-loop learning rate.
function (Bat)Man-CLR($\phi, task, s, v$)
  $\mathcal{D}_s, \mathcal{D}_q = task$
  $X_s, Y_s = \mathcal{D}_s$
  $X_q, Y_q = \mathcal{D}_q$
  for $I_i \in [\texttt{BatMan}(\mathcal{D}_s)]_{i=1}^{s}$ **do**
    $\phi \leftarrow \phi - \frac{\alpha}{s} \nabla_\phi \sum_{I_j \in I_i} \mathcal{L}_{con}(\phi(I_i)))$
  end for
  $I_{outer} = \texttt{BatMan}_{outer}(\mathcal{D}_q)$
  $\mathcal{L}_{outer} = \sum_{I_j \in I_{outer}} \mathcal{L}_{con}(\phi(I_j))$
  return $\phi, \mathcal{L}_{outer}$
end function

3 Proposed method

The core challenge of dealing with noise in meta-(or few shots) learning is that labels lose meaning and a standard supervised approach can be misguided by the wrongly labeled samples. This challenge is amplified in the FSL setting as the limited number of samples (shots) in each class (way) makes it harder to isolate the noise from the signal in each class. In other words, the few clean samples– those corresponding to the class they were provided as–may not be enough to appropriately guide the gradient descent algorithm in the correct direction of the class label due to the significant presence of samples with corrupted labels. Thus, our approach aims at building clean ways and shots–such that each becomes more likely to be a unique one–through re-sampling. Specifically, the Man sampling described below ensures that each class consists of clean shots, as these are created via augmentation, guaranteeing positive samples for individual shots, rather than leveraging *other* shots provided with an equal label. We further introduce batches, with BatMan sampling, to increase the likelihood of observing all N classes in a single inner-loop step. In this section we go over the three main components of our proposed method, namely, i) Man and BatMan re-

sampling, ii) 'semi'-supervised meta-training with a contrastive loss, and iii) classification of new tasks (meta-testing) leveraging a zeroing trick. Together these components can be incorporated into *existing meta-learning algorithms* to achieve label noise robustness. In section 4 we provide results for both Man and BatMan re-sampling, where both integrate the two latter steps and differ only in the type of sampling performed. Figure 2 provides a graphical depiction of our proposed method applied on a noisy (3, 2)-FSL task.

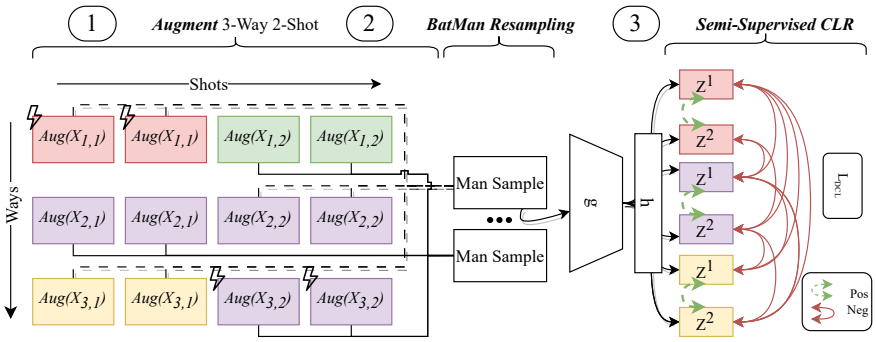


Figure 2: Overview label learning with `BatMan`-CLR on a noisy 3-way 2-shot $(3, 2)$-Few-Shot (FS), colors represent ground truth classes, lightning bolts indicate noisy samples (shots 1 way 1, shot 2 way 3). i. For each sample independent random augmentations are made through `Aug`; ii. (3, 2)-FSL semi-supervised sub-tasks are created through Man sampling (Algorithm 1); iii. semi-supervised samples in each sub-task are fed into a contrastive loss, to be jointly optimized.

Man and `BatMan` re-sampling. We start with an (N, K)-FSL problem with noisy labels and observe that we can re-frame it as an $(N, 2)$-FSL problem by sampling 1 data point from each class, assuming each corresponds to a unique class, and creating one more sample for each class by means of augmentation. Thus, we end up with two *clean* shots for each of the N classes in the sampled task. Since we sample N observations from all NK in $\mathcal{D}_{\text{test}}$, we end up with *up to N actual classes*, as some classes may end up contributing more than one observation to a sub-sampled task. In this manner, many potential $(N, 2)$ sub-tasks can be created, each corresponding to a different task 'manifold' [17] due to noisy labels. We coin this sampling approach **Man**ifold' (Man) sampling, described as pseudo-code in Algorithm 1. Note that to avoid introducing a bias towards the original image, for each shot we use `Aug` draw two random augmentations. As a simple extension of Man sampling, we propose a **Bat**ched Manifold sampling, `BatMan`, where several Man samples are batched together to employ more samples in a single step. By grouping a batch of Man samples together, we can jointly optimize multiple 'sub-problems'. Additionally, this increases the likelihood of considering *all* query classes together in the calculation of meta-gradients.

This process is illustrated in Figure 2, with a 3-way 2-shot FSL setting, where each row index represents an FSL-class and each column an instance. The classes are illustrated with different colors, where initially there are 2 shots of each of the green, purple and yellow classes. Noise has caused the class of the first shot in the green class to have a red class label, and the second shot of the yellow class to have a purple class label. A pool of Man samples are generated by sampling 1 shot from each of 3 classes, and augmenting it twice to have an $(N, 2)$ FSL task. A batch of such tasks is generated to perform a meta-training step through gradient descent.

In the presence of noise, the process of cleaning up the classes is necessary as samples from two apparently different classes may actually come from the same class. In the simplest case with 2 classes and a probability p that a sample has a ground truth label, the classes resulting from the Man sampling process belong to different classes with probability $p^2 + (1-p)^2$, as either both lack noise or both are noisy. On the contrary, the 2 samples selected actually belong to the same class with probability $2p(1-p)$. In general, with N classes there are $N!$ combinations in which the Man samples actually correspond to the N *different* classes while there are N^N combinations to select the N samples, with replacement. Let us consider the case of symmetric noise where a sample has a ground truth label i with probability p and that the sample mislabeled as a different class $j \neq i$ with probability $(1-p)/(K-1)$. The probability of obtaining a clean selection of classes can be then posed as the probability of obtaining one of the $N!$ combinations in which this occurs. To represent each possible selection we employ permutation matrices. Let P_N^i be the $N \times N$ i-th permutation matrix out of the $N!$ such matrices. For instance, in the $N = 3$ case, we have 6 different permutation matrices, i.e.,

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&0&1\\0&1&0\end{bmatrix}, \begin{bmatrix}0&1&0\\1&0&0\\0&0&1\end{bmatrix}, \begin{bmatrix}0&1&0\\0&0&1\\1&0&0\end{bmatrix}, \begin{bmatrix}0&0&1\\1&0&0\\0&1&0\end{bmatrix}, \begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}.$$

Let us also define the $N \times N$ matrix Q with entries q_{ij} such that $q_{ii} = p$ and $q_{ij} = (1-p)/(N-1)$ for $i \neq j$. The probability of obtaining one of these valid permutations under Man sampling can thus be obtained as the trace of the matrix $P_N^i Q$. As a result, the probability of obtaining a clean selection of ways can be expressed as

$$\sum_{i=1}^{N!} \text{trace}(P_N^i Q),$$

considering all the possible valid selection of samples that lead to a set of N different classes in the sample. As this probability becomes smaller with increasing label noise p, BatMan sampling helps by introducing additional samples that increase the likelihood of observing all N classes in a single inner loop.

'Semi'-supervised meta-training with contrastive loss. Although resampling allows for likely valid $(N, 2)$ sub-tasks, *which classes* they contain remains unknown. As such these sub-tasks can be considered as *semi-supervised*,

as there are now at most N classes[1]. To aid this we incorporate a contrastive loss to allow for joint optimization of semantically misaligned sub-tasks. Note that from the sampled augmentations obtained in step 1 of Figure 2, we artificially build positive and negative pairs. These positive and negative pairs can then be optimized under a contrastive learning strategy. Here we use the Decoupled Contrastive Loss (DCL) [23], which has shown to be particularly well-suited for small data sets, although it can be easily replaced by alternative contrastive losses. As shown in Figure 2 with the 3-way 2-shot FSL setting, once samples are augmented and the BatMan sampling applied, the batches are used in a meta-training step where the embeddings z are computed and contrasted using a contrastive loss.

Classification of new tasks. The trained meta-model θ^* using the contrastive loss produces embeddings not classes as output. To solve this we append an additional fully connected layer $c_0 = (\boldsymbol{W}, \underline{\boldsymbol{b}})$ with $\boldsymbol{W} = \mathbf{0}$ and $\underline{\boldsymbol{b}} = \underline{\mathbf{0}}$ to the model. This approach decouples the embedding learning from the classification task. Similar to [7], this allows to treat the model as a semi-supervised meta-learned backbone that can rapidly adapt to different tasks. The resulting θ^* can then be treated as a semi-supervised embedding space, on which the fully connected layer can be adapted. To train this layer we employ the Zeroing Out trick [7] in which the weights are initialized as zero. We found that applying the Zeroing Out trick on the *classification* layer greatly impacts the learners' performance because it allows leveraging the optimized embedding from the pre-trained meta-model θ^*. This is because the stochastic gradient descent will directly use the embeddings as activations, without noise introduced caused by random weights and biases.

4 Evaluation Results

In this section, we present the effectiveness of BatMan-CLR in enhancing the noise resilience for state-of-the-art meta-learners, namely Reptile, iMAML, and foMAML+ZO, under the presence of different noise levels. We further include EigenReptile, a noise-aware FSL as one of the baselines.

4.1 Setup

We consider three data sets in a (5, 5)-FSL setting: Omniglot, Cifar Few-Shot (CifarFS), and miniImagenet. To emulate the label noise, we add symmetric random noise according to a specified label noise percentage of 30 and 60% to the meta-training split of each dataset. For instance, for experiments with 60% noise, we randomly select 60% samples of each class in the meta-train split, and assign them to a different class *within the same train split*. As such, both the support and query sets have an *expected* noise percentage of 60%. Reported meta-test results are on *clean* data, to evaluate the learners under a base-case

[1] As opposed to $N \times K$ in the worst case with noise.

scenario. The support set is of size 5 (15), and query set 15 (Reptile learners), as in [5].

We incorporate Man and BatMan sampling into Reptile (Rep.), EigenReptile (ER.), iMAML (iM.) and foMAML with Zeroing-trick (fM.). Each learner uses a ConvNet-4 architecture with 64 filters, and a linear layer with output dimension $\mathbb{R}^{128}$. We use proximal weight decay[2] on iMAML, and foMAML+ZO resets its final layer to zero at the beginning of each inner-loop. The learners were trained with the hyper-parameters provided in the original paper, except the following changes. We use a fixed learning rate of 0.001 for both the inner-and outer-loop (Reptile). Eigen-Reptile and Reptile run with 7 inner-loop steps, iMAML with 12 (16 for Omniglot), and foMAML with 5. iMAML's proximal decay was set to 0.5 (2 for Omniglot). Each learner was meta-tested after 5K, 15K (10K), 15K (10K), training outer-loop steps (iMAML) respectively for Omniglot, CifarFS and MiniImagenet. For augmentations on CifarFS and MiniImagenet we use the augmentations proposed in [1]. For Omniglot, we follow [3], applying one of random crop, affine transform or perspective transform. During meta-testing the learned meta-model model is further trained with 10 steps on Omniglot and CifarFS, and 20 steps on MiniImagenet for task-specific adaptation.

When applying BatMan-CLR on those meta learners, we keep the same model sizes with the addition of a larger 'decision' head $\mathbb{R}^{128}$ rather than $\mathbb{R}^{5}$. The batch size of BatMan is set to 5 for all inner-loop adaptations[3], and 15 for the meta-gradient calculation of iMAML and foMAML. For each support sample 5 augmentations are created, whereas each query sample is augmented twice, allowing the inner-loop to sample more diverse tasks. The final reported testing accuracy is averaged over 3 testing runs, using 2048 tasks sampled from the test split.

4.2 Testing accuracy

The results of BatMan-CLR on noisy CifarFS, Omniglot, and MiniImagenet are summarized in Table 2 and Table 3, respectively. We report testing accuracy with both Man and BatMan for CifarFS and Omniglot. Prior to analyzing the results, we highlight all of these learners face significant degradation under label noise, as shown in Section 2.

BatMan-CLR clearly strengthens the resilience of all FSL on all three datasets, showing a marginal decrease in the testing accuracy. On Omniglot, when encountering the label noise in meta-training, all learners can still learn effective initial models for task adaptation, reaching an accuracy of around 96%. In fact, all learners are able to display a performance under label noise similar to that without noise. As for CifarFS, we observe similar results to Omniglot, with most learners reaching a test accuracy between 62-64%, except when applying Man sampling on Eigen-Reptile. These results strongly validate the effectiveness of

[2] Weight decay centered around θ_t, as per [18].

[3] For Reptile style learners this then yields 'minibatches', as only 5 of 15 shots are used each adaptation step.

Table 2: (clean) Meta-test results with 95^{th} Confidence Interval ($\overline{acc}_{\pm \text{ CI95}}$) of Meta-Pretrained models (5, 5)-FSL with different noise levels ϵ during training.

Alg.	Sampler	ϵ=0.0	ϵ=0.3	ϵ=0.6
Reptile	BatMan	$66.5_{\pm 0.168}$	$65.0_{\pm 0.170}$	$64.1_{\pm 0.170}$
	Man	$61.8_{\pm 0.176}$	$62.0_{\pm 0.175}$	$61.4_{\pm 0.173}$
Eigen Reptile	BatMan	$66.3_{\pm 0.171}$	$64.4_{\pm 0.170}$	$63.8_{\pm 0.176}$
	Man	$61.7_{\pm 0.170}$	$55.8_{\pm 0.378}$	$52.3_{\pm 0.365}$
foMAML	BatMan	$66.6_{\pm 0.167}$	$65.2_{\pm 0.169}$	$64.8_{\pm 0.164}$
	Man	$66.2_{\pm 0.164}$	$64.8_{\pm 0.165}$	$64.0_{\pm 0.165}$
iMAML	BatMan	$64.2_{\pm 0.185}$	$62.7_{\pm 0.250}$	$62.9_{\pm 0.203}$
	Man	$62.8_{\pm 0.172}$	$62.6_{\pm 0.168}$	$61.7_{\pm 0.169}$

(a) CifarFS results

ϵ=0.0	ϵ=0.3	ϵ=0.6
$97.9_{\pm 0.070}$	$97.3_{\pm 0.078}$	$96.2_{\pm 0.100}$
$97.8_{\pm 0.068}$	$97.8_{\pm 0.068}$	$97.7_{\pm 0.070}$
$92.6_{\pm 0.128}$	$93.0_{\pm 0.119}$	$93.2_{\pm 0.117}$
$93.7_{\pm 0.116}$	$93.9_{\pm 0.114}$	$94.0_{\pm 0.111}$
$98.2_{\pm 0.066}$	$98.2_{\pm 0.063}$	$98.0_{\pm 0.067}$
$98.1_{\pm 0.062}$	$98.1_{\pm 0.062}$	$98.1_{\pm 0.061}$
$97.5_{\pm 0.078}$	$98.1_{\pm 0.069}$	$98.3_{\pm 0.063}$
$97.8_{\pm 0.076}$	$98.2_{\pm 0.074}$	$98.2_{\pm 0.064}$

(b) Omniglot results.

BatMan, which self-cleanses the 'shots' by creating contrastive pairs and 'ways' by batching Man samples.

In terms of comparison between BatMan and Man, there is a visible advantage of using BatMan, especially on a more difficult CifarFS. This suggests that taking steps with more information, as in BatMan, provides greater benefits than taking a larger number of simpler steps, as in Man. Zooming into the performance of different learners on CifarFS, the difference in testing accuracy between Man and BatMan on CifarFS is smaller with foMAML and iMAML, compared to Reptile and EigenReptile. This can be explained by the fact that in our experiments, the MAML style learners utilize BatMan to calculate the meta-gradient, resulting in more informative updates. As Reptile learners do not calculate their meta-gradients utilizing the query data, but the inner-optimization process.

An observation worth mentioning is that Eigen Reptile paired with Man, deteriorates under noise on CifarFS. We speculate that this is due to the fact that in Eigen Reptile the meta-gradient approximation is performed by selecting the optimization direction with the highest variance. However, a high level of noise introduces a high variance into the optimization directions, making it harder to select an appropriate direction even with the use of Man. By employing the less noisy BatMan estimation strategy, the learner is able to better select an optimization direction and achieves a performance comparable to Reptile.

MiniImagenet. We further evaluate BatMan-CLR on MiniImagenet, a more difficult dataset consisting of more diverse classes and larger inputs, and summarize the results in Table 3. Under BatMan-CLR, the testing accuracy of most meta learners has minor drops, i.e., ranging between .5 to 1%.

Table 3: (clean) Meta-test accuracy with 95^{th} Confidence Interval ($\overline{acc}_{\pm \text{CI95}}$) BatMan on (5, 5)-FSL MiniImagenet with different noise levels (ϵ). White columns correspond to supervised learners, grey columns to `BatMan`-CLR.

Alg.	ϵ=0.0		ϵ=0.3		ϵ=0.6	
Reptile	$54.2_{\pm 0.206}$	$53.2_{\pm 0.163}$	$27.8_{\pm 0.141}$	$52.8_{\pm 0.145}$	$24.6_{\pm 0.143}$	$52.1_{\pm 0.147}$
Eigen Reptile	$58.7_{\pm 0.250}$	$50.9_{\pm 0.146}$	$44.8_{\pm 0.200}$	$50.5_{\pm 0.148}$	$24.8_{\pm 0.140}$	$50.5_{\pm 0.144}$
foMAML	$52.2_{\pm 0.217}$	$51.5_{\pm 0.216}$	$37.1_{\pm 0.179}$	$51.2_{\pm 0.218}$	$28.2_{\pm 0.145}$	$51.5_{\pm 0.217}$
iMAML	$53.9_{\pm 0.215}$	$50.4_{\pm 0.218}$	$45.5_{\pm 0.211}$	$50.5_{\pm 0.221}$	$20.0_{\pm 0.076}$	$50.4_{\pm 0.212}$

4.3 Ablations

First, we consider the impact of supervised task generation by training meta-learners in an unsupervised setting. Similar to UMTRA [8] and CACTUS [6] we construct $(5, 5/15)$-FSL tasks (MAML/Reptile) by drawing 5 random images– irrespective of their provided label. To construct shots, $K + Q$ augmentations are created to provide the necessary support and query shots, $|\mathcal{D}_{\text{support}}| = K, |\mathcal{D}_{\text{query}}| = Q$ sets. The Table 4 shows the results on Omniglot and CifarFS in the first row (SSL). The hyper-parameters and loss function were kept the same as `BatMan`-CLR, with the meta-batch size increased to 25. Although this approach shows similar performance to BatMan-CLR on Omniglot, on CifarFS a considerable performance gap of 13.8-11% points compared to BatMan-CLR (gray rows). Indicating that BatMan-CLR profits from seeing *more* unique samples during the inner-loop adaptation.

Second, we replace BatMan with a random manifold sampler (RanMan) to investigate the impact of the BatMan sampler. Results are shown in **??**. We pair RanMan with BatMan in different configurations for the inner and outer loop, again evaluated on Omniglot and CifarFS. Note that as Reptile and Eigen Reptile do not leverage the query set, we only consider replacing its inner-loop sampler. We keep hyper-parameters equivalent to those used in the corresponding BatMan-CLR setting.

In general, the learners trained with RanMan in the outer loop show an uplift in accuracy as the noise level increases. Learners show an increase in accuracy of around 2-8% and 2-3% on Omniglot and CifarFS, comparing noiseless with $\epsilon = 0.6$, whereas BatMan-CLR sees a slight drop while staying ahead of RanMan across the board. Showing that BatMan has the capability 'self' clean. An interesting exception is present on Omniglot combined with Eigen Reptile when increasing the noise level to 0.3 from 0(Table 4a). This is expected, as with an increased noise level, we expect more unique classes in a constructed task, resulting in Robin samples being more likely to be consistent. When only replacing either inner or outer loop sampler for for iMAML and foMAML with RanMan, we see that the contribution of the inner loop is less significant than the outer loop. Showing that BatMan-CLR is also an effective strategy when replacing only the outer-loop (R/B).

Table 4: (clean) Meta-test accuracy with confidence intervals ($\overline{acc}_{\pm \text{ CI95}}$), meta-trained with different Inner/Outer samplers on (5, 5)-FSL tasks with different noise levels (ϵ). Sampler indicates applied Inner/Outer loop sampler; Random (batched) Manifold '(R)anMan', Batched Manifold '(B)atMan'. SLL represents a meta-learned model trained in an unsupervised few-shot learning strategy. Gray rows are reprinted results of Table 2.

(ϵ)	Sampler	Reptile	E. Reptile	foMAML	iMAML	Reptile	E. Reptile	foMAML	iMAML
SSL	default	96.0±0.098	95.1±0.134	94.7±0.119	97.0±0.090	55.0±0.167	54.5±0.164	52.8±0.158	54.5±0.236
0.0	R/R	65.3±0.287	82.5±0.207	94.5±0.120	94.5±0.119	53.9±0.220	57.2±0.272	58.0±0.236	56.9±0.288
	R/B	-	-	98.3±0.048	98.1±0.071	-	-	62.0±0.234	60.9±0.296
	B/R	-	-	94.1±0.123	94.9±0.114	-	-	58.4±0.236	58.0±0.238
	B/B	97.9±0.070	92.6±0.128	98.2±0.066	97.5±0.078	66.5±0.168	66.3±0.171	66.6±0.167	64.2±0.185
0.3	R/R	69.5±0.271	71.9±0.270	96.7±0.091	96.7±0.091	55.3±0.219	57.8±0.227	59.7±0.233	58.3±0.287
	R/B	-	-	98.4±0.047	98.3±0.066	-	-	61.8±0.236	60.3±0.234
	B/R	-	-	96.9±0.089	96.5±0.097	-	-	59.7±0.232	59.2±0.240
	B/B	97.3±0.078	93.0±0.119	98.2±0.063	98.1±0.069	65.0±0.170	64.4±0.170	65.2±0.169	62.7±0.250
0.6	R/R	73.1±0.272	73.5±0.259	98.0±0.073	97.9±0.073	56.4±0.221	58.8±0.227	60.5±0.243	60.0±0.289
	R/B	-	-	98.4±0.047	98.2±0.068	-	-	61.1±0.237	60.3±0.286
	B/R	-	-	97.9±0.072	97.8±0.075	-	-	60.5±0.235	59.7±0.238
	B/B	96.2±0.100	93.2±0.117	98.0±0.067	98.3±0.063	64.1±0.170	63.8±0.176	64.8±0.164	62.9±0.203

(a) Omniglot results. (b) CifarFS results.

5 Conclusion

Motivated by the ubiquitous presence of label noise, we empirically unveil the impact of label noise on existing few-shots meta learners, with a particular focus on noise in meta-training. As the number of shots per class is low, the label noise can be exceedingly detrimental to meta learners and extremely challenging to address. To enhance the resilience against label noise for few-shots learners, we propose BatMan–a generic approach that turns supervised FS tasks into semi-supervised ones. BatMan is capable of self-cleansing the noisy N-way K shots instance by (i) batch manifold sampling that re-constructs N-way 2-contrastive-shots via augmentation, and (ii) introducing DCL contrastive loss. Our results on three datasets, Omniglot, CifarFS and MiniImagenet, show that BatMan can maintain the effectiveness of few-shot learners independent of the presence of label noise levels, i.e., reserving almost 30% accuracy degradation. In terms of future work, we will extend BatMan on label noise present in meta testing, different contrastive losses, as well as class awareness.

References

1. Bachman, P., Hjelm, D., Buchwalter, W.: Learning Representations by Maximizing Mutual Information Across Views https://github.com/Philip-Bachman/amdim-public.
2. Bertinetto, L., Henriques, J., Torr, P.H.S., Vedaldi, A.: META-LEARNING WITH DIFFERENTIABLE CLOSED-FORM SOLVERS
3. Boutin, V., Singhal, L., Thomas, X., Serre, T.: Diversity vs. Recognizability: Human-like generalization in one-shot generative models. ArXiv (2022). https://doi.org/10.48550/ARXIV.2205.10370
4. Chen, D., Wu, L., Tang, S., Yun, X., Long, B., Zhuang, Y.: Robust Meta-learning with Sampling Noise and Label Noise via Eigen-Reptile. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 3662–3678. PMLR (2022), https://proceedings.mlr.press/v162/chen22aa.html
5. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. 34th International Conference on Machine Learning, ICML 2017 **3**, 1856–1868 (2017), http://proceedings.mlr.press/v70/finn17a/finn17a.pdf
6. Hsu, K., Levine, S., Finn, C.: Unsupervised Learning via Meta-Learning (1 2019)
7. Kao, C.H., Chiu, W.C., Chen, P.Y.: MAML is a Noisy Contrastive Learner in Classification. The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022 (2021), https://openreview.net/forum?id=LDAwu17QaJzhttp://arxiv.org/abs/2106.15367
8. Khodadadeh, S., Bölöni, L., Shah, M.: Unsupervised Meta-Learning for Few-Shot Image Classification
9. Killamsetty, K., Li, C., Zhao, C., Chen, F., Iyer, R.: A Nested Bi-level Optimization Framework for Robust Few Shot Learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 36, pp. 7176–7184. OpenReview.net (2022). https://doi.org/10.1609/aaai.v36i7.20678, https://openreview.net/forum?id=LDAwu17QaJzwww.aaai.org
10. Lake, B.M., Salakhutdinov, R., Gross, J., Tenenbaum, J.B.: One shot learning of simple visual concepts. Proceedings of the Annual Meeting of the Cognitive Science Society **33**(33) (2011), http://web.mit.edu/brenden/www/charactervideos.html.
11. Li, M., Soltanolkotabi, M., Oymak, S.: Gradient Descent with Early Stopping is Provably Robust to Label Noise for Overparameterized Neural Networks. Tech. rep. (2020)
12. Liang, K.J., Rangrej, S.B., Petrovic, V., Hassner, T.: Few-shot Learning with Noisy Labels. CVPR (2022). https://doi.org/10.48550/arXiv.2204.05494, https://doi.org/10.48550/arXiv.2204.05494https://arxiv.org/pdf/2204.05494.pdf
13. Lu, J., Jin, S., Liang, J., Zhang, C.: Robust few-shot learning for user-provided data. IEEE Transactions on Neural Networks and Learning Systems **32**(4), 1433–1447 (4 2021). https://doi.org/10.1109/TNNLS.2020.2984710
14. Mazumder, P., Singh, P., Namboodiri, V.P.: RNNP: A Robust Few-Shot Learning Approach. In: IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021. pp. 2663–2672. IEEE (2021). https://doi.org/10.1109/WACV48630.2021.00271, https://doi.org/10.1109/WACV48630.2021.00271

15. Mazumder, P., Singh, P., Namboodiri, V.P.: RNNP: A Robust Few-Shot Learning Approach. Tech. rep. (2021)
16. Muhammad Abdullah Jamal, Wang, L., Gong, B.: A Lazy Approach to Long-Horizon Gradient-Based Meta-Learning. International Conference on Computer Vision pp. 6577–6586 (2021), https://openaccess.thecvf.com/content/ICCV2021/papers/Jamal_A_Lazy_Approach_to_Long-Horizon_Gradient-Based_Meta-Learning_ICCV_2021_paper.pdf
17. Nichol, A., Achiam, J., Schulman, J.: On First-Order Meta-Learning Algorithms pp. 1–15 (2018), http://arxiv.org/abs/1803.02999
18. Rajeswaran, A., Kakade, S.M., Finn, C., Levine, S.: Meta-learning with implicit gradients. Advances in Neural Information Processing Systems **32**(NeurIPS), 1–12 (2019)
19. Schmidhuber, J.: Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook (1987)
20. Song, H., Kim, M., Park, D., Shin, Y., Lee, J.G.: Learning from noisy labels with deep neural networks: A survey. IEEE Transactions on Neural Networks and Learning Systems pp. 1–19 (2022). https://doi.org/10.1109/TNNLS.2022.3152527
21. Wang, Z., Hu, G., Hu, Q.: Training Noise-Robust Deep Neural Networks via Meta-Learning. Tech. rep. (2020)
22. Yao, H., Wang, Y., Wei, Y., Zhao, P., Mahdavi, M., Lian, D., Finn, C.: Meta-learning with an Adaptive Task Scheduler
23. Yeh, C.H., Hong, C.Y., Hsu, Y.C., Liu, T.L., Chen, Y., Lecun, Y.: Decoupled Contrastive Learning (2022)
24. Yu, X., Liu, T., Gong, M., Zhang, K., Batmanghelich, K., Tao, D.: Label-Noise Robust Domain Adaptation (2020)

3

Background

Before we continue with the details in the rest of this work, we provide a primer on the three core concepts used in this work. We provide visual representations to help strengthen the understanding of these concepts.

First, we introduce the few-shot learning problem more formally. Additionally, we further detail the different types of few-shot learning settings considered in this work–with noisy labels and unsupervised. Second, we discuss gradient-based meta-learning more in-depth and provide its conceptual pseudocode. Lastly, we discuss representation learning focusing on contrastive style learning as is used in our proposed BatMan-CLR.

3.1. Few-shot learning (FSL)

Few-shot learning challenges a learner to predict (a property of) *unseen classes*, e.g., labels, based on a small set of supporting data. Contrary to typical supervised learning, which learns *a fixed set of classes* to predict properties of *unseen instances of the same classes*. This section provides a more formal definition of the few-shot problems. Afterward, we discuss the generation of few-shot learning tasks from existing datasets and provide a short overview of classification benchmarks used in this work. We conclude by introducing *label noisy* few-shot learning.

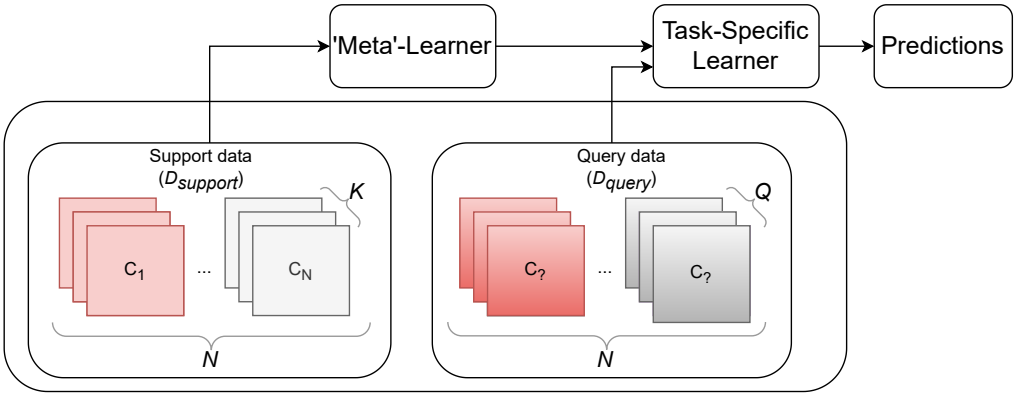


Figure 3.1: General N-way-k-shot instance. For each of the N 'classes' (ways), there are k support samples (shots) provided in $\mathcal{D}_{\text{support}}$ *including their labels*. The corresponding query set $\mathcal{D}_{\text{query}}$ contains 'evaluation' samples a learner must predict given $\mathcal{D}_{\text{support}}$.

Figure 3.2 shows how a (single) few-shot task is used. Solid colors represent samples for which labels are provided (support), and those with a gradient represent unlabeled the unlabeled query data. We note that during the training phase of meta-learners, the query sample labels are provided to learn to generalize. The general recipe is as follows. First, a few-shot learner uses the support samples to adapt to the task. After generalizing, the task-specific learner predicts the query's labels. Each few-shot problem effectively forms its own *independent* (supervised) dataset, similar to regular training in

supervised learning. These few-shot tasks each provide a small set of support data $\mathcal{D}_{\text{support}}$, generally N different classes with K samples each, and challenge a few-shot learner to classify query samples. Let us more formally define N-way K-shot[1] (N, K) few-shot tasks as follows,

$$\mathcal{D}_{\text{support}} = \bigcup_{i=1}^{N} \{(x_j^i, y_j^i)\}_{j=1}^{K}, \; \mathcal{D}_{\text{query}} = \bigcup_{i=1}^{N} \{(x_j^i, y_j^i)\}_{j=1}^{Q}. \tag{3.1}$$

Although this provides the recipe for few-shot task creations, a key ingredient is missing to create few-shot tasks from *existing* datasets. Simply taking a few samples from an existing dataset to construct such tasks, e.g., 'lion' and 'zebra', would allow the learner to learn a general classification learner. Moreover, this would disallow the learner to predict new classes, which a few-shot should be capable of. To address this, the samples' original labels are re-labeled such that $y_*^i \in [0, N)$[2]. Randomly assigning labels to classes in few-shot tasks ensures that a few-shot learner is incentivized to learn to generalize, as labels do not provide useful information across tasks.

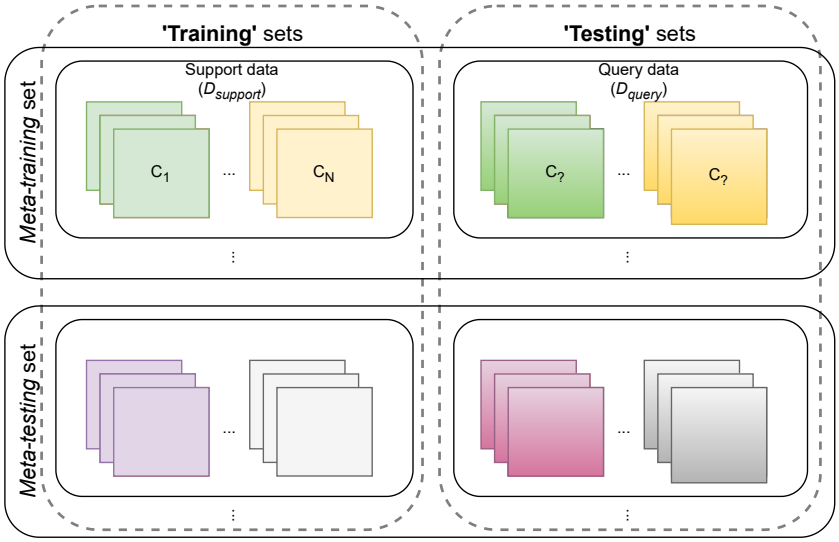


Figure 3.2: Meta-learning instance. The meta train set consists of a collection of few-shot instances, see Figure 3.1, characterized by a function $p : \mathcal{D} \mapsto \mathcal{T}$ The meta-test set is constructed similarly, so no samples co-occur in the meta-training and meta-test sets. The learner uses the support sets to 'generalize' to the query set during meta-testing. Hence only *meta-train*, and no *meta-test*, data is used for the final model.

We can construct few-shot tasks using existing datasets with this few-shot task recipe. For image classification tasks, as considered in this work, a task distribution p is defined, allowing a learner to sample from, i.e., $\mathcal{T} \sim p(\mathcal{D})$. Figure 3.2 shows a conceptual overview of the layout of few-shot learning datasets. Although not shown, samples in each data-split, e.g., meta-training split, may occur in multiple few-shot tasks. Conceptually, p achieves this by selecting N classes (ways) and taking $K + Q$ samples for each (shots)–both for the support and query set. Afterward, as discussed before, p remaps the labels to ensure that the tasks' labels leak no original label information. Otherwise, a few-shot learner could exploit the label information and learn a general classification model.

By providing different data splits, sampling p allows us to obtain tasks to train, test or validate few-shot learners. Each of these data splits is assumed to be disjoint. As a result, it allows us to evaluate a learner's ability to adapt to new *unseen classes* with few samples. For image classification tasks, this makes that combinatorially many few-shot tasks can be created.

[1] Or classes and samples, respectively.

[2] Generally, a (random) mapping function maps each sample to its new label. For example, in the three-way setting, $cat, dog, mouse$, such a mapping function could be; $cat \mapsto 2, dog \mapsto 0, mouse \mapsto 1$.

Remark 1

To prevent confusion with 'training' and 'testing' data offered by $\mathcal{D}_{\text{support}}$ and $\mathcal{D}_{\text{train}}$, and $\mathcal{D}_{\text{query}}$ and $\mathcal{D}_{\text{test}}$, we introduce terms to differentiate between the two. In Figure 3.2 provides a visualization for the proposed terms. We refer to the underlying data splits–i.e., train, validation, and test– as meta-training, meta-validation, and meta-testing data. For a specific task's 'train' and 'test' $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}})$ we refer to as training (support) or testing (query) data as shown in the dotted boxes in Figure 3.2.

3.1.1. Benchmarks

Image classification datasets are commonly used to evaluate few-shot learners. These vary in image size, number and granularity of categories, and the number of images per category, providing different levels of challenge for few-shot learning tasks. Here we provide a short overview of common benchmarks also utilized in this work.

Omniglot [29] provides a small-scale few-shot learning dataset that contains images of handwritten characters from 50 different alphabets. Each alphabet contains multiple characters and is provided with 20 images for each. All samples consist of a gray-scale image of 28x28 pixels.

CIFAR-FS [3] (CifarFS) is a variation of the CIFAR-100 dataset, where each category has only 600 images. As in CIFAR-100, the images are full-color with dimensions 32x32. For the split of the train, validation, and test data, we follow [3]'s suggested data splits.

FC100 [56] (Few-shot Classification with 100 Categories), similar to CIFAR-FS, is a CIFAR-100-based few-shot dataset. In its design, care is taken to minimize the semantic overlap between the data splits. As such, related classes are grouped in the validation, testing, and training split, thus posing a more challenging setting than CIFAR-FS.

Mini-ImageNet [54] is a simplified few-shot dataset based on the ImageNet [11] dataset, with 100 with 600 images per category. Each sample is a downsampled RGB image of 84x84 pixels, down from the original 224x224 pixels. Its larger input size and increased visual complexity make it a more challenging benchmark to adapt to.

3.1.2. Few-shot Learning Variations

Generally, few-shot learning assumes that clean data is provided, i.e., samples and their provided labels are correct. More recently, however, research has studied settings where this assumption is alleviated. We identified two main categories; few-shot learning with label corruption [34, 26, 6, 56, 38] and semi- and unsupervised few-shot learning [24, 1, 57, 32, 30]. In a noisy label setting, a fraction of the data provided to the learner is corrupted. Due to the construction of few-shot tasks, label noise generation can occur at two stages; the underlying data splits [6, 26], or at task-level [34, 56, 36]. Regardless of the type of label noise, only a few works consider meta-training with noise [56, 26, 6, 36]. Most of these works assume clean meta-training data is present, allowing them to provide ground-truth labels to learn a noise rejection mechanism. An exception is TADAM [56], which models label noise at task-level for a fraction of the tasks available tasks[3]. A limited number of works consider corrupted meta-training *datasets*, similar to this work. To the best of our knowledge, only Re-Weight-MAML [26] (RW-MAML) and Eigen Reptile [6] evaluate in this scenario. RW-MAML proposes a meta-objective to learn to filter noisy samples using a *clean* validation set. Eigen Reptiles proposes a robust meta-optimizer for Reptile [41] style learners by stepping towards the direction of the inner-optimization process with the highest variance.

[3]Hence, no image occurs in more than one task during training

Remark 2

Due to the construction of few-shot tasks from existing classification datasets, the label noise can occur at two levels; dataset level and task level. We refer to data(set) level noise if the *underlying datasplit*, i.e., training split, is corrupted. As such, a fraction (ϵ) in the original dataset gets its labels re-assigned and remains fixed afterward. We refer to task-level noise if label noise is (dynamically) injected into an initially uncorrupted task, as such samples may be corrupt in one task but correctly labeled in another. This latter noise approach is more suitable for meta-testing or learning a noise rejection mechanism–in which case the correct label is also provided to the learner. In this work, we generally perform training with *dataset level* noise and evaluate with *task level noise* (during experiments with label noise).

Semi-supervised [30, 32] and unsupervised learning [24, 21, 1] few-shot learning have shown promising results as approaches toward few-shot learning. These works assume that some or even all labels are unavailable. As a result, these works generally construct artificial tasks using unlabeled data. They then leverage augmentation techniques to create more 'shots' for the constructed tasks. Our proposed BatMan-CLR technique is related to these works in its sub-sampling technique and use of contrastive learning.

3.2. Gradient Based Meta-Learners

With the few-shot problem discussed, we continue with details of Gradient-Based Meta-Learning. First, we relate meta-learning of few-shot tasks to ordinary classification learning. Second, a more formal introduction of gradient-based meta-learners is given, leveraging a graphical perspective to help build intuition. Last, we introduce first-order meta-learners, which address common problems with exact (second-order) meta-learners.

Gradient-based meta-learners [15] approach few-shot learners by learning a model[4] that is trained to generalize with few samples. The idea is that a meta-learner learns the models' parameterization θ, such that it lays close to a good task-specific one. As such, few adaptation steps (and consequently samples) are required to adapt. To achieve this, the training objective is made to match that of meta-testing. First, let us recall the general objective of typical classification learners,

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \underset{(x,y)\in\mathcal{D}}{\mathbb{E}} \left[P_\theta(y|x)\right], \tag{3.2}$$

where P_θ expresses the probability of models parameterized by θ on predicting the label y given input x. Through this formulation, the training objective (Equation 3.2) reflects what needs to be done during evaluation, *predict unseen samples of known classes*. GMBLs, however, require the notion of tasks $\mathcal{T} = (\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}})$ and an adaptation process. Let us define $g_\phi(\theta, \mathcal{D}_{\text{support}})$ as the adaptation process from an initial model θ using a task's support data. Then the objective becomes,

$$\theta^* = \underset{\theta}{\operatorname{argmax}} \underset{(\mathcal{D}_{\text{support}},\mathcal{D}_{\text{query}})\in\mathcal{D}}{\mathbb{E}} \left[\underset{(x,y)\in\mathcal{D}_{\text{query}}}{\mathbb{E}} \left[P_{g_\phi(\theta,\mathcal{D}_{\text{support}})}(y|x)\right]\right]. \tag{3.3}$$

As a result, this objective imposes that the found meta-model θ^* should be capable of rapidly adapting. To achieve this without the task-adaptation process–rewriting Equation 3.3 to mimic the structure of Equation 3.2–requires the learner to have seen all tasks beforehand, defeating the purpose of few-shot learning altogether[5].

GMBLs use a two-stage approach to optimize their objective, i.e., to 'learn to learn'. Before introducing the formal definition, we provide the geometric interpretation in Figure 3.3, showing a GMBLs' meta-learning process in parameter space. At the interrupted point, we find meta-learner θ at meta-epoch t, where the inner-loop process $[\theta, \phi_1, \ldots, \phi_s$ is shown (thin solid black arrow). First, a tasks-specific model ϕ is adapted from θ using $\mathcal{D}_{\text{support}}$, i.e. $g_\phi(\theta, \mathcal{D}_{\text{support}})$ in Equation 3.3. Afterward, the gradient of ϕ_s on the query data is computed (purple dashed arrow) and back-propagated back to its start at θ

[4] Parameterized by a deep neural network.

[5] Although outside the scope of this work, multi-task learning can be seen as an in-between of few-shot learning and ordinary 'many'-shot learning.

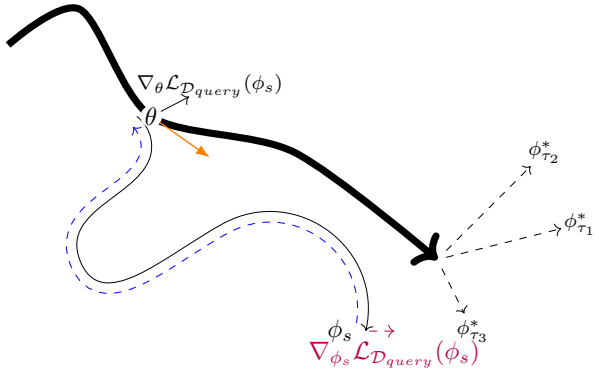


Figure 3.3: Geometric interpretation of MAML [15] in parameter-space. The bold (black) arrow shows the optimization path of the meta-model θ. We show the steps performed at each meta-epoch for a single train task, θ is adapted to task-specific ϕ_s are trained using support data (curved thin black arrow). Each ϕ_s query loss (purple) is related through the inner-adaptation process (dashed blue arrow) back to θ to calculate its meta-gradient (straight thin black arrow). A meta-batch consisting of multiple tasks (not shown) allows us better to estimate the meta-optimization direction (orange arrow). At the end of meta-training, the adaptation on meta-test tasks requires only a single adaptation step to find a good task-specific model ϕ^*_τ.

(blue dashed arrow). This operation then yields the tasks' meta-gradient (thin solid black arrow) for the next meta-iteration. Although only one task adaptation process at meta-epoch t is shown, generally a collection of tasks are jointly optimized at each step for more informative updates. Combining multiple updates from different tasks determines the meta-optimizer's next optimization direction (orange arrow), towards which the meta-optimizer steps. This meta-optimization process generally uses standard optimizers such as SGD or Adam [27]. The same steps are repeated to evaluate the model using meta-testing data, but drop the gradient descent through the gradient descent step (blue arrow), i.e., θ remains fixed after meta-training.

Let us formally introduce the optimization of these learners as proposed by Model Agnostic Meta-Learning [15] (MAML). First, a task-specific model optimized through s (stochastic) gradient descent (SGD) steps to find task-specific ϕ_s,

$$\vec{\phi}_{i+1} = \vec{\phi}_i + \alpha \nabla_{\vec{\theta}} \mathcal{L}(f_{\vec{\phi}_i}), \tag{3.4}$$

where $\phi_0 = \theta$, and $\mathcal{L}$ is a loss function, e.g., Cross Entropy loss. We drop the meta-epoch t in our notation here for convenience. To then optimize the meta-model θ, the error of tasks-specific models' on the query data is related to its initial parameterization,

$$\begin{aligned}
\theta_{t+1} &= \theta_t + \beta \nabla_\theta \mathcal{L}(\phi_s) \\
&= \theta_t + \beta \nabla_{\phi_s} \mathcal{L}(\phi_s)(\nabla_{\phi_{s-1}} \phi_s) \dots (\nabla_{\phi_0} \phi_1)(\nabla_\theta \theta) \\
&= \theta_t + \beta \nabla_{\phi_s} \mathcal{L}(\phi_s) \prod_{i=1}^{s} (\nabla_{\phi_{i-1}} \phi_i) \\
&= \theta_t + \beta \nabla_{\phi_s} \mathcal{L}(\phi_s) \prod_{i=1}^{s} (\nabla_{\phi_{i-1}} (\phi_{i-1} - \alpha \nabla_\theta \mathcal{L}(\phi_{i-1}))) \\
&= \theta_t + \beta \nabla_{\phi_s} \mathcal{L}(\phi_s) \prod_{i=1}^{s} (\mathbb{I} - \alpha \nabla_{\phi_{i-1}} \nabla_\theta \mathcal{L}(\phi_{i-1})).
\end{aligned} \tag{3.5}$$

This bi-level optimization scheme then requires the gradients and model parameters of each intermediate step ϕ_* to be retained to calculate meta-gradients. Thereby resulting that the meta-gradient computation growing linearly in space and time, as the query set loss from ϕ_s is related to θ_t requiring s Hessian vector computations.

Using this formal definition of these meta-learners, we provide the canonical pseudo-code that most implementations leverage. It models the task adaptation and meta-generalization through an inner (Equation 3.4) and outer-loop Algorithm 2.

Algorithm 1 Typical outer-loop of GBML, dropping meta-batches for convenience.

Require: β meta-learning rate.

Require: $p : \mathcal{D} \mapsto T$ Task generation

Require: $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$

θ is randomly initialized.

Perform meta-training

for $i \in [0, \dots, epochs)$ **do**

  $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}}) \sim p(\mathcal{D}_{\text{train}})$

  $\phi_i \leftarrow \texttt{InnerLoop}(\theta, \mathcal{D}_{\text{support}})$

  $\theta \leftarrow \theta - \beta \mathcal{L}_{\mathcal{D}_{\text{query}}}(\phi_i)$

Perform meta-testing

for $i \in [0, \dots, epochs)$ **do**

  $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}}) \sim p(\mathcal{D}_{\text{test}})$

  $\phi_i \leftarrow \texttt{InnerLoop}(\theta, \mathcal{D}_{\text{support}})$

  Evaluate on $\mathcal{D}_{\text{query}}$

Algorithm 2 Typical inner-loop structure of GMBL meta-learners.

Require: α adaptation learning rate.

Require: $\mathcal{D}_{\text{support}}$ task support set.

$\phi \leftarrow \theta$

Perform task adaptation

for $i \in [0, steps)$ **do**

  $\phi \leftarrow \phi - \alpha \nabla \mathcal{L}_{\mathcal{D}_{\text{support}}}(\phi)$

return ϕ

3.2.1. Approximating the outer-loop

As mentioned before, the exact computation of meta-gradients incurs significant memory and computational overhead as the number of inner-loop steps increases s. To address this, various approaches have been proposed to address this computational burden. Most of these works consider approximating the 2nd gradient calculation, i.e., the blue arrow in Figure 3.3. Thereby generally requiring only the calculation of 'first order' gradients, dropping the back-propagation through the inner loop.

The so-called first-order meta-learners learners aim to approximate the exact meta-gradient. A large section of work approaches this under the assumption that the direction of the meta-gradient is mainly influenced by the first-order component, i.e., $\nabla_{\vec{\phi}_s} \mathcal{L}(\vec{\phi}_s)$. Approaches such as first-order MAML [15] and sign-MAML [14] directly realize this by equating the hessian vector product (Equation 3.5) to the identity matrix $\mathbb{I}$ through an (implicit) stop-gradient[6] function. As a result, it eliminates the need for calculating higher-order meta-gradients. Reptile [41], on the other hand, approximates the meta-gradient using only support data. Contrary to other GMBLs, Reptile learners sub-sample the few-shot task in the inner-loop[7]. They effectively simulate slight variations of the provided few-shot task in each inner loop step. To approximate the meta-gradient, Reptile steps in the direction of ϕ_s[8]. Note that this thus drops the need for $\mathcal{D}_{\text{query}}$ during meta-training. Eigen Reptile [6] builds on this by decomposing the inner loops optimization path and stepping only in the direction with the highest variance. Implicit MAML [46] (iMAML) enforces that its meta-gradient can be exactly computed using only the final task-specific model ϕ_s. IMAML uses weight regularization and a longer inner-loop adaptation process to achieve this.

3.3. Representation Learning

Part of our proposed method BatMan-CLR–the Contrastive Learning (CLR) component–leverages representation[9] learning. The general idea is to learn a mapping of input (images) to a representation (embedding) such that similar inputs have similar representations. Contrastive learning is a type of representation learning for images, also used in our proposed BatMan-CLR. In this section, we provide additional information on contrastive learning.

Contrastive learning's goal is that visually similar samples (images) should be embedded to close points in the embedding space. This is generally achieved by using related inputs and unrelated inputs–positive and negative samples. Different formulations allow for contrastive objectives to be learned in supervised [25] as well as unsupervised/self-supervised [7, 49, 31] fashion. Especially the latter has

[6] A conceptual function, that detaches the computational graph, making that differentiating w.r.t θ *becomes equal to* differentiating w.r.t. ϕ_s.

[7] I.e., leveraging all samples in the support set for each inner-loop step.

[8] Equivalently setting the meta-gradient to $\theta - \phi_s$, as then the optimization step becomes $\theta' = \theta - \beta(\theta - \phi_s) = \theta + \beta(\phi_s - \theta)$.

[9] Or equivalently embedding learning, we will use these terms interchangeably.

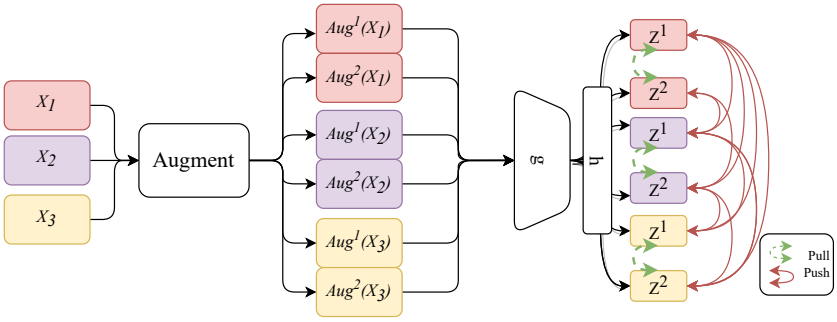


Figure 3.4: Pipeline for infoNCE [7]-style learning. First, samples (3) are individually augmented twice. The embedding learner-generated embeddings z_*, to 'pull' positives (green arrows) closer and 'push' negative samples (red arrows) further away. The abstract 'Augment' box represents the domain-specific implementation to generate unique augmentations. The embedding learner generally has an architecture that leverages a convolutional (g) and fully connected (h) component.

shown to be capable of rivaling or even exceeding supervised learners in classification settings [7, 58].

In this work, we make use of an unsupervised visual contrastive learning strategy. Different approaches have been proposed, ranging from requiring positive and negative views such as Bootstrap Your Own Latent [17] (BYOL), SimCLR [7], and Momentum Contrastive [19] (MoCo) learning, to ones that require only positive examples such as Simple Siamese (SimSam) Representation Learning [8]. In our evaluations of BatMan-CLR, we use a loss related to SimCLR [7]. By design, it allows for easy incorporation into the episodic style of learning leveraged by meta-learners, as it only requires samples, without needing a memory queue, or altered network architecture.

SimCLR [7] style learning leverages the NX-Xent (normalized contrastive temperature-scaled cross-entropy loss) loss or infoNCE (noise contrasting estimation) loss. Conceptually, SimCLR approaches the representation learning problem by 'pulling' positive samples closer in embedding space and 'pushing' negative samples further away. We show the general pipeline of this learning style in Figure 3.4. First, the original samples are augmented. Consecutively, each augmentation is embedded using a deep neural network $f = h \circ g$, consisting of a convolutional (g) and fully connected part h. Before we define the actual objective, we define what 1) negative and positive (samples) are and 2) a notion of distance (or similarity) to realize 'pulling' and 'pushing'. First, positives (and negatives) are generated through *augmenting* the original samples. Here *each sample* is thus assumed to be a unique class. By applying transformation–e.g., (color) distortion, shearing, cropping, etc.–on original images, variations can be generated[10], which are then treated as the learners' input. Augmentations from the same 'source' image are treated as positive examples (green arrows in Figure 3.4). In contrast, embeddings from all other samples are treated as negatives (red arrows Figure 3.4), forming $N_{aug}N(N-1)$ negative pairs with N_{aug} augmentations and N 'original' samples. Then to make that samples can be pulled or pushed, a metric is a function to express (dis)similarity. For this, infoNCE uses the cosine similarity on embedding vectors z_a, z_b computed by f,

$$s(z_a, z_b) = \frac{z_a^\top \cdot z_b}{||z_a|| ||z_b||}. \tag{3.6}$$

We will use the short-hand bra-ket notation $\langle z_a | z_b \rangle$ to denote this similarity–assuming embeddings are $L2$ normalized.

The infoNCE loss utilizes all pair-wise cosine-similarities between positive and negative samples[11]. We then write the objective from the perspective of each sample's first augmentation $z_i^{(1)}$, with a total

[10] Generally, each augmentation is generated independent of other augmentations
[11] Except embeddings' self-similarity, as embedding is generally *really* close to itself.

of 2 augmentations per sample,

$$\mathcal{L}_{\text{infoNCE}} = -\frac{1}{N}\sum_{i=1}^{N}\log\Big(\frac{\exp(\langle z_i^{(1)}|z_i^{(2)}\rangle)}{\sum_{j=1}^{2N}\mathbb{1}[i\neq j]\exp(\langle z_i^{(1)}|z_j\rangle)}\Big)$$
$$\mathcal{L}_{\text{infoNCE}} = -\frac{1}{N}\sum_{i=1}^{N}\log\Big(\frac{\exp(\langle z_i^{(1)}|z_i^{(2)}\rangle)}{\exp(\langle z_i^{(1)}|z_i^{(2)}\rangle)+\sum_{j=1}^{2N}\mathbb{1}[z_j\notin\{z_i^{(1)},z_i^{(2)}\}]\exp(\langle z_i^{(1)}|z_j\rangle)}\Big) \tag{3.7}$$

We drop the temperature scaling coefficient (τ) for notational convenience. Via symmetry, this can be extended to include the second augmentation by changing the order of augmentations. Similarly, adding a summation of positive views in the first term allows for incorporating additional positive augmentations. Generally, this loss is used with large batch sizes, allowing many negative pairs ($N_{aug}\cdot N(N-1)$) to be constructed.

Remark 3

Although this style of contrastive learning does not impose that images of the same class are embedded close in feature space, empirical evaluations show that this does happen [7]. However, this strongly depends on the type of augmentations used [7], which are domain-specific and introduce additional hyper-parameters. Different augmentations have been proposed on the input and the embedding level. In this work, we focus on standard input-level embeddings.

In our work, we leverage the Decoupled Contrastive Loss [58], an adaptation of the infoNCE loss. Rather than summing over all pair-wise similarities in the denominator, it proposes to take out $\langle z_i^{(1)}|z_i^{(2)}\rangle$). This change is motivated by the fact that the 'repel' force is negatively affected by $z_i^{(1)}$ when $z_i^{(2)}$ lies close [58], and vice versa. As a result, DCL performs better with smaller batch sizes, which is favorable in a few-shot scenario. The contrastive loss then becomes,

$$\mathcal{L}_{\text{DCL}} = -\frac{1}{N}\sum_{i=1}^{N}\log\Big(\frac{\exp(\langle z_i^{(1)}|z_i^{(2)}\rangle)}{\sum_{j=1}^{2N}\mathbb{1}[z_j\notin\{z_i^{(1)},z_i^{(2)}\}]\exp(\langle z_i^{(1)}|z_j\rangle)}\Big). \tag{3.8}$$

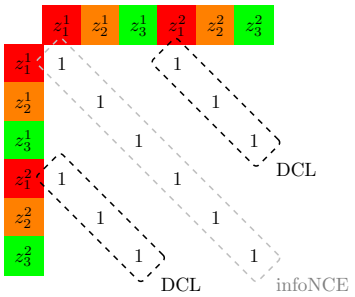


Figure 3.5: Example of the (inverted) selection masks for in infoNCE's [7] (Equation 3.7) and DCL's [58] (Equation 3.8) denominator. Each sample's i features $z_i^{(1,2)}$ are assigned the same 'class' represented by color. The top index z_*^j indicates the augmentation index corresponding to a sample's augmented embedding z_i^j. Note that DCL's mask consists of the union of the DCL and infoNCE. mask.

Remark 4

Generally, data is differently collated in actual implementations as represented in Figure 3.4. Rather than having different augmentations of the same image in consecutive indices, samples are in order of sample ($X_1, X_2, \dots$) and then in order of 'augmentation' ($A_1, A_2, \dots$). As such, selecting the samples can be easily done using masks to select positives and negatives for the computation. We provide a small visual example of how (the inverse) of such a selection mask looks like for computing the denominator of the infoNCE (Equation 3.7) and DCL (Equation 3.8) loss in Figure 3.5 in a (3,3)-FSL setting with two augmentations. Note that DCL's mask consists of the union of the DCL and infoNCE. This selection mask is then used to select values from a similarity matrix of feature embeddings. Note that the different embeddings corresponding to the original samples are in order of shot firsts and then ordered by the view.

4

nmfw Framework

We develop a framework for meta-learning under different few-shot settings: nmfw[1]. nmfw provides a set of *extensible* tools to implement meta-learning training loops efficiently. Moreover, it provides utilities to experiments in few-shot scenarios that currently are not supported by related works: (i) few-shot learning with train and test label noise, and (ii) unsupervised few-shot learning. One of the key features of nmfw is its decoupled implementation for meta-learners and the training loop, which makes it compatible with existing meta-learning libraries.

First, we present a common issue found in available meta-learning implementations, coined the 'coupling problem' Second, we provide an overview of existing meta-learning frameworks. Third, we introduce our frameworks' which address the coupling issue using a 'Learner' and 'Algorithm' abstraction. Forth, we detail nmfw's approach towards extensible few-shot data generation. Last, we perform a case study to highlight the benefit of nmfw, analyzing the run-time of meta-learners.

4.1. Learning: 'Algorithm' and Meta-'Learner'

This section presents a more detailed depiction of the pseudo-code for meta-learning algorithms, with additional details for improved understanding. We analyze its structure and note that its canonical form leads to a coupling between the meta-learner and training algorithm when implemented. Lastly, we propose a solution to this problem, which lays the foundation for the developed frameworks' design.

[1] Noisy Meta-learning Framework.

Algorithm 3 Typical outer-loop of GBML, with the added notion of meta-batches.

Require: β meta-learning rate.

Require: $p : \mathcal{D} \mapsto \mathcal{T}$ Task generation

Require: $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$

Require: B Meta-Batch size.

Require: T Evaluation size.

1: θ is randomly initialized.
2: *Perform meta-training*
3: **for** $i \in [0, \dots, epochs)$ **do**
4: **for** $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}}) \in [p(\mathcal{D}_{\text{train}})]_{i=1}^{B}$ **do**
5: $\phi_i \leftarrow \texttt{InnerLoop}(\theta, \mathcal{D}_{\text{support}})$
6: $\theta \leftarrow \theta - \frac{N}{B} \sum_{i=1}^{B} \nabla_\theta \mathcal{L}_{\mathcal{D}_{\text{query}}}(\phi_i)$
7: *Perform meta-testing*
8: **for** $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}}) \in [p(\mathcal{D}_{\text{train}})]_{i=1}^{T}$ **do**
9: $\phi_i \leftarrow \texttt{InnerLoop}(\theta, \mathcal{D}_{\text{support}})$
10: Evaluate on $\mathcal{D}_{\text{query}}$

Algorithm 4 Typical inner-loop structure of GMBL meta-learners, with (implicit) resampling.

Require: α adaptation learning rate.

Require: $steps$ Adaptation steps.

Require: $\mathcal{D}_{\text{support}}$ task support set.

Require: `ReSample` intra-task resampler.

1: $\phi \leftarrow \theta$
2: *Perform task adaptation*
3: **for** $i \in [0, steps)$ **do**
4: *Take a subset of samples in the inner loop, e.g., mini batching.*
5: $\mathcal{D}^{i}_{\text{support}} \leftarrow \texttt{ReSample}(\mathcal{D}_{\text{support}})$
6: $\phi \leftarrow \phi - \alpha \nabla \mathcal{L}_{\mathcal{D}^{i}_{\text{support}}}(\phi)$
7: **return** ϕ

Let us first re-print the canonical pseudo-code (Algorithm 3 and Algorithm 4). In lines 4-6 of the outer-loop (Algorithm 3), we introduce meta-batches[2]. Meta-batches, introduced in lines 4-6 of the outer loop, reduce noise in the estimated meta-gradient, similar to batching in Stochastic Gradient Descent. Additionally, resampling, also known as 'mini-batching' [41], in line 5 of the inner loop, plays a crucial role in Reptile-like learners. While this resampling step is often assumed to be implicit, we highlight its importance in the context of Reptile-like learners [41, 6].

This high-level depiction of the meta-learning loop provides an intuitive two-level abstraction, translating to a straightforward nested implementation. The outer-loop Algorithm 3 performs the meta-level generalization, and the inner loop Algorithm 4 performs the task-specific adaptation. However, when translating it to an actual implementation, the nested optimization handles that part of the meta-learners' responsibility in the outer loop. Specifically, the computation of meta-gradients (Line 6) becomes part of the outer loop–which conceptually should be *meta-learner* specific. On the meta-learners' side, data management of support and query data and sub-sampling[3] (Line 6) gets handled by the meta-learner. Hence, each meta-learner requires re-implementation of most of the training loop.

Ideally, a clear separation of concerns should exist, thereby decoupling the training loop and meta-learner. To address this coupling between the meta-learner and training algorithm, we propose a clear separation between the training 'Algorithm' and a meta-'Learner'. In this abstraction, the Algorithm should concern itself with the general structure of the inner- and outer loop, i.e., data management, validation, and testing. The Learner should provide interfaces to the Algorithm to perform the task-adaptation steps and outer-loop gradient calculations, separating their respective concerns. When using this decomposition, evaluating a learner in different settings only requires swapping the Algorithm or used dataset. For example, when moving from (clean) supervised learning to noisy testing or learning a meta-learner in an unsupervised fashion. Additionally, this allows for features of different meta-learners to become re-usable, e.g., sub-sampling [41] and regularization [46, 59, 23].

4.2. Meta-Learning Libraries

At the time of writing, a collection of related works are available. Although these provide (part of) the necessary tools to implement Gradient Based Meta Learners (GMBL) few-shot learners–training loop, `higher` order differentiation, and few-shot shot data generation–these lack desirable features for re-

[2] For the keen reader, we remark that generally line 6 in Algorithm 3 is computed directly in the loop of lines 4-5, using the meta-model itself to accumulate gradients. Consecutively, this allows for directly freeing a task's required memory after calculating its meta-gradient. However, we keep such optimization details out of the discussion for conceptual clarity.

[3] Most works [2, 33] load support and query data into a single data-tensor. As such, the meta-learner needs to know whether samples are support or query.

producible and comparative studies. Namely a *decoupled* implementation of meta-learner and training loop, and flexible few-shot data generation. We consider the works that leverage the PyTorch [44] Deep Learning Library. In no particular order; higher [16], Learn2Learn [2], TorchMeta [10], TorchOpt [48], EasyFSL, LibFewShot [33], and PaddleFSL [50].

In this section, we compare the different related frameworks for meta-learning. We conclude with common issues found in the reviewed libraries[4]; a lack of separation of concerns and no support for (extensible) few-shot data generation beyond supervised learning.

4.2.1. Overview

`higher` [16] provides a lightweight library with the tools to implement differentiable optimizers easily. By design, `higher` provides the core components to compute higher-order gradients for PyTorch [44] optimizers, such as SGD, Adam [27], RMSProp [20], etc., as well as user-provided optimizers. In turn, making the higher-order gradient calculations straightforward for second-order meta-learners like MAML [15]. This allows for straightforward implementation of meta-learning algorithms through its provided wrappers for existing as well as custom Torch optimizers[5] to become differentiable. This latter is needed, as by default, Torch's `autograd` frees memory after each optimization step[6]. Higher achieves this by monkey-patching[7] calls to PyTorch. In its approach, `higher` transforms the stateful Torch `Module`s–which provide concrete implementations of neural network (components)–into stateless ones, treating model parameters as first-class citizens. Combined with a context manager–which tracks the required gradients and intermediate parameters–it provides a lightweight wrapper to implement `higher`-order meta-learning algorithms.

TorchOpt [48] provides a collection of differentiable optimizers. Moreover, it provides a suite of tools to visualize computational graphs and perform distributed training. Similar to `higher`'s approach, this recasts the implementation of GMBL meta-learners as differentiable optimizers. However, they provide both a functional and an object-oriented interface, similar to PyTorch's object-oriented syntax. Additionally, it provides native support for distributed learning, allowing for learning at scale. It provides a collection of common differentiation types; exact, implicit, and zero-order differentiation. Similar to `higher`, it does not provide functionality to for few-shot data generation.

Learn2learn [2] is closest related to our framework. Similar to the previous works, it provides tools to implement meta-learning algorithms. Learn2Learn provides example implementations of seminal meta-learning works such as Reptile and MAML. Moreover, Learn2Learn provides few-shot benchmarks ranging from regression and classification tasks to reinforcement learning environments. In their examples, Learn2Learn leverages the inner- and outer-loop abstraction (Algorithm 4 and Algorithm 3). This results in a tight coupling between the meta-learners and the training loop, as discussed in section 4.1, requiring the re-implementation of most of the training loop for each meta-learner.

PaddleFSL [50] provides a PaddlePaddle [4] like implementation for FSL learning. It provides a collection of few-shot benchmarks and meta-learner implementations. However, each learner's implementation requires the implementation of most of the inner- and outer loops. Moreover, PaddleFSL requires a dataset-specific implementation before it can be used for few-shot data generation. This approach entails that each type of few-shot learning–such as unsupervised–would require a parallel implementation, limiting its extensibility to different few-shot learning settings. These latter two factors thereby limit PaddleFSL's value for experimenting in different settings outside those provided.

TorchMeta [10], contrary to the other works, focuses on providing commonly used datasets. In addition, it provides a wrapper to implement MAML, which currently supports a subset of PyTorch's `Module`'s. Although TorchMeta provides a large collection of few-shot datasets, it requires a custom wrapper per dataset. As such, it is less suited for explorative research with new data, as we will discuss in section 4.4.

EasyFSL[8] and LibFewShot [33] are two frameworks more tailored towards the more general few-shot learning, contrary to other related works. While EasyFSL does not provide meta-learner reference

[4] And by extension, reference implementations that leverage these.

[5] Optimizers in PyTorch handle the application of gradients of parameters in an in-place fashion, which makes that each optimization `step` overwrites a models parameterization.

[6] Generally, this is beneficial as otherwise the cached data–gradients and activations–would require explicit managing after taking an optimization step.

[7] Monkey-patches dynamically change the behavior, allowing to capture and alter calls to specific (library) functions. As Python is a dynamic language, this allows for adding functionality using flexible 'hooks' that implement the required functionality.

[8] https://github.com/sicara/easy-few-shot-learning

implementations, LibFewShot provides a complete set of seminal meta-learners. However, these works offer a relatively restrictive pipeline to create few-shot datasets. We note that LibFewShot uses a conceptually similar architecture design as nmfw, as we will discuss in section 4.3, it does not decouple the meta-learners' implementation from the general training loop.

4.2.2. Remarks

To the best of our knowledge, these works lack support outside supervised few-shot learning. Hence, exploring different settings–e.g., with label-noise, semi- or unsupervised, etc.–requires custom implementations, making a fair comparison between related works difficult. Indeed we found that works that provide public implementations [24, 21, 34, 6] all utilize custom data-loading. As a result, this makes a fair comparison between these works more difficult. Addressing this through a flexible and extensible data-loading pipeline would improve the reproducibility of future label noisy and semi/unsupervised meta-learning research.

Generally, the examples provided by works show a tight coupling between the learners and their respective training loops. Conceptually, each meta-learner defines *how to adapt given support or query data*, requiring it to have little knowledge of 'the rest' of the training loop. We hypothesize that this originates from the direct translation of the canonical inner and outer loop to code, as discussed in section 4.1. Moreover, we found the used data loaders were often inflexible, without support for label noisy or unsupervised few-shot learning. Making it less straightforward to make fair comparisons between different works in similar settings, as they must each implement this functionality differently. For example, small details in the noise generation make it difficult to compare such results one to one. Combined, these two points that implementations provided by libraries and reference implementations are difficult to compare.

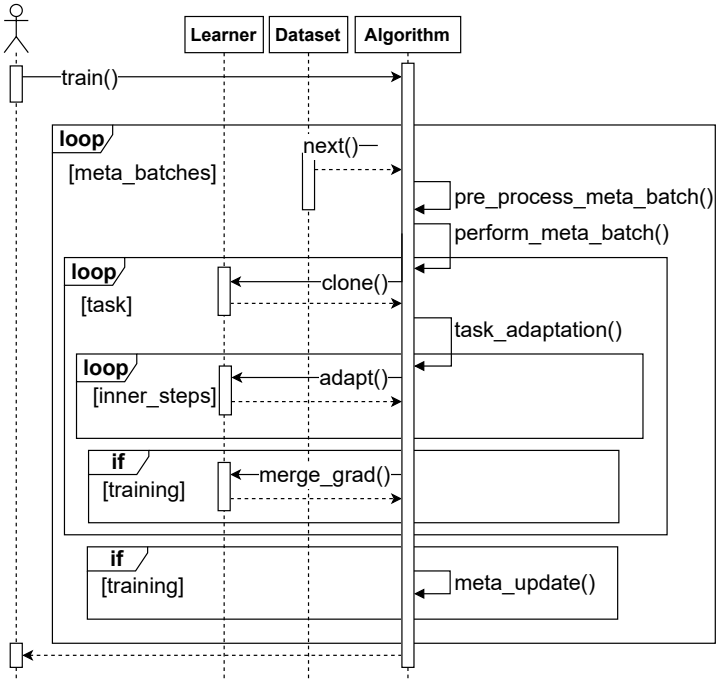


Figure 4.1: High-level sequence diagram of nmfw's core components and interaction for meta-training. Additional details such as optimizers, validation, and testing are kept out in favor of conceptual clarity. Note that we use the syntax as is provided by our framework, which uses this abstraction.

4.3. Framework Architecture

This section reviews the design details of the developed nmfw (subsection 4.2.2). It aims to provide an extensible basis for upcoming research directions, such as unsupervised [21, 1, 57] and noisy few-shot learning [34, 38, 6].

This section is structured as follows. First, we provide a high-level overview of the meta-learners by

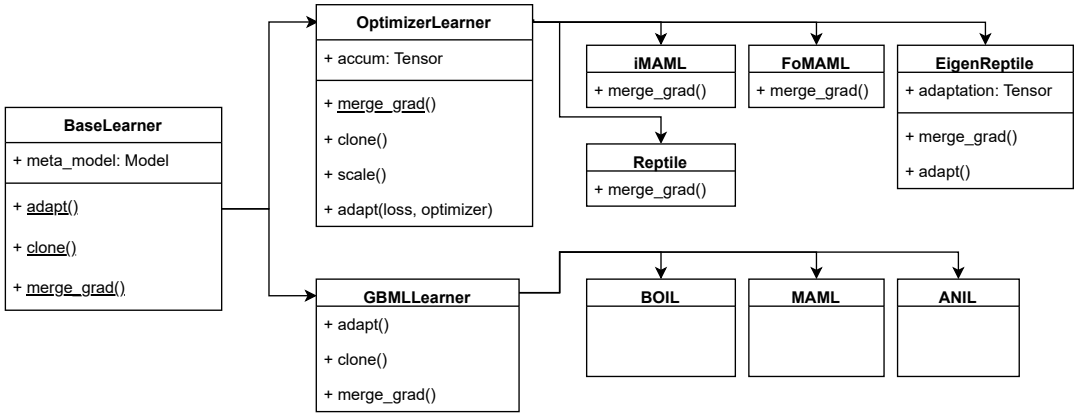


Figure 4.2: Overview of `nmfw`'s Learner design.

showing the interaction of the 'Learner' and 'Algorithm' components. Second, we highlight the 'Learner' component, which encapsulates the meta-learners' responsibilities, and provides interfaces for the 'Algorithm'. Third, we provide the 'Algorithm' component design, providing the general 'training loop'. Last, we motive how `nmfw`'s design–leveraging the Learner and Algorithm components–addresses the coupling problem as sketched before.

4.3.1. A birds-eye view

Before we discuss the more specific details of `nmfw`, we provide an overview of `nmfw`'s proposed training loop in Figure 4.1. This provides insight into how `nmfw` structures the conceptual pseudo-code in its implementation. Although Figure 4.1 depicts the overview for meta-training, meta-testing, and validation follow a similar structure[9]. After training starts, the Algorithm directs the overall training process and manages the data. It then delegates meta-gradient and adaptation calculations to the Learner through its `clone`, `adapt`, and `merge_grad` interfaces. From a high level, three nested loops are present from the outer- to the inner-most level the meta-epoch level, the task level, and the inner-loop level. The outer-most loop handles the general training loop, tracking and logging statistics, saving checkpoints, etc., and interacts with the Dataset to get meta-batches. The Algorithm pre-processes the meta-batch and processes the tasks sequentially. The adaptation process starts by requesting a task-specific copy of the Learner through a call to `clone`. Upon return, the Algorithm computes the adaptation loss (not shown) and requests the task-specific Learner to `adapt` given the loss. Upon completing the inner-loop adaptation, the Algorithm requests the Learner to compute the meta-gradient via `merge_grad`. Note that the *meta-model* performs this last action using the adapted model and its loss on the query set.

4.3.2. Learner

Starting with the 'Learner' abstraction, we expand on the structure proposed by Learn2Learn [2]. We provide a graphical overview of its structure in Figure 4.2. Learn2Learn proposes the usage of `adapt` and `clone` methods. This former function provides an inner-loop step interface, i.e., at Line 6. The latter, `clone`, provides an interface to create a task-specific learner from the meta-learned model. Thereby responsible for handling how the task-specific meta-learners' computational graph should be constructed. For exact meta-learners, this is important, as a complete computational graph is required to calculate the meta-gradient. We extend this with the `merge_grad` method. This last method provides a transparent interface to calculate and apply the meta-gradients to the meta-learner[10].

As shown in Figure 4.2, we utilize two categories for the provided meta-learners; `GBMLLearner` and `OptimizerLearner`. This former provides an implementation leveraging Learn2Learn's meta-learners for exact 2nd order GBML learners[11], e.g., BOIL [42], ANIL [45], and MAML [15]. The `Opti-`

[9] Generally, while testing the meta-batch size is fixed to `1`, and the conditional blocks on `training` are skipped.

[10] We note that the `scale_grad` of the 'Learner' appears superfluous. However, separating the scaling from the calculation of the meta-gradient–with the meta-batch size–makes the 'Learner' only requires a single during adaptation.

[11] These are provided as a starting point for experiments and a baseline for implementations.

mizerLearner allows for the generic implementation of meta-learners through ordinary optimizers. Moreover, this allows for implementations leveraging other meta-learning frameworks' optimizers to be incorporated. For example, integrating an exact meta-learner using higher's differentiable optimizer is achieved through extending OptimizerLearner to handle interacting with higher components[12]

Although we currently provide implementations for four first-order algorithms: iMAML [46], Reptile [41], first-order MAML [15], and Eigen Reptile [6], these can be used as guides to extend OptimizerLearner to incorporate different meta-learners. Opening up the implementation to the public allows for extension through collaboration.

4.3.3. Algorithm

We continue with the Algorithm component of nmfw. This module handles the general 'learning loop', i.e., data management, validation, and testing. Figure 4.3 shows the currently provided Algorithms provided by nmfw. Here AbstractAlgorithm provides a general interface, and BaseAlgorithm provides a general training loop for supervised training and training with label noise.

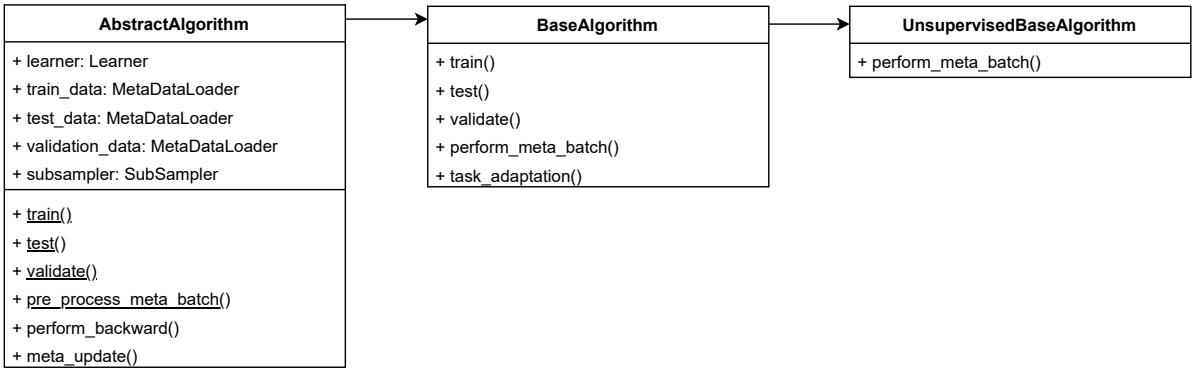


Figure 4.3: High-level overview of Algorithm abstraction of nmfw.

To allow for an extensible approach, we decompose the canonical algorithm into four steps; pre_process_meta_batch, perform_meta_batch (line 4-6 in Algorithm 4), task_adaptation Algorithm 4, perform_backward and meta_update (line 6 in Algorithm 4). This achieves that the logic for data-management (pre_process_meta_batch), can be seperated from the overall loop (perform_meta_batch, task_adaptation, from deferring calls to the meta-learners (perform_backward and meta_update). Moreover, pre_process_meta_batch allows a straightforward way to extend nmfw to incorporate different styles of few-shot learning. For example, for unsupervised meta-learning, rather than rewriting the entire training loop, only the pre_process_meta_batch required re-definition to organize features (inputs) appropriately.

4.4. Few-Shot Data Generation

We conclude this chapter with the design motivations of nmfw's data-loading utilities. Data loading for (exploratory) research into few-shot learning is paramount. For these experiments, this should both be flexible and reproducible–to easily explore new directions and validate new ideas. First, we compare two commonly applied major libraries used in meta-learning implementations; inflexibility to different few-shot settings. Our implementation provides a flexible set of tools to implement different few-shot learning experiments quickly.

4.4.1. Related Work

Regarding few-shot data loading, we found that two general approaches are taken to construct tasks; using wrapper or indices. Here we use TorchMeta's and Learn2Learn's syntax as representatives for these methods. In Algorithm 5 and Algorithm 6, we provide a Python-like pseudocode showing their data managing approaches using TorchMeta and Learn2Learn.

[12]This would consist of making a call to higher.innerloop_ctx in the Learner's clone method.

Algorithm 5 Dataloading example in TorchMeta-style.

Require: N number of 'ways' (classes)
Require: K number of support 'shots' (samples)
Require: Q number of query 'shots'

```
# Get dataset-specific 'wrapper' to create ways.
dataset = DatasetWrapper(N)
# Create (N, K)-FSL dataset.
datset = ClassSplitter(dataset, N, K, Q)
```

Algorithm 6 Dataloading example in Learn2Learn style.

Require: N number of 'ways' (classes)
Require: K number of support 'shots' (samples)
Require: Q number of query 'shots'

```
# Using any Torch-compatible dataset.
dataset = MetaDataset(Dataset())
transforms = [
    NWays(dataset, N), # Define ways.
    KShots(dataset, K), # Define shots.
    LoadData(dataset),
    ... # Augmentations, etc.
    ]
# Create (N, K)-FSL dataset.
taskset = TaskDataset(dataset, transforms)
```

Wrapper based approaches (Algorithm 5, using TorchMeta as an example) leverage a straightforward approach for task generation. Most few-shot frameworks use this strategy to generate few-shot tasks. The first step requires a dataset-specific wrapper that organizes the dataset's data, providing splits based on sample data. The second stage leverages these splits to draw support and query samples, realized in TorchMeta through the `ClassSplitter` module. Although this lightweight approach allows for fast data generation, it requires a definition of a `Wrapper` for each new dataset. As a result, a `DatasetWrapper` implementation *per dataset* for each few-shot learning setting needs to be provided–e.g., training or with label noise, or unsupervised meta-learning. Moreover, due to its coarse control over the data flow during task creation, it is less suited for an extension to different types of noisy few-shot learning. For example, to implement with out-of-task noise[13] would require an additional wrapper per noise type.

Index-based approaches, on the other hand, make use of two indices to find samples per class and an inverse mapping. To our knowledge, Learn2Learn is the only related library that leverages this approach. Although this introduces additional overhead–to compute and store the indices–this is a one-time cost and allows for fine-grained data control during the generation of tasks. Learn2Learn leverages these lookup tables in its `TaskTransforms`, which implement parts of the few-shot task data pipeline. As shown in Algorithm 6, multiple transforms are combined to define the creation of few-shot data generation. By design, each of these transforms allows direct access to the indices through the provided `MetaDataset` object. Under the hood, Learn2Learn leverages a symbolic `DataDescription` object, which represents the different 'shot' and 'samples' until the actual data is loaded.

4.4.2. Flexible (label noisy) Few-Shot Data

Motivated by Learn2Learn's fine-grained control, `nmfw` extends its functionality to support different types of few-shot learning settings. It provides necessary utilities to generate noisy `MetaDataset`s as well as corrupted tasks. Moreover, `nmfw` provides a set of `Transforms` that allow for experimenting with unsupervised few-shot learning. Like Learn2Learn, `nmfw` achieves this through task and data set level transformation functions, making few shot task generation flexible and extensible.

First, we discuss the design considerations for meta-learning with noisy label datasets, i.e., few-shot learning with a corrupted dataset label noise. Last, we detail the task-level utilities provided by `nmfw`, providing an extensible pipeline for few-shot learning and testing with corrupted *tasks* and unsupervised task generation.

Training with noise: `NoisyMetaDataset`

We extend Learn2Learn's `MetaDataset` through a `NoisyMetaDataset` class to perform training and testing with label noise. This 'noisy' counterpart of `MetaDataset` provides a generic interface to create a noisy dataset of any `MetaDataset`–thereby making it compatible with every Torch-compatible

[13] Where samples are replaced in a task $\mathcal{T}$ with samples from classes outside of $\mathcal{T}$

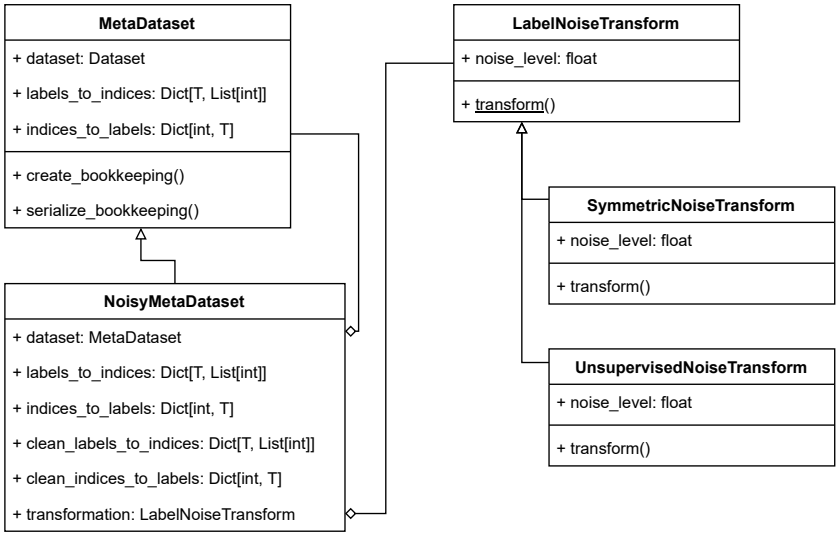


Figure 4.4: Overview of core components provided for training on noisy few-shot datasets through decorating Learn2Learn's `MetaDataset`, as provided in `nmfw`.

dataset. An `LabelNoiseTransform` performs the label corruption by providing corrupted indices to the `NoisyMetaDataset`. Its design leverages the decorator design pattern, which makes the corrupted indices transparently available to existing `TaskTransform`, making it a drop-in replacement for `MetaDataset`s.

Figure 4.4 provides an overview of the `NoisyMetaDataset`'s core components. The `NoisyMetaDataset` creates updated indices so that the `NoisyMetaDataset` can provide samples with corrupted labels. These 'noisy' indices are concrete implementations of the `LabelNoiseTransform`, which acts as a wrapper for `MetaDataset`s to inject noise.

Currently, `nmfw` provides a concrete implementation for symmetric dataset level noise through a user-provided `LabelNoiseTransform`. In addition, we provide a `UnsupervisedNoiseTransform`, which acts as a basis for unsupervised few-shot learning experiments. Unlike its supervised counterparts, it randomly selects N samples regardless of their labels–as these are assumed to be unavailable. This latter transform[14], provides support for unsupervised few-shot learning settings. Thereby re-casting the unsupervised few-shot learning setting as *noisy label learning with dynamic noise*. We plan on adding more common noise types `nmfw` after its initial release.

Augmenting Tasks

To complete the data-generation pipeline, `nmfw` provides a set of `TaskTransform`s. When combined with the appropriate `(Noisy)MetaDataset`, varied few-shot learning settings are quickly realized. Figure 4.5 provides an overview of the introduced transformations, grouped into three categories; dynamic noise generation, noisy label management, and input augmentation transform. Here we will set the three categories of transformations apart.

Dynamic noise generation introduces the support for generating noise at a per-task instance. `NoisyFusedNWayKshotTransform` provides this functionality which levereges `SamplerTransform` that corrupts labels (or samples). This approach provides a basis for meta-training with supervised noise[15] and meta-testing under different label noise scenarios.

From a high level, the `NoisyFusedNWayKshotTransform` samples a clean few-shot task and selects which samples to corrupt. The fused transform provides the indices to corrupt to the `SamplerTransform`, which implements a concrete type of label noise–similar to the `LabelNoiseTransform`.

As the specific noise provided by the `SamplerTransform` may re-assign labels to samples *per task*, a notion of 'intended' class (or way) is needed. Hence, the `NoisyFusedNWayKShot-`

[14] Combined with the appropriate task-level transforms, as will be discussed next
[15] Where the learner has access to dirty and clean label information

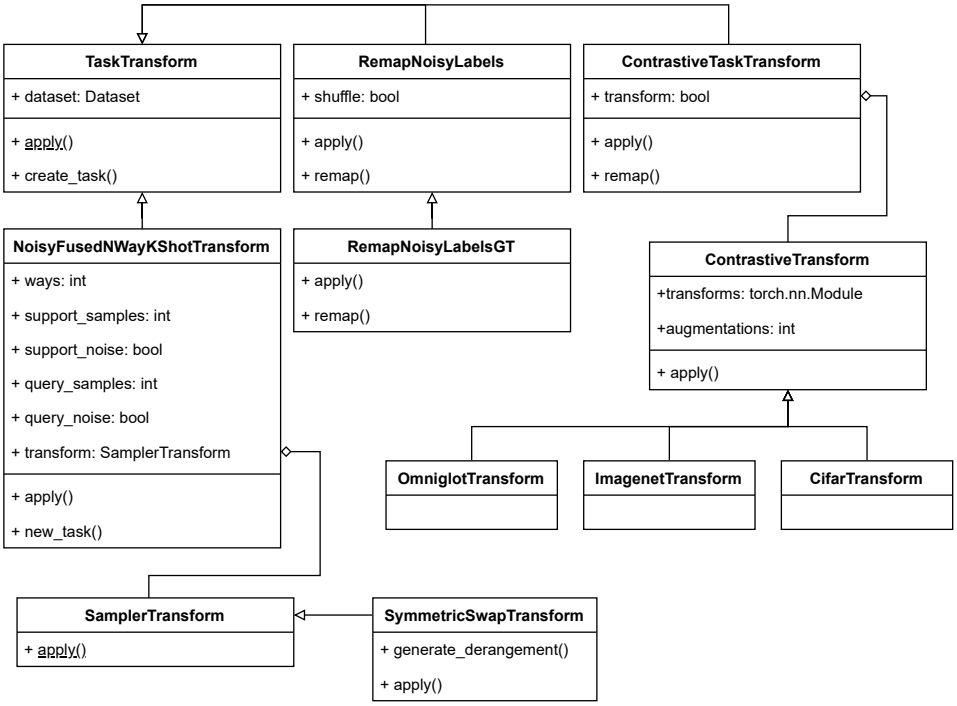


Figure 4.5: Schematic overview of task level transformations provided by `nmfw`.

`Transform` uses a `NoisyDataDescription` (not shown)–rather than Learn2Learn's `DataDescription`. Through this data object, the task generation process can keep track of each selected sample's 'intended' class, thus ensuring that noisy and ground truth labels are correct.

Noisy label management handles the assignment of correct labels to clean and noisy samples, providing a mapping from sample indices to their intended and ground-truth label. For experiments leveraging a `NoisyMetaDataset`, `RemapNoisyLabels` provides using the `NoisyMetaDataset`s' indices. Experiments with dynamic noise, i.e., generated at task level with `NoisyFusedNWayKshotTransform`, are required to use the `RemapNoisyLabelGT` transform. `RemapNoisyLabelGT` uses a `NoisyDataDescription` to provide a shots' index and its intended class. Thereby, `SamplerTransforms` can dynamically re-assign class labels to selected shots while remaining compatible with existing transformations.

Lastly, **Input augmentation** transformations, for which `nmfw` currently provides a `ContrastiveTaskTransform` for few-shot learning with contrastive losses and unsupervised meta-learning. As shown in Figure 4.5, the `ContrastiveTaskTransform` uses a `ContrastiveTransform` which provides the augmentation implementation. `nmfw` provides a set of such contrastive transformations for CIFAR, ImageNet, and Omniglot-based datasets. For the RBG image-based datasets, we provide standard [7] using rescale cropping, color jittering, grayscale, and horizontal flips. Omniglot contains grayscale images of hand-written characters, so we provide augmentations following [5]. These transformations use standard Torch Vision [37] transformations, but the `ContrsativeTaskTransform` supports any Torch-compatible module.

Remark 5

To provide users with a way to get started with these tools, `nmfw` provides high-level utilities to instantiate standard benchmarks. These make for easy creation of datasets for meta-training and testing in unsupervised and labeled noisy few-shot settings. Currently, `nmfw` provides such interfaces for Omniglot, FC100, CifarFS, and MiniImagenet, with more planned for future releases. Moreover, we provide example implementations of unsupervised/noisy few-shot meta-learning and meta-testing in different configurations.

4.5. Performance evaluation

As a small case study of nmfw's utility, we perform a small performance evaluation experiment. We compare the impact of different experiment configurations on the meta-epoch train, indicative of a meta-learners' training time. We consider the impact of the number of shots and inner-loop adaptation steps for two deep learning models commonly used in few-shot learning.

Table 4.1: Experiment parameters and levels considered in performance evaluation on MiniImagenet [54].

Parameter	Levels
Meta-Learner	iMAML, foMAML, Reptile, Eigen Reptile
Shots	5, 10, 15
Inner loop steps	5, 15, 25
Models	Conv4 [47], ResNet12 [18]

We summarize the considered levels of the evaluated parameters in Table 4.1. Four meta-learners are considered, iMAML [46], foMAML [15], Reptile [41], and Eigen Reptile [6]. Note that we set the mini batching parameter for Reptile and Eigen Reptile 'mini batching' to the number of shots in an experiment. For all experiments, we fix the meta-batch size to 5. After training each meta-learner for 100 meta-epochs, we report the average meta-epoch time from three runs. Three threads were used for data-loading, with a `pre_fetch` factor of 15 (meta-batches). We evaluate the experiments on a system equipped with an AMD Ryzen 5900X (12c/24t), 32GB memory, and an Nvidia RTX 3090 (24 GB VRAM).

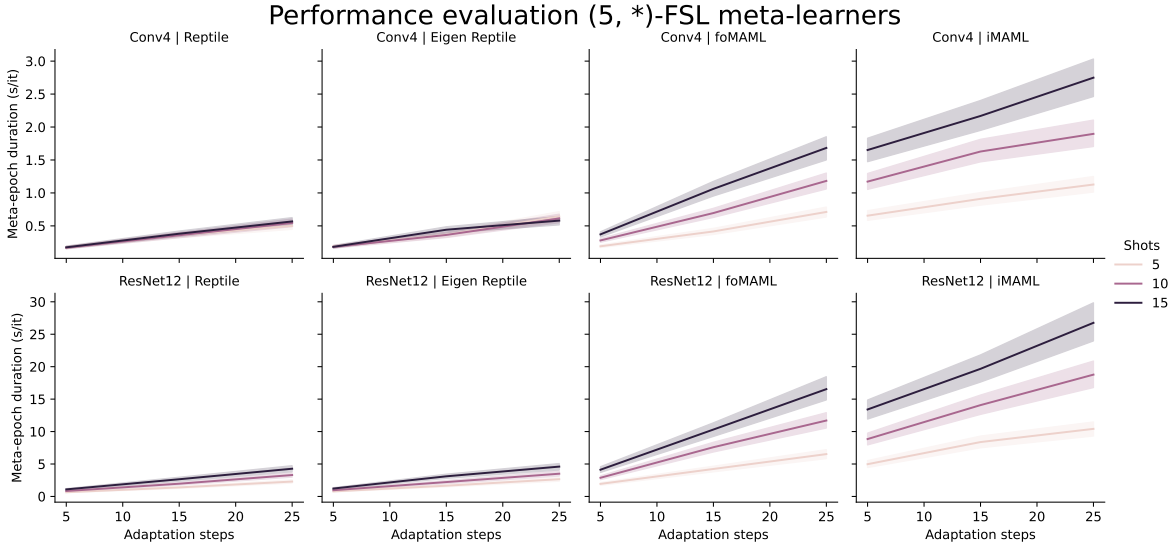


Figure 4.6: Graph showing the increase in meta-epoch duration per iteration (y-axis) under different configurations of inner-loop adaptation steps (x-axis) and the number of shots (hue). Shaded areas show 95th confidence intervals. Evaluated on MiniImageNet [54] with a Conv4 [53] and ResNet12 [18] architectures. The Meta epoch duration (y-axis) is scaled *per row*.

Figure 4.6 shows the results from the performance evaluation study. We see a stark difference between the Reptile and MAML style learners across both models. Due to Reptile learners' subsampling strategy, these learners require considerably fewer computations per adaptation step. foMAML and iMAML show a steeper incline as the number of shots increases, as these learners use all samples for each inner-loop step. Although iMAML and foMAML have the same inner-loop complexity, the *offsets* between the curves are significantly larger with iMAML. This results from the Conjugate Gradient method used by iMAML to compute its meta-gradient, which is a computationally heavy operation.

Comparing the average meta-epoch time between the Conv4 and ResNet12 model (the two rows in Figure 4.6), we note that Resnet12 experiments have considerably longer meta-epoch times. Looking at the different architectures of the models, we see that the ResNet12 architecture is much more complex than the Conv4 one. Where the Conv4 model has $121,093$ trainable parameters with 97.40 million

multiplication-addition (mult-add) operations, ResNet12 has $12,427,525$ parameters with 3.48 billion mult-add operations. This 10-fold increase in parameters and a 30-fold increase in mult-add explains the larger differences between the Conv4 and ResNet12 models.

5

Additional results

Here we provide additional results obtained during the work outside those presented in the research paper in chapter 2.

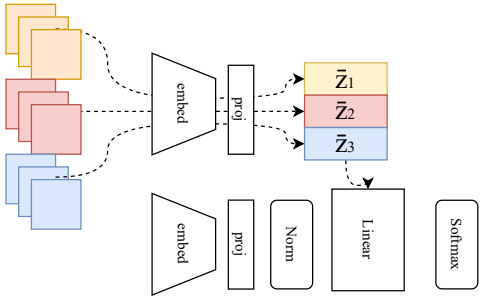


Figure 5.1: Proposed Meta-Baseline [9] style classification learner. Before adaptation, each classes i's (normalized) centroids are computed $\bar{z}_i$ (Equation 5.1). Consecutively these centroids are used to initialize an additional fully-connected layer. The new layer is placed in between a Normalization and Softmax non-linearity. This constructs a (softmax) cosine-similarity classification output per-class probability scores (Equation 5.2).

5.1. Constructing Classification Model

Generally, the zero-ing trick was used to create a classification for meta-learners trained with BatMan-CLR. We also perform meta-testing with a classification model adapted from Meta Baseline [9]. A visualization in Figure 5.1 shows how such a classifier is constructed from a pre-trained model on a (3, 3)-FSL task support set. Rather than appending the model with a zero-ed out layer, the model appended with a module consisting of a normalization, fully connected, and softmax layer. The fully-connected layer is initialized before adaptation with a support sets' normalized centroids, i.e.,

$$\bar{z}_i = normalize(\frac{1}{K}\sum_{i=j}^{K} h(f(X_{i,j}))), \tag{5.1}$$

with $X_{i,j}$ being class i's j^{th} support shot, assuming that each class is provided with K shots. Here f represents the embed(ing) layer and h the proj(ection) layer as depicted in Figure 5.1, i.e., the convolutional and fully connected part of the BatMan-CLR trained model. Using the added module, the model is mapped to a cosine-similarity classification model, between embeddings and class centroids. By construction, this implements a (softmax scaled) cosine similarity-based classification, as a samples similarity with a class i's centroid $\bar{z}_i$ is expressed as,

$$s_{\cos}(\bar{z}_i, z_{i,j}) = \langle \bar{z}_i | z_{i,j} \rangle = \frac{\bar{z}_i^{\top} z_{i,j}}{||z_{i,j}||} = \bar{z}_i^{\top} normalize(h(f(X_{i,j})). \tag{5.2}$$

As such, each of the classifiers' outputs represents the (softmax scaled) embeddings' similarity with a specific different few-shot class centroid. Contrary to [9], we only set the weights of the classification layer once, after which the learner's inner optimizer optimizes it. The construction of this classifier was made to attempt to maximize the objective between the contrastive meta-training and supervised meta-testing and evaluate the meta-learned model's ability to adapt its embeddings.

Table 5.1: (clean) Meta-test accuracy with 95th Confidence Interval ($\overline{acc}_{\pm \text{CI95}}$) of Supervised meta-learners on (5, 5)-FSL Omniglot and CifarFS with different meta-training noise levels (ϵ). Comparing the performance of Meta-Baseline [9] (MB) and (reprinted) zeroing-trick [23] (ZO) performance with BatMan-CLR meta-trained models.

		$\epsilon = 0.0$		$\epsilon = 0.3$		$\epsilon = 0.6$	
Alg.	Sampler	MB	ZO	MB	ZO	MB	ZO
Reptile	BatMan	55.3$_{\pm 0.174}$	66.5$_{\pm 0.168}$	54.8$_{\pm 0.169}$	65.0$_{\pm 0.170}$	54.9$_{\pm 0.174}$	64.1$_{\pm 0.170}$
	Man	44.8$_{\pm 0.175}$	61.8$_{\pm 0.176}$	44.7$_{\pm 0.168}$	62.0$_{\pm 0.175}$	44.6$_{\pm 0.173}$	61.4$_{\pm 0.173}$
Eigen Reptile	BatMan	55.5$_{\pm 0.181}$	66.3$_{\pm 0.171}$	55.1$_{\pm 0.175}$	64.4$_{\pm 0.170}$	55.2$_{\pm 0.174}$	63.8$_{\pm 0.176}$
	Man	49.5$_{\pm 0.171}$	61.7$_{\pm 0.170}$	49.5$_{\pm 0.173}$	55.8$_{\pm 0.378}$	49.3$_{\pm 0.169}$	52.3$_{\pm 0.365}$
foMAML	BatMan	20.0$_{\pm 0.142}$	66.6$_{\pm 0.167}$	20.0$_{\pm 0.130}$	65.2$_{\pm 0.169}$	20.1$_{\pm 0.123}$	64.8$_{\pm 0.164}$
	Man	20.2$_{\pm 0.130}$	66.2$_{\pm 0.164}$	20.1$_{\pm 0.130}$	64.8$_{\pm 0.165}$	20.1$_{\pm 0.130}$	64.0$_{\pm 0.165}$
iMAML	BatMan	21.3$_{\pm 0.142}$	64.2$_{\pm 0.185}$	21.4$_{\pm 0.138}$	62.7$_{\pm 0.250}$	21.3$_{\pm 0.123}$	62.9$_{\pm 0.203}$
	Man	21.1$_{\pm 0.139}$	62.8$_{\pm 0.172}$	21.1$_{\pm 0.137}$	62.6$_{\pm 0.168}$	21.0$_{\pm 0.127}$	61.7$_{\pm 0.169}$

(a) CifarFS results.

		$\epsilon = 0.0$		$\epsilon = 0.3$		$\epsilon = 0.6$	
Alg.	Sampler	MB	ZO	MB	ZO	MB	ZO
Reptile	BatMan	89.0$_{\pm 0.195}$	97.9$_{\pm 0.070}$	89.3$_{\pm 0.188}$	97.3$_{\pm 0.078}$	86.8$_{\pm 0.218}$	96.2$_{\pm 0.100}$
	Man	84.4$_{\pm 0.226}$	97.8$_{\pm 0.068}$	84.4$_{\pm 0.219}$	97.8$_{\pm 0.068}$	83.6$_{\pm 0.235}$	97.7$_{\pm 0.070}$
Eigen Reptile	BatMan	66.8$_{\pm 0.332}$	92.6$_{\pm 0.128}$	66.7$_{\pm 0.326}$	93.0$_{\pm 0.119}$	66.2$_{\pm 0.332}$	93.2$_{\pm 0.117}$
	Man	53.4$_{\pm 0.390}$	93.7$_{\pm 0.116}$	52.5$_{\pm 0.370}$	93.9$_{\pm 0.114}$	52.3$_{\pm 0.377}$	94.0$_{\pm 0.111}$
foMAML	BatMan	20.1$_{\pm 0.293}$	98.2$_{\pm 0.066}$	20.2$_{\pm 0.264}$	98.2$_{\pm 0.063}$	20.2$_{\pm 0.273}$	98.0$_{\pm 0.067}$
	Man	20.3$_{\pm 0.298}$	98.1$_{\pm 0.062}$	20.2$_{\pm 0.224}$	98.1$_{\pm 0.062}$	20.0$_{\pm 0.256}$	98.1$_{\pm 0.061}$
iMAML	BatMan	20.7$_{\pm 0.315}$	97.5$_{\pm 0.078}$	20.7$_{\pm 0.294}$	98.1$_{\pm 0.069}$	20.7$_{\pm 0.277}$	98.3$_{\pm 0.063}$
	Man	20.6$_{\pm 0.312}$	97.8$_{\pm 0.076}$	20.4$_{\pm 0.234}$	98.1$_{\pm 0.061}$	20.8$_{\pm 0.285}$	98.2$_{\pm 0.064}$

(b) Omniglot results.

We evaluate the Meta-Baseline (MB) and Zero-ing Trick (ZO) classifier using BatMan-CLR and Man-CLR meta-trained on CifarFS and Omniglot. In , we provide an overview of these experiments. The grey columns represent re-printed results from chapter 2. The classifier constructed with the zero-ing trick achieves higher meta-testing accuracy, regardless of the training noise level. Moreover, the MAML style learners deteriorate to random label prediction under the Meta-Baseline (MB) style classification. The performance gap between MO and ZO is smaller for the Reptile style learners. Reptile and Eigen Reptile paired with BatMan on Omniglot sees a performance decrease of $\sim 11\%$, with a slightly larger gap with Man sampling. On Omniglot, however, only Reptile shows the same pattern, decreasing $\sim 9\%$ and $\sim 13\%$) for BatMan and Man respectively. On the other hand, Eigen Reptile shows between ZO and MB a considerably larger performance gap of $\sim 26\%$ and 44% between BatMan and Man.

5.2. Contrastive Loss and Embedding Size

The utilized contrastive loss (DCL) and the embedding layer size are essential components of our proposed method. The embedding layer is characterized by a single fully-connected layer that maps the outputs of the convolutional backbone to an embedding vector in $\mathbb{R}^{128}$. To evaluate the impact of the embedding size and contrastive loss, we evaluate both at different levels. We consider the DCL and simCLR losses with embedding sizes of 5 and 128 paired trained with a Reptile meta-learner. In addition, we evaluate an Oracle variant of DCL (O-DCL) to act as a 'skyline' in this setting. O-

DCL is allowed access to ground-truth labels, ensuring that no false-negative pairs are used in DCL's (Equation 3.8) denominator. As a baseline, we train supervised meta-learners under the same levels of training noise noise Table 5.2 with the same model architecture.

Table 5.2: (clean) Meta-test accuracy with 95th Confidence Interval ($\overline{acc}$± stdev) of Supervised meta-learners on (5, 5)-FSL FC100 with different noise levels (ϵ).

Learner	$\epsilon = 0.0$	$\epsilon = 0.3$	$\epsilon = 0.6$
Eigen Reptile	44.6±4.0	41.9±3.9	39.7±3.8
Reptile	44.5±4.1	42.6±3.9	39.9±3.7
foMAML	42.6±4.0	39.0±4.5	32.1±4.9
iMAML	45.2±4.1	41.8±3.9	31.9±4.7

Table 5.3: (clean) Meta-test accuracy with 95th Confidence Interval ($\overline{acc}$ ± stdev) of meta-learners trained with different contrastive losses in the inner-loop on (5, 5)-FSL FC100 with different noise levels (ϵ).

Loss	$\epsilon = 0.0$	$\epsilon = 0.3$	$\epsilon = 0.6$
O-DCL	44.3±1.6	44.0±1.8	44.0±1.6
simCLR	41.5±1.6	41.5±1.5	41.5±1.5
DCL	41.5±1.5	41.8±1.6	41.4±1.5

(a) Embedding dimension of 128.

$\epsilon = 0.0$	$\epsilon = 0.3$	$\epsilon = 0.6$
37.7±1.4	37.6±1.5	37.2±15
36.2±1.5	36.2±1.5	36.4±15
36.2±1.4	36.3±1.4	36.2±15

(b) Embedding dimension of 5.

Table 5.3 provides the results of meta-learners trained with different contrastive losses on FC100. Table 5.2 summarizes the results of meta-learners trained with supervised losses on the same dataset. We see a relatively low accuracy of around $44/45\%$ at zero noise, indicating that the trained model did not generalize well. We hypothesize that this is due to the model architecture being insufficient to learn to generalize in this setting. Nonetheless, these results allow us to draw some conclusions. Moreover, compared to its supervised counterpart in Table 5.2 with clean training data, a performance gap is shown of about 3%. However, as training noise increases, the supervised learners deteriorate by up to 14.6%, while the contrastive learned models remain largely unaffected.

Between the different contrastive configurations, we see two remarkable patterns in Table 5.3a and Table 5.3b. As the embedding dimension decreases, we see an accuracy drop of around 4%. Moreover, we see a noticeable difference between O-DCL and DCL trained without noise ($\epsilon = 0.0$). Showing that the false-negative constructed by DCL (from samples of the same class) hurt the learners' performance.

5.3. Meta-test noise

Besides investigating meta-training noise, we also study the effect of meta-testing with label noise. First, we provide results of supervised meta-learners with task-level symmetric label noise during training and testing. Afterward, we provide results on the baseline and BatMan-CLR learned models under different trained with a noisy meta-dataset and asymmetric label noise during testing.

5.3.1. Symmetric Task-level Noise

Initial experiments were trained and tested with task-level symmetric label noise. These tasks were constructed by sampling clean tasks and then reassigning the labels to classes from the same task for a fraction (ϵ). In this setting, samples can be provided with a correct or corrupted label in different few-shot tasks. Supervised meta-learners were trained and tested under different levels of meta-training noise ($\epsilon \in [0.0, 0.3, 0.6]$) and meta-testing noise ($\epsilon \in [0.0, 0.15, 0.3, 0.45, 0.6]$). The meta-learners were trained on the CifarFS and FC100 datasets.

Figure 5.2 shows the results on the CIFAR-derived datasets. Two negative trends are visible in Figure 5.2 as the noise levels increase. First, as the meta-training noise increases, the meta-test performance drops. However, the Reptile style learners–Reptile and Eig(en)Reptile–are seemingly

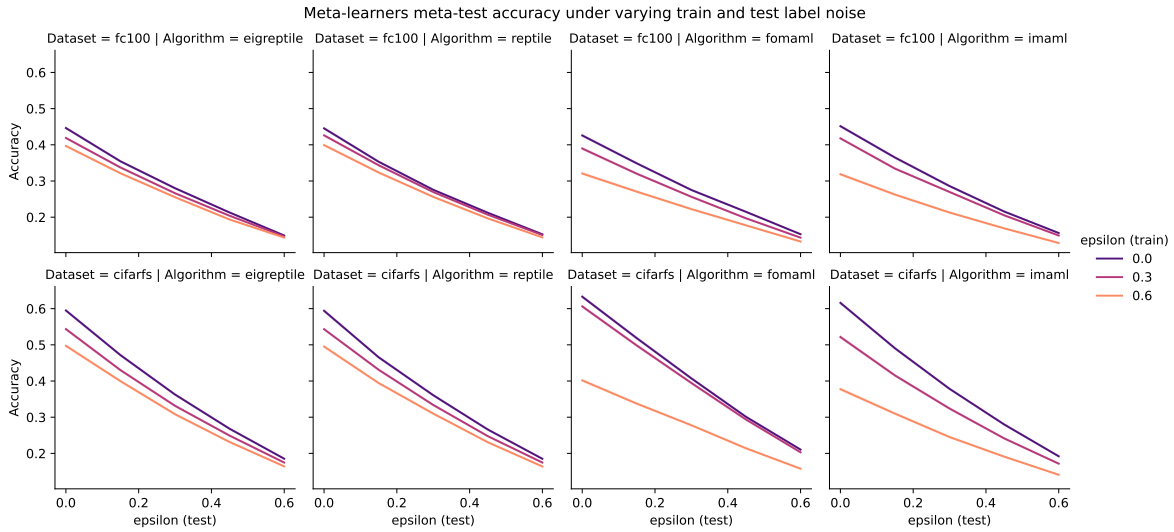


Figure 5.2: Meta-test performance comparison of supervised meta-trained on (5, 5)-FSL tasks, with different training and testing noise levels. Showing results from training under varying levels of task-level label noise during meta-training ($\epsilon \in \{0.0, 0.3, 0.6\}$) and meta-testing ($\epsilon \in \{0.0, 0.15, 0.3, 0.45, 0.6\}$).

less affected by the meta-training noise, as the spread of the accuracy curves is narrower compared to iMAML and foMAML. Second, as the meta-test noise increases, all learners are negatively impacted by the noise, indicating that they overfit to meta-testing noise. This impact, however, is more pronounced than the impact of meta-training noise. Compared with the impact of training noise, this accuracy drop is more severe, showing that under $\epsilon = 0.6$, the few-shot learners deteriorate to random guessing– regardless of the training noise.

Figure 5.3: Example of task-level corrupted (5, 5)-FSL tasks' support set with asymmetric *task level* label noise. The samples are ordered per row with the label they were provided with. Can you spot the incorrectly labeled samples[1]? Original images courtesy of (Mini)Imagenet [11].

5.3.2. Meta-testing with Asymmetric Label Noise

Lastly, we consider the impact of meta-testing noise on supervised and BatMan-CLR meta-learned models. During the meta-train time, the noise is generated at the dataset level and during meta-testing

[1] Row 1: 3 (row 3), Row 2: 1 (row 4), Row 3: 4 (row 4), Row 4: 2 (1st row), Row 5: 1 (row 2).

at the task level. Note that query data during testing is not corrupted. Figure 5.3 shows a corrupted support set of a (5, 5)-FSL task with asymmetric label noise ($\epsilon = 0.2$); samples for each class are grouped together per row. During meta-testing, the noise is created by creating one-directional mappings for classes, so no class gets mapped to itself. Afterward, a fraction (ϵ) of each class's shots get corrupted according to the created mappings. Thereby assigning a fraction of each class to another, modeling class confusion. Here we evaluate the impact of meta test-noise under supervised and BatMan-CLR The results are plotted in Figure 5.4.

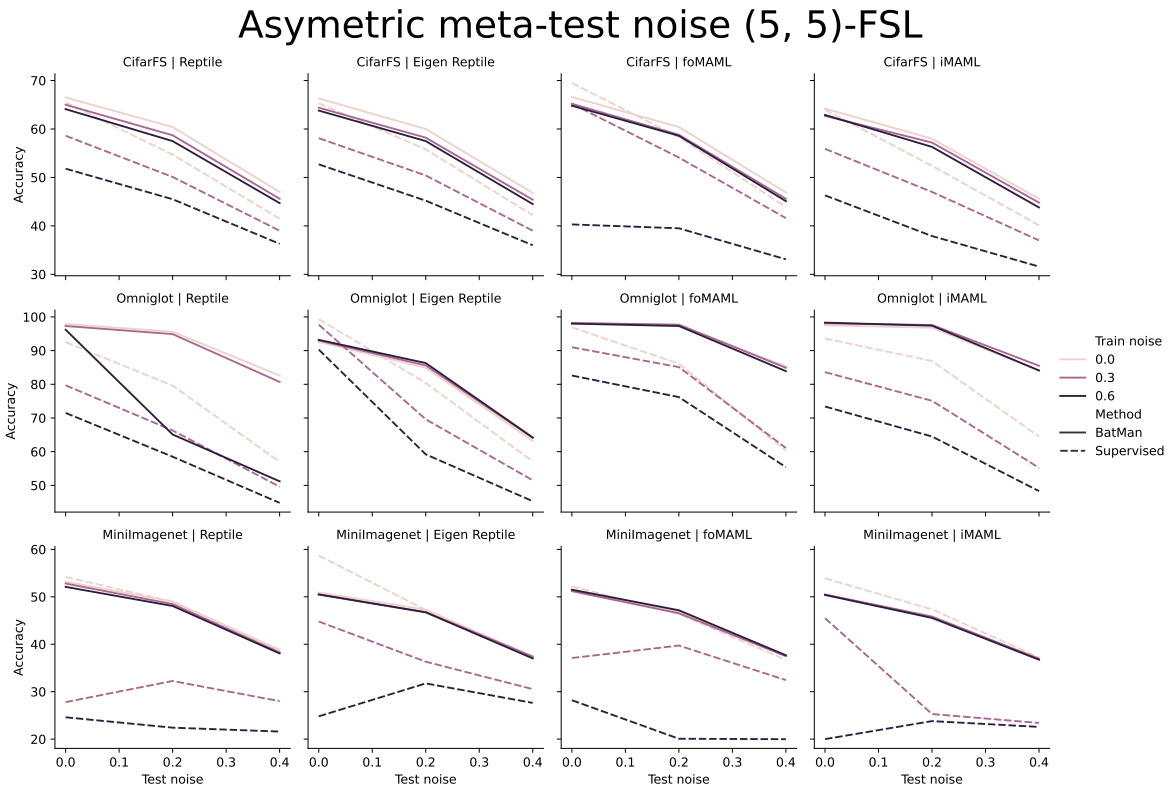


Figure 5.4: Meta-test performance comparison of supervised and BatMan-CLR meta-trained models with different meta-learners, training, and testing noise levels. Showing results from training with dataset-level symmetric label noise ($\epsilon \in \{0.0, 0.3, 0.6\}$) and asymmetric task-level meta-testing label noise ($\epsilon \in \{0.0, 0.2, 0.4\}$). Note that the accuracy axes (y-axes) are scaled *per dataset* (row).

Generally, Figure 5.4 shows a negative trend as the meta-train and meta-test noise increases. Overall we see that the spread of the supervised accuracy curves is higher for supervised learners than for BatMan-CLR trained models, corresponding with their resistance against meta-training label noise. Except for Reptile+BatMan-CLR with $\epsilon = 0.6$ meta-training noise. In this configuration, the Reptile learners' performance degrades to the level of its supervised counterpart at $\epsilon = 0.3$ training noise.

On CifarFS and especially (Omniglot) (top rows in Figure 5.4), the BatMan-CLR meta-learned models show a less pronounced impact as meta-test noise increases meta-testing, compared to the supervised results. On these datasets, we see an improvement across all meta-learners at $\epsilon = 0.2$ of $2.1 - 5.7\%$ $(4.5 - 16.0\%)$ and $4.6 - 5.5\%$ $(6.1 - 25.6\%)$ at $\epsilon = 0.4$, over supervised results. Nonetheless, the BatMan-CLR learners see a degradation as the meta-testing noise increases, of $5.5 - 6.6\%$ $(0.5 - 31.1\%)$ at $\epsilon = 0.2$ and $18.6 - 19.7\%$ $(12.6 - 45.0\%)$ with $\epsilon = 0.4$. MiniImagenet results (bottom row in Figure 5.4) show that the BatMan-CLR meta-learned models under test noise performs equivalent to the cleanly pre-trained meta-learner. Supervised models trained with label noise on MiniImagenet show a drastic deterioration in performance,

In general, comparing the BatMan-CLR with the supervised results, the supervised model models show a larger spread of the accuracy curves. Meanwhile, the BatMan-CLR curves show a relatively small spread, excluding the abovementioned exception. Lastly, we remark on an unexpected pattern

of supervised learners on MiniImagenet. Where some learners see a slight uplift ($\sim 2.7 - 6.9\%$) uplift at $\epsilon = 0.3$ training noise as the meta-testing noise increases.

6

Conclusion and Future Work

This chapter aims to answer the research questions posed in chapter 1. Afterward, we discuss potential directions for future work.

6.1. Conclusions

Following the results as presented in the research paper in chapter 2 and chapter 5 we can shine some light on the research questions as posed in chapter 1.

> 1. How does meta-training with label noise affect a meta-learned model to adapt to new tasks during meta-training?

As discussed in the paper in chapter 2, we evaluate four different meta-learners, Reptile, Eigen Reptile, foMAML, and iMAML, on three few-shot benchmarks, Omniglot, CifarFS, and MiniImagenet. These configurations are meta-learned with varying degrees of symmetric label noise $\epsilon \in [0.0, 0.3, 0.6]$ on the underlying meta-train data split.

The Reptile, iMAML, and foMAML meta-learners show a degradation in meta-test performance up to 42% on the Omniglot and CifarFS datasets. Additionally, we consider the Eigen Reptile meta-learner, which aims to regularize the meta-gradient calculation to improve label noise robustness. Eigen Reptile shows an improved label noise robustness over Reptile of 4.9% and (17.7%) with $\epsilon = 0.3$ and 1.8% (0.2%) with $\epsilon = 0.6$ for Omniglot and (MiniImagenet) respectively. However, it still performs markedly worse than its noise-free trained counterpart with degradation of 10% and 13.9% for Omniglot and MiniImagenet compared to the noiseless their noiseless counterpart at $\epsilon = 0.3$, degrading further with higher levels of noise. On the more challenging MiniImagenet, all supervised meta-learners' deteriorate close to random guessing $(28.1 - 20.0\%)$ at a training noise level of 60%. As such, this provides evidence that meta-training with label noise negatively impacts the learned models' ability to adapt to new tasks in few-shot learning.

> 2. How can existing meta-learners be adapted to mitigate the impact of label noise during meta-training?

We have implemented two novel sub-sampling techniques Man and BatMan. These samplers allow semi-supervised manifold samples to be constructed from noisy few-shot learning instances. We pair these samplers with a contrastive loss (DCL), which allows for meta-learning with the manifold samples. Consecutively, we train meta-learners equipped with Man and BatMan with the same datasets, meta-learners, and noise levels as in the abovementioned experiments.

We show that meta-learners trained with contrastive losses paired with BatMan-CLR and (Man-CLR) on Omniglot and CifarFS can generalize regardless of the label noise during meta-testing. Showing a small performance degradation on CifarFS and Omniglot of $1.3-2.5\%$ $(0.4-9.4\%)$ and $-0.8-1.7\%$ $(-0.4 - 0.2\%)$ as the meta-training label noise increases from 0 to 0.6, respectively. On MiniImagenet, we see that BatMan-CLR trained meta-learners see an accuracy decrease of $\sim 0 - 1.1\%$ across all meta-learners.

As a result, showing that BatMan-CLR and Man-CLR provide We show that under the same configurations of the supervised learners, the Man-CLR and BatMan-CLR learned meta-learners are minimally impacted by label noise. Thereby showing that the objective function used during meta-training can help mitigate the impact of label noise.

> 3. How does label noise affect meta-learned few-shot classifiers during meta-testing? Does label-noise robust meta-training improve meta-testing robustness against meta-testing label noise?

Lastly, in chapter 5, we consider the impact of two types of meta-testing noise. We evaluate the four meta-learners with two types of task-level noise during meta-testing. First, we evaluate supervised meta-learners with symmetric task-level noise during meta-testing. Second, we consider the supervised and BatMan-CLR meta-learned few-shot learners and subject them to asymmetric task-level noise ($\epsilon \in [0.0, 0.2, 0.4]$). These results agree with related works on meta-testing with label noise few-shot learners and do not inherently provide label noise robustness.

The symmetric meta-testing results indicate that supervised meta-learners tend to overfit to label noise. Similarly, when evaluating meta-learners, we observed that the meta-learned few-shot models do not generalize well as the test noise increases. The robust meta-trained BatMan-CLR few-shot models show a slightly improved meta-test robustness compared to their supervised counterparts on CifarFS and Omniglot under noise levels of 0.2 and (0.4). Showing a gain of 2.1-5.7% (3.0-5.5%) for CifarFS and 4.5-16.0% (5.1-25.6%). On MiniImagenet, however, this difference is considerably smaller, with differences of -1.5-0.6% (-0.5%-0.9%). As such, this indicates that training with a BatMan-CLR does not necessarily translate to a label noise robust few-shot learning model during meta-testing.

6.2. Future work

This work focuses on meta-training with label noise. We focus on symmetric label noise in the meta-data training data. Extending this to different types of label noise would be valuable to gain additional insight into the effect of label noise on meta-learners. For example, introducing out-of-domain samples or considering a fixed set of tasks, similar to [56].

Although we perform ablations on meta-testing with label noise, this could be further explored. This work focuses mainly on intra-task types and noise types in these studies. Extending this to out-of-task types of label noise, would be of interest.

Besides exploring different types of meta-testing noise, adapting BatMan sampling to be used during meta-testing is a natural future extension of this work. Although preliminary experiments were performed (see Appendix A), we consider that BatMan or an adaptation thereof could allow for improved meta-testing performance with label noise. Using BatMan-CLR or combined with a self-paced (introspective) learning strategy could improve label noise robustness during evaluation. Moreover, such strategies could be explored for fine-tuning a BatMan-CLR meta-learned few-shot learner on noisy meta-training data.

Currently, we consider a limited set of (contrastive) losses and a single (cosine similarity) metric function. Recent works [31, 39] have shown that the Prototypical losses [53] are well suited for few-shot contrastive learning. We consider learning BatMan-CLR under different losses and metrics could be interesting to explore further.

We have not extensively explored filtering or correcting noisily labeled samples during our work. Although samples are noisy, further exploiting the relations between samples can be beneficial. For example, noisy samples may correspond to another class in a task, or multiple out-of-task samples may be of the same class. Further leveraging such approaches could further improve the performance of meta-learners trained with label noise. We consider that incorporating a representation rectification approach, as considered in unsupervised few-shot learning [51, 52], could be a promising avenue to explore. Moreover, such techniques could provide a means to leverage such a rectification module to combat meta-testing noise, akin to [36, 34], *without the need for clean training data*.

Finally, regarding the developed framework `nmfw`. Its current feature set for few-shot data generation and meta-learners leaves room for extension. We plan on improving its feature set for few-shot learning (and testing) with more noise types. Additionally, some examples for using different meta-learning libraries, such as TorchOpt and `higher`, are planned to be included. Additionally, more meta-learners and other few-shot learners would be valuable to add to the toolkit.

A

Unsuccessful directions

Analysis of inner-loop generalization process. This direction regarded collecting and analyzing the inner-loop statistics during the meta-learning training loop. This approach was motivated by noise-robust learning, which uses early stopping as regularization against noise. These works leverage a deep learners' ability first to learn 'easy' patterns, represented by samples with clean labels. After fitting to the 'easy' samples, these learners tend to overfit to noisy samples, effectively memorizing them. However, we did not observe this behavior in few-shot meta-learners due to the limited data available in each few-shot task.

Modeling GMBLs as recurrent networks. By implementing an optimizer that treats the model updates as 'hidden states', the gradient descent through gradient descent could be more configurable. However, we discarded this idea in favor of a less complicated approach.

Approximating MAML with random paths. MAML and foMAML represent an 'all-or-nothing' approach for back-propagating through the inner loop. Leveraging Hessian Free-MAML [13], the idea is to approximate the gradient descent through stepping through randomly sampled inner-loop steps.

Stochastic Weighted Averaging inner-loop. As an extension of the previous, the idea was to create a Stochastic Weight Averaging [22] (SWA) optimizer for gradient-based meta-learners. Initially, to leverage its ability to learn more generalized models through its averaging approach. Consecutively, SWA's averaging coefficient would become a hyper-parameter to be tuned/optimized by the meta-learner. They were motivated by the observation that earlier steps of inner-loop adaptation would improve meta-learners' resilience against noise. However, this did not yield improvements after evaluating a basic before the step towards 'learning not to overfit to label noise' was made.

Meta-Lottery tickets was explored to combine the lottery-ticket hypothesis [60]. This hypothesis states that in a randomly initialized (deep) neural network, subnetworks–winning 'lottery tickets'–are capable of achieving high accuracy on a given task when trained in isolation. These winning tickets can be identified through pruning the network, which iteratively extracts these tickets. The idea being that during meta-learning, the meta-networks could be pruned to find a smaller meta-lottery ticket. Given that such networks have less trainable weights, these could be used to meta-learn a model capable of resisting overfitting on label noise, due to its 'memory' constraint. However, this avenue was dropped to explore a label noise-robust optimization strategy for the inner and/or outer loop of meta-learners.

Robust per-sample gradient decomposition optimizer. From the perspective of gradients, label noise introduces gradients that point in the 'wrong' optimization direction. Then by exploiting per-sample gradients, the idea was to find updated directions both per provided class and overall through KRUM and multi-KRUM inspired decomposition. This would allow for a drop-in replacement that could be used for all meta-learners. This, however, did not translate into improved testing performance.

Robust inner-optimizer. A natural extension of the Eigen-Reptile outer-loop decomposition rule would be to (iteratively) apply it in the inner loop of meta-learner. As a result, reducing the impact of label noise would by 'filtering out' the impact of the label noise component of the inner adaptation process. However, this approach did not positively improve meta-testing performance in the explorative experiments. As such, this idea was dropped in favor of exploring contrastive learning.

Introspective regularization term. The idea was that as the training process of BatMan-CLR continued, false negatives would become more aparent, as these would lie close to their true class

embeddings. As such, samples with a different provided label, but high similarity would likely be a false negative. We considered different techniques to find such false negatives, random walks over similarity matrices, and clustering based on similarity metrics (cosine similarity and reciprocal Jaccard similarity). Although small-scale testing seemed to have promising results, these did not translate into improvements when applied to noisy meta-testing or meta-training.

BatMan-CLR during noisy meta-testing. As an extension of BatMan-CLR for meta-training, meta-testing with BatMan-CLR was considered as well. Adding a BatMan (-CLR) loss term in the inner-loop adaptation process could help prevent overfitting to samples with incorrect labels. However, small-scale evaluations showed that the few-shot learners did not improve from an additive contrastive loss term during testing. Due to time constraints, it was chosen not to explore this avenue further.

Label noise implementation with TorchMeta. TorchMeta was the initial library that `nmfw` utilized for (noisy) data generation in `nmfw`. However, as the work shifted from task-level corruption to meta-training noise, it became apparent that Learn2Learn provided a more suitable abstraction for few-shot learning with label noise. These 'results' have contributed to the discussion in chapter 4.

Bibliography

[1] Antreas Antoniou and Amos Storkey. “Assume, Augment and Learn: Unsupervised Few-Shot Meta-Learning via Random Labels and Data Augmentation”. In: ().

[2] Sébastien M. R. Arnold et al. “learn2learn: A Library for Meta-Learning Research”. In: (Aug. 2020).

[3] Luca Bertinetto et al. “Meta-learning with differentiable closed-form solvers”. In: *7th International Conference on Learning Representations, ICLR 2019*. 2019.

[4] Ran Bi et al. “PaddlePaddle: A Production-Oriented Deep Learning Platform Facilitating the Competency of Enterprises”. In: *Proceedings - 24th IEEE International Conference on High Performance Computing and Communications, 8th IEEE International Conference on Data Science and Systems, 20th IEEE International Conference on Smart City and 8th IEEE International Conference on Dep*. 2022, pp. 92–99. ISBN: 9798350319934. DOI: 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00046.

[5] Victor Boutin et al. “Diversity vs. Recognizability: Human-like generalization in one-shot generative models”. In: *ArXiv* (2022). DOI: 10.48550/ARXIV.2205.10370.

[6] Dong Chen et al. “Robust Meta-learning with Sampling Noise and Label Noise via Eigen-Reptile”. In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 3662–3678.

[7] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *37th International Conference on Machine Learning, ICML 2020*. Vol. PartF16814. 2020, pp. 1575–1585. ISBN: 9781713821120.

[8] Xinlei Chen and Kaiming He. “Exploring Simple Siamese Representation Learning”. In: *{IEEE} Conference on Computer Vision and Pattern Recognition, {CVPR} 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / {IEEE}, 2021, pp. 15750–15758. DOI: 10.1109/CVPR46437.2021.01549.

[9] Yinbo Chen et al. “Meta-Baseline: Exploring Simple Meta-Learning for Few-Shot Learning”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2021), pp. 9042–9051. ISSN: 15505499. DOI: 10.1109/ICCV48922.2021.00893.

[10] Tristan Deleu et al. “Torchmeta: A Meta-Learning library for PyTorch”. In: *CoRR* (Sept. 2019).

[11] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. Institute of Electrical and Electronics Engineers (IEEE), Mar. 2010, pp. 248–255. DOI: 10.1109/cvpr.2009.5206848.

[12] Alexey Dosovitskiy et al. “an Image Is Worth 16X16 Words: Transformers for Image Recognition At Scale”. In: *ICLR 2021 - 9th International Conference on Learning Representations*. OpenReview.net, 2021.

[13] Alireza Fallah, Aryan Mokhtari, and Asuman E Ozdaglar. “On the Convergence Theory of Gradient-Based Model-Agnostic Meta-Learning Algorithms”. In: *The 23rd International Conference on Artificial Intelligence and Statistics, {AISTATS} 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*. Ed. by Silvia Chiappa and Roberto Calandra. Vol. 108. Proceedings of Machine Learning Research. PMLR, 2020, pp. 1082–1092.

[14] Chen Fan, Parikshit Ram, and Sijia Liu. “Sign-MAML: Efficient Model-Agnostic Meta-Learning by SignSGD”. In: *CoRR* (Dec. 2021).

[15] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *34th International Conference on Machine Learning, ICML 2017* 3 (2017), pp. 1856–1868.

[16] Edward Grefenstette et al. “Generalized Inner Loop Meta-Learning”. In: *CoRR* (2019).

[17] Jean-Bastien Grill et al. “Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21271–21284.

[18] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 {IEEE} Conference on Computer Vision and Pattern Recognition, {CVPR} 2016, Las Vegas, NV, USA, June 27-30, 2016*. {IEEE} Computer Society, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[19] Kaiming He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *2020 {IEEE/CVF} Conference on Computer Vision and Pattern Recognition, {CVPR} 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / {IEEE}, 2020, pp. 9726–9735. DOI: 10.1109/CVPR42600.2020.00975.

[20] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *Neural Networks for Machine Learning Lecture 6a Overview of mini--batch gradient descent*. 2012.

[21] Kyle Hsu, Sergey Levine, and Chelsea Finn. “Unsupervised Learning via Meta-Learning”. In: *7th International Conference on Learning Representations, {ICLR} 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, Jan. 2019.

[22] Pavel Izmailov et al. “Averaging weights leads to wider optima and better generalization”. In: *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*. Ed. by Amir Globerson and Ricardo Silva. Vol. 2. AUAI Press, 2018, pp. 876–885. ISBN: 9781510871601.

[23] Chia-Hsiang Kao, Wei-Chen Chiu, and Pin-Yu Chen. “MAML is a Noisy Contrastive Learner in Classification”. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022* (2021).

[24] Siavash Khodadadeh, Ladislau Bölöni, and Mubarak Shah. “Unsupervised Meta-Learning for Few-Shot Image Classification”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M Wallach et al. Vol. 32. 2019, pp. 10132–10142.

[25] Prannay Khosla et al. “Supervised Contrastive Learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18661–18673.

[26] Krishnateja Killamsetty et al. “A Nested Bi-level Optimization Framework for Robust Few Shot Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 7. OpenReview.net, 2022, pp. 7176–7184. ISBN: 1577358767. DOI: 10.1609/aaai.v36i7.20678.

[27] Diederik P. Kingma and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (Dec. 2014).

[28] Alexander Kirillov et al. “Segment Anything”. In: *CoRR* abs/2304.0 (2023). DOI: 10.48550/arXiv.2304.02643.

[29] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* 350.6266 (Dec. 2015), pp. 1332–1338. ISSN: 10959203. DOI: 10.1126/SCIENCE.AAB3050.

[30] Michalis Lazarou, Tania Stathaki, and Yannis Avrithis. “Iterative label cleaning for transductive and semi-supervised few-shot learning”. In: ().

[31] Junnan Li et al. “Prototypical Contrastive Learning of Unsupervised Representations”. In: *9th International Conference on Learning Representations, {ICLR} 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[32] Pan Li et al. “SEMI-SUPERVISED FEW-SHOT LEARNING WITH PSEUDO LABEL REFINEMENT”. In: *Proceedings - IEEE International Conference on Multimedia and Expo* (2021). ISSN: 1945788X. DOI: 10.1109/ICME51207.2021.9428178.

[33] Wenbin Li et al. “LibFewShot: A Comprehensive Library for Few-shot Learning”. In: (Sept. 2021).

[34] Kevin J Liang et al. “Few-shot Learning with Noisy Labels”. In: *CVPR* (2022). DOI: 10.48550/arXiv.2204.05494.

[35] Chen Liu et al. “Learning a Few-shot Embedding Model with Contrastive Learning”. In: *35th AAAI Conference on Artificial Intelligence, AAAI 2021*. Vol. 10A. 2021, pp. 8635–8643. ISBN: 9781713835974. DOI: 10.1609/aaai.v35i10.17047.

[36] Jiang Lu et al. “Robust few-shot learning for user-provided data”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.4 (Apr. 2021), pp. 1433–1447. ISSN: 21622388. DOI: 10.1109/TNNLS.2020.2984710.

[37] Sébastien Marcel and Yann Rodriguez. “Torchvision the machine-vision package of Torch”. In: (2010).

[38] Pratik Mazumder, Pravendra Singh, and Vinay P Namboodiri. “RNNP: A Robust Few-Shot Learning Approach”. In: *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*. IEEE, 2021, pp. 2663–2672. DOI: 10.1109/WACV48630.2021.00271.

[39] Carlos Medina, Arnout Devos, and Matthias Grossglauser. “Self-Supervised Prototypical Transfer Learning for Few-Shot Classification”. In: *CoRR* abs/2006.1 (2020).

[40] Muhammad Abdullah Jamal, Liqiang Wang, and Boqing Gong. “A Lazy Approach to Long-Horizon Gradient-Based Meta-Learning”. In: *International Conference on Computer Vision* (2021), pp. 6577–6586.

[41] Alex Nichol, Joshua Achiam, and John Schulman. “On First-Order Meta-Learning Algorithms”. In: *CoRR* abs/1803.0 (2018), pp. 1–15.

[42] Jaehoon Oh et al. “BOIL: Towards Representation Change for Few-shot Learning”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[43] OpenAI. “GPT-4 Technical Report”. In: *CoRR* abs/2303.0 (Mar. 2023). DOI: 10.48550/arXiv.2303.08774.

[44] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[45] Aniruddh Raghu et al. “Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML”. In: *8th International Conference on Learning Representations, {ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020, pp. 1–21.

[46] Aravind Rajeswaran et al. “Meta-learning with implicit gradients”. In: *Advances in Neural Information Processing Systems* 32.NeurIPS (2019), pp. 1–12. ISSN: 10495258.

[47] Sachin Ravi and Hugo Larochelle. “Optimization as a model for few-shot learning”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. July 2017.

[48] Jie Ren et al. “TorchOpt: An Efficient Library for Differentiable Optimization”. In: (Nov. 2022).

[49] Anshul Shah et al. “Max-Margin Contrastive Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.8 (2022), pp. 8220–8230. ISSN: 2159-5399. DOI: 10.1609/aaai.v36i8.20796.

[50] Zhenqian Shen et al. “PaddleFSL: A General Few-Shot Learning Toolbox in Python”. In: *Journal of Machine Learning Research* 1 (2000), pp. 1–48.

[51] Ojas Kishore Shirekar and Hadi Jamali-Rad. “Self-Supervised Class-Cognizant Few-Shot Classification”. In: *Proceedings - International Conference on Image Processing, ICIP*. IEEE Computer Society, 2022, pp. 976–980. ISBN: 9781665496209. DOI: 10.1109/ICIP46576.2022.9897431.

[52] Ojas Kishorkumar Shirekar, Anuj Singh, and Hadi Jamali Rad. “Self-Attention Message Passing for Contrastive Few-Shot Learning”. In: *{IEEE/CVF} Winter Conference on Applications of Computer Vision, {WACV} 2023, Waikoloa, HI, USA, January 2-7, 2023*. IEEE, Oct. 2023, pp. 5414–5425. ISBN: 9781665493468. DOI: 10.1109/WACV56688.2023.00539.

[53] Jake Snell, Kevin Swersky, and Richard Zemel. “Prototypical Networks for Few-shot Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by I Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[54] Oriol Vinyals et al. “Matching networks for one shot learning”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016, pp. 3637–3645.

[55] Zhanyuan Yang, Jinghua Wang, and Yingying Zhu. “Few-Shot Classification with Contrastive Learning”. In: *Computer Vision - ECCV 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XX*. Ed. by Shai Avidan et al. Vol. 13680. Lecture Notes in Computer Science. Springer, 2022, pp. 293–309. DOI: 10.1007/978-3-031-20044-1{_}17.

[56] Huaxiu Yao et al. “Meta-learning with an Adaptive Task Scheduler”. In: *Advances in Neural Information Processing Systems*. Vol. 9. 2021, pp. 7497–7509. ISBN: 9781713845393.

[57] Han Jia Ye, Lu Han, and De Chuan Zhan. “Revisiting Unsupervised Meta-Learning via the Characteristics of Few-Shot Tasks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Mar. 2022). ISSN: 19393539. DOI: 10.1109/TPAMI.2022.3179368.

[58] Chun-Hsiao Yeh et al. “Decoupled Contrastive Learning”. In: *Computer Vision - {ECCV} 2022 - 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part {XXVI}*. Ed. by Shai Avidan et al. Vol. 13686. Lecture Notes in Computer Science. Springer, 2022, pp. 668–684. DOI: 10.1007/978-3-031-19809-0{\{}{\textbackslash}{_}{\}}38.

[59] Mingzhang Yin et al. “Meta-Learning without Memorization”. In: *8th International Conference on Learning Representations, {ICLR} 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[60] Shuai Zhang et al. “Why Lottery Ticket Wins? A Theoretical Perspective of Sample Complexity on Pruned Neural Networks”. In: NeurIPS 2021 (2021), pp. 1–14.