Data and Computation Efficient Meta-Learning

John Fitzgerald Bronskill

Department of Engineering
University of Cambridge

This thesis is submitted for the degree of
Doctor of Philosophy

Wolfson College July 2020

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my thesis has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

John Fitzgerald Bronskill
July 2020

Data and Computation Efficient Meta-Learning

John Fitzgerald Bronskill

Abstract

In order to make predictions with high accuracy, conventional deep learning systems require large training datasets consisting of thousands or millions of examples and long training times measured in hours or days, consuming high levels of electricity with a negative impact on our environment. It is desirable to have have machine learning systems that can emulate human behavior such that they can quickly learn new concepts from only a few examples. This is especially true if we need to quickly customize or personalize machine learning models to specific scenarios where it would be impractical to acquire a large amount of training data and where a mobile device is the means for computation. We define a *data efficient* machine learning system to be one that can learn a new concept from only a few examples (or *shots*) and a *computation efficient* machine learning system to be one that can learn a new concept rapidly without retraining on an everyday computing device such as a smart phone.

In this work, we design, develop, analyze, and extend the theory of machine learning systems that are both *data efficient* and *computation efficient*. We present systems that are trained using multiple tasks such that it "learns how to learn" to solve new tasks from only a few examples. These systems can efficiently solve new, unseen tasks drawn from a broad range of data distributions, in both the low and high data regimes, without the need for costly retraining. Adapting to a new task requires only a forward pass of the example task data through the trained network making the learning of new tasks possible on mobile devices. In particular, we focus on few-shot image classification systems, i.e. machine learning systems that can distinguish between numerous classes of objects depicted in digital images given only a few examples of each class of object to learn from.

To accomplish this, we first develop ML-PIP, a general framework for Meta-Learning approximate Probabilistic Inference for Prediction. ML-PIP extends existing probabilistic interpretations of meta-learning to cover a broad class of methods. We then introduce VERSA, an instance of the framework employing a fast, flexible and versatile amortization network that takes few-shot learning datasets as inputs, with arbitrary numbers of training examples, and outputs a distribution over task-specific parameters in a single forward pass of the network. We evaluate VERSA on benchmark datasets, where at the time, the method achieved state-of-the-art results when compared to meta-learning approaches using similar training regimes and feature extractor capacity.

Next, we build on VERSA and add a second amortized network to adapt key parameters in the feature extractor to the current task. To accomplish this, we introduce CNAPS, a conditional neural process based approach to multi-task classification. We demonstrate that, at the time, CNAPS achieved state-of-the-art results on the challenging META-DATASET benchmark indicating high-quality transfer-learning. Timing experiments reveal that CNAPs is computationally efficient when adapting to an unseen task as it does not involve gradient back propagation computations. We show that trained models are immediately deployable to continual learning and active learning where they can outperform existing approaches that do not leverage transfer learning.

Finally, we investigate the effects of different methods of batch normalization on meta-learning systems. Batch normalization has become an essential component of deep learning systems as it significantly accelerates the training of neural networks by allowing the use of higher learning rates and decreasing the sensitivity to network initialization. We show that the hierarchical nature of the meta-learning setting presents several challenges that can render conventional batch normalization ineffective. We evaluate a range of approaches to batch normalization for few-shot learning scenarios, and develop a novel approach that we call TaskNorm. Experiments demonstrate that the choice of batch normalization has a dramatic effect on both classification accuracy and training time for both gradient based- and gradient-free meta-learning approaches and that TaskNorm consistently improves performance.

I would like to dedicate this thesis to my wonderful wife Andrea who demonstrated both encouragement and patience while I pursued this work.

Acknowledgements

I have been very fortunate to have Richard Turner and Sebastian Nowozin as PhD supervisors. Both have provided superb guidance, deep insights, and extensive contributions to my work. I want to thank Rich for taking me on as a student even though it had been more than thirty years since I completed my Masters degree. Being in Rich's group has been a fantastic experience and a great privilege. I have always had deep respect for Sebastian's work and know-how and I am lucky that he agreed to co-supervise and contribute numerous key ideas and suggestions. I would also like to thank my advisor Dr. José Miguel Hernández Lobato for his support.

I was also very fortunate to have a desk next to Jonathan Gordon who has been a fabulous collaborator. I learned a lot from him and he contributed jointly to almost all aspects of the research contained in this thesis. I would also like to acknowledge the valuable and significant contributions of my other awesome student collaborators and co-authors Matthias Bauer and James Requeima - it has been a great pleasure to work with both of them.

Finally, a big thanks to all the members of the Computational and Biological Learning group at the University of Cambridge for helping make my PhD experience so enjoyable.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

1D	1-Dimensional
2D	2-Dimensional
3D	3-Dimensional
C-VAE	Conditional Variational Autoencoder
CNAPs	Conditional Neural Adaptive Processes
CNP	Conditional Neural Process
D^τ	The context set for a task τ
GPU	Graphics Processing Unit
GQN	Generative Query Network
i.i.d.	independent and identically distributed
LSTM	Long Short-Term Memory
MAML	Model-Agnostic Meta-Learning
MAP	Maximum A Posteriori
ML-PIP	Meta-learning Probabilistic Inference for Prediction
NL	Normalization Layer
NP	Neural Process
SVGD	Stein Variational Gradient Descent
T^τ	The target set for a task τ
τ	A task in a multi-task learning setting

Chapter 1

Introduction

1.1 Motivation

Humans are able to learn a new concept from just a few examples and can leverage past experience with concepts in one domain to learn new concepts in a different domain (Lake et al., 2011). In addition, humans can learn continuously with minimal forgetting of older concepts. In the quest to emulate this ability, the machine learning community has made spectacular advances in the areas of computer vision (Krizhevsky et al., 2012), natural language processing (Devlin et al., 2018), and speech recognition (Xiong et al., 2018). Despite this, to learn new concepts or tasks, conventional deep learning techniques:

- require thousands of training examples (e.g. training an image classifier on ImageNet to achieve state of the art accuracy requires 1300 labeled examples per class (Krizhevsky et al., 2012));
- overfit when only a few training examples are available (Goodfellow et al., 2016);
- have only basic ability for leveraging previous learning (e.g. transfer learning (Pratt et al., 1991) or fine-tuning (Yosinski et al., 2014)) and this is typically done at *training* time as opposed to post-training *deployment* time;
- exhibit catastrophic forgetting (i.e. the tendency for a neural network to suddenly lose knowledge it had already learned when learning new information (French, 1999));
- need long training times measured in hours or days, which result in high computation costs, large power consumption and carbon dioxide emmisions. For example, training the BERT model (Devlin et al., 2018) takes 79 hours using 64 NVIDIA V100 GPUs, costing up to USD12,571 of cloud computing time, utilizing an effective 1507 kilowatt-hours of electricity (about the same as used by the average UK citizen in 15 weeks) and emitting 1438 pounds of carbon dioxide (about the same emitted by the average UK citizen over a period of 5 weeks) (Strubell et al., 2019);

- are trained offline using specialized computing platforms (e.g. cloud data center, GPU cluster, etc.) as opposed to online on common computing devices; and
- have difficulty adding new classes or datasets in the classification setting without retraining.

On the other hand, we desire machine learning systems that are:

- *data efficient*: able to learn a new task from only a few examples without over-fitting, but be resistant to under-fitting if more data is available;
- *computation efficient*: able to learn a new task rapidly without retraining on common computing platforms (e.g. on a mobile device);
- make predictions in real-time in a power efficient manner on common computing platforms (e.g. on a mobile device);
- share knowledge across tasks;
- learn continuously, without forgetting.

In this thesis we aim to design, develop, analyze, and extend the theory of machine learning systems that are both *data efficient* and *computation efficient*. During training, our system is exposed to multiple tasks such that it "learns how to learn" (i.e. meta-learns) to solve new tasks from only a few examples. The system can effectively and efficiently solve new, unseen tasks from a broad range of data distributions, in both the low and high data regimes, without the need for costly retraining. Adapting to a new task requires only a forward pass of the example task data through the trained network making the learning of new tasks possible on mobile devices.

In particular, we focus on few-shot image classification systems. We argue that such systems require mechanisms that adapt to each task, and that these mechanisms should themselves be learned from a diversity of datasets and tasks at training time. The adaptation mechanism should affect the entire system, not just the classifier component and the resulting system should be adept at continual learning.

1.2 Overview and Main Contributions

This section provides an overview of the work contained in this thesis. For each topic, the main results are summarized and the publications that I co-authored with my supervisors and fellow students that resulted from the research are cited. My personal contributions to each topic are listed at the beginning of the respective chapters.

VERSA: Meta-Learning Probabilistic Inference For Prediction Chapter 3 introduces a new framework for data efficient, computation efficient, and versatile learning. Specifically: (i) We develop ML-PIP, a general framework for Meta-Learning approximate Probabilistic Inference for Prediction. ML-PIP extends existing probabilistic interpretations of meta-learning to cover a broad class of methods. (ii) We introduce VERSA, an instance of the framework employing a fast, flexible and versatile amortization network that takes few-shot learning datasets as inputs, with arbitrary numbers of training examples, and outputs a distribution over task-specific parameters in a single forward pass of the network. VERSA substitutes optimization at test time with forward passes through inference networks, amortizing the cost of inference and relieving the need for second derivatives during training. (iii) We evaluate VERSA on benchmark datasets where the method set state-of-the-art results (at the time), handles arbitrary numbers of training examples, and for classification, arbitrary numbers of classes at train and test time. The power of the approach is then demonstrated through a challenging few-shot ShapeNet view reconstruction task.

These results are joint work with my co-first author Jonathan Gordon, as well as Matthias Bauer, Sebastian Nowozin, and Richard Turner and were published as 'Meta-Learning Probabilistic Inference For Prediction' (Gordon et al., 2019) in the ICLR 2019 conference proceedings.

CNAPS: Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes Chapter 4 builds on the work in the Chapter 3 and adds a second amortized network to adapt key parameters in the feature extractor to the current task in addition to adapting the final layer classifier. To accomplish this, we introduce a conditional neural process based approach to the multi-task classification setting and establish connections to the meta-learning and few-shot learning literature. The resulting approach, called CNAPS, comprises a classifier whose parameters are modulated by an adaptation network that takes the current task's dataset as input. We demonstrate that CNAPS achieved state-of-the-art results (at the time) on the challenging META-DATASET benchmark indicating high-quality transfer-learning. We show that the approach is robust, avoiding both over-fitting in low-shot regimes and under-fitting in high-shot regimes. Timing experiments reveal that CNAPS is computationally efficient at test-time as it does not involve gradient based adaptation. Finally, we show that trained models are immediately deployable to continual learning and active learning where they can outperform existing approaches that do not leverage transfer learning.

These results are joint work with my co-first authors James Requeima and Jonathan Gordon, as well as Sebastian Nowozin, and Richard Turner and were published as 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes' (Requeima et al., 2019b) in the NeurIPS 2019 conference proceedings and was selected for a spotlight talk.

TASKNORM: Rethinking Batch Normalization for Meta-Learning Chapter 3 and Chapter 4 demonstrate that modern meta-learning approaches for image classification rely on increasingly deep networks to achieve state-of-the-art performance, making batch normalization an essential component of meta-learning pipelines. However, the hierarchical nature of the meta-learning setting presents several challenges that can render conventional batch normalization ineffective, giving rise to the need to rethink normalization in this setting. In Chapter 5, we evaluate a range of approaches to batch normalization for meta-learning scenarios, and develop a novel approach that we call TaskNorm. Experiments on fourteen datasets demonstrate that the choice of batch normalization has a dramatic effect on both classification accuracy and training time for both gradient based- and gradient-free meta-learning approaches. Importantly, TaskNorm is found to consistently improve performance. Finally, we provide a set of best practices for normalization that will allow fair comparison of meta-learning algorithms.

These results are joint work with my co-first author Jonathan Gordon, as well as James Requeima, Sebastian Nowozin, and Richard Turner and were published as 'TaskNorm: Rethinking Batch Normalization for Meta-Learning' in the ICML 2020 conference proceedings. (Bronskill et al., 2020).

1.3 List of Publications

The following provides a list of all publications I have co-authored in the course of pursuing my DPhil degree.

1.3.1 Conference Proceedings

Jonathan Gordon*, John Bronskill*, Matthias Bauer, Sebastian Nowozin, and Richard Turner (2019). 'Meta-Learning Probabilistic Inference for Prediction'. In: *Proceedings of the 7th International Conference on Learning Representations*. *equal contribution

James Requeima*, Jonathan Gordon*, John Bronskill*, Sebastian Nowozin, and Richard E. Turner (2019). 'Fast and flexible multi-task classification using conditional neural adaptive processes.' In: *Advances in Neural Information Processing Systems 32*, pages 7957–7968. This paper was selected for a spotlight presentation at the conference. *equal contribution

John Bronskill*, Jonathan Gordon*, James Requeima, Sebastian Nowozin, and Richard E. Turner (2020). 'TaskNorm: Rethinking Batch Normalization for Meta-Learning.' In *Proceedings of the 37th International Conference on Machine Learning*. *equal contribution

Daniela Massiceti, John Bronskill, Luisa M. Zintgraf, Sebastian Tschiatschek, Katja Hofmann and Cecily Morrison (2020). 'Meta-learning from Poor-Quality Videos for Personalised Object Recognition' Submitted to: *The 2020 European Conference on Computer Vision.*

1.3.2 Workshops

Jonathan Gordon*, John Bronskill*, Matthias Bauer, Sebastian Nowozin and Richard E. Turner (2018). ‘Decision-Theoretic Meta-Learning: Versatile and Efficient Amortization of Few-Shot Learning’ In: *International Workshop on Automatic Machine Learning (AutoML 2018) at Thirty-fifth International Conference on Machine Learning.* *equal contribution

Jonathan Gordon*, John Bronskill*, Matthias Bauer*, Sebastian Nowozin, and Richard E. Turner (2018). ‘Versa: Versatile and Efficient Few-shot Learning’. In: *Third Workshop on Bayesian Deep Learning at the 32nd Conference on Neural Information Processing Systems.* *equal contribution

Jonathan Gordon*, John Bronskill*, Matthias Bauer*, Sebastian Nowozin, and Richard E. Turner (2018). ‘Consolidating the Meta-Learning Zoo: A Unifying Perspective as Posterior Predictive Inference’. In: *Workshop on Meta-Learning (MetaLearn 2018) at the 32nd Conference on Neural Information Processing Systems.* *equal contribution

1.3.3 Source Code Repositories

John Bronskill, Jonathan Gordon, and Matthias Bauer (2019). Code for ‘Meta-Learning Probabilistic Inference for Prediction’. https://github.com/Gordonjo/versa

John Bronskill, Jonathan Gordon, and James Requeima (2019). Code for ‘Fast and flexible multi-task classification using conditional neural adaptive processes.’ and ‘TaskNorm: Rethinking Batch Normalization for Meta-Learning.’ https://github.com/cambridge-mlg/cnaps

Chapter 2

Background

In this chapter, we review the concepts and literature that form the basis of the research work that follows. First, we give a high-level overview of Meta-Learning, Multi-Task Learning and Transfer Learning in Section 2.1. In Section 2.2, we explain the concepts that underlie few-shot learning and then in Section 2.3, we survey the few-shot learning methods that are the most relevant to the work in this thesis. In Section 2.4, we summarize Neural Processes, a meta-learning approach that combines the benefits of Gaussian Processes which are probabilistic and data efficient with the computational efficiency of deep neural networks. We then provide brief introductions to Continual Learning and Active Learning and their relationship to few-shot learning methods in Section 2.5 and Section 2.6, respectively. Finally, we describe the primary datasets used to benchmark few-shot learning methods in Section 2.7.

2.1 Meta-Learning, Multi-Task Learning, and Transfer Learning

Meta-Learning, Multi-Task Learning, and Transfer Learning are all closely related topics that have a range of definitions depending on the specific domain. In the following, only the main ideas behind these concepts will be outlined with emphasis on their use in this work.

2.1.1 Meta-Learning

Conventional deep learning learning systems are normally trained to solve a specific prediction task and in doing so require copious data and computational resources (Krizhevsky et al., 2012). This can be problematic if training data or computing resources are scarce or impractical to obtain. A key concept throughout this thesis is meta-learning or *learning to learn*. In meta-learning, a machine learning model gains experience over multiple learning tasks and leverages this past cross-task experience to improve future learning ability. This ability to learn how to learn leads to benefits in both data and computation efficiency (Hospedales et al., 2020).

The concept of meta-learning originated in cognitive psychology (Maudsley, 1980). In computer science, early approaches to meta-learning include a system called *Shift To A Better Bias* (Utgoff, 1986), the variable bias management system (Rendell et al., 1987) which automatically selects between different learning algorithms, and the work by Schmidhuber (1987) that attempts to learn entire learning algorithms via genetic evolution. Since recurrent neural networks (RNNs) are universal computers, Schmidhuber (1993) showed how self-referential RNNs can learn to run their own weight change algorithm. In this work, we focus on more recent deep learning approaches to meta-learning.

Historically, machine learning models were learned based on hand-engineered features. In the deep learning era that followed, features were jointly learned with the model. Meta-learning can be viewed as the next step in this progression by jointly learning the features, model, and the learning algorithm itself (Hospedales et al., 2020). In the context of this thesis, we use the term meta-learning to refer to the ability of a system to generate or induce a learning algorithm to meet the demands of a novel machine learning task based on experience acquired from prior tasks.

2.1.2 Multi-Task Learning

The goal of multi-task learning (Caruana, 1997) is to improve learning efficiency and accuracy of prediction on a group of tasks by learning from the group of tasks in parallel, compared to training on each task separately. It accomplishes this by leveraging a shared representation between tasks i.e. what is learned for a single task will aid in the learning other tasks. Caruana (1997) shows that multi-task learning also improves generalization by leveraging the domain-specific information contained in the training signals of related tasks.

An example of this is a system to filter email that is customized for each user to only display email that is of importance to them (Edelen et al., 2019). However, some email will be important to many users, but other email will only be important for a small subset of users. When implementing such a system, one possibility is train a unique filtering model for each user in the system. Alternatively, multi-task learning could be employed to potentially improve overall accuracy across all users by leveraging various commonalities and differences across the user base. This is especially true if some email users do not receive much email and hence have little training data to contribute. Caruana (1997) shows reduction of prediction error by up to 30% on certain tasks using multi-task learning versus single-task learning.

Multi-task learning differs from meta-learning in that the goal of multi-task learning is to make predictions on a fixed number of tasks whereas the goal of meta-learning is to make predictions on future tasks that are not seen during training. In addition, multi-task learning employs a single level of optimization during learning while meta-learning also employs a higher level objective to learn across many tasks such that a novel task can be rapidly solved. However, in Chapter 3 and Chapter 4 we demonstrate that multi-task learning principles can aid in training neural networks that learn to rapidly generate classifier parameters for classifying novel tasks.

2.1.3 Transfer Learning

Transfer learning aims to transfer knowledge from one learning task to another related task. In particular, it is common to use a source task to improve learning on a target task by transferring some or all of the parameters from the source task to the target task (Hospedales et al., 2020).

For example, it is common to train a classification model on a large image dataset such as ImageNet (Russakovsky et al., 2015) and use the learned feature representations to aid classification on a different dataset such as CIFAR (Krizhevsky and Hinton, 2009). This process is often referred to as *fine tuning* since the transferred ImageNet features are fine-tuned with the CIFAR image data.

In an impressive success story, Bird et al. (2020) demonstrate that transfer learning between the Electroencephalographic (EEG) brainwave domain and the Electromyographic (EMG) muscular wave domain showed significant gains in classification accuracy. In particular, when the pre-trained weights from the EMG model are used to initialize training of the EEG model, classification accuracy increases to 92.83% from 62.37% using random initialization.

Transfer learning differs from meta-learning in that the information transferred via transfer learning is learned by conventional supervised learning on a source task without use of a higher-level meta-learning training objective that aims to learn across many tasks such that it can rapidly learn a new task. That said, in Chapter 4 we show how transfer learning can be used in conjunction with meta-learning to great effect by employing a pre-trained feature extractor and modulating it's activations with meta-learned weights.

2.2 Few-Shot Learning Fundamentals

This thesis is primarily concerned with developing data and computation efficient systems for multi-task, few-shot image classification and regression based on a meta-learning approach. In this section, we describe the key ingredients for few-shot learning: task, context and target sets, as well as a meta-learning training protocol, and a probabilistic view of meta-learning.

2.2.1 Tasks, Context and Target Sets

In the following, we consider the multi-task image classification scenario. Rather than a single, large dataset D, we assume access to a dataset $\mathcal{D} = \{\tau_t\}_{t=1}^{K}$ comprising a large number of training *tasks* τ_t, drawn i.i.d. from a distribution $p(\tau)$. The data for a task τ consists of a *context set* $D^\tau = \{(\boldsymbol{x}_n^\tau, \boldsymbol{y}_n^\tau)\}_{n=1}^{N_\tau}$ with N_τ elements with the inputs $\boldsymbol{x}_n^\tau$ and labels $\boldsymbol{y}_n^\tau$ observed, and a *target set* $T^\tau = \{(\boldsymbol{x}_m^{\tau*}, \boldsymbol{y}_m^{\tau*})\}_{m=1}^{M_\tau}$ with M_τ elements for which we wish to make predictions. Here the inputs $\boldsymbol{x}^{\tau*}$ are observed and the labels $\boldsymbol{y}^{\tau*}$ are only observed during training. The examples from a single task are assumed i.i.d., but examples across tasks are not. Note that the target set examples are drawn from the same set of labels as the examples in the context set. An example task is shown in Figure 2.1. In the few-shot learning literature, a task is also referred to as an episode. In the next section, a task-aware training regime termed *episodic* training will be described.

Context Set D_τ

$N_\tau = 8$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
$\boldsymbol{x}^\tau$								
$\boldsymbol{y}^\tau$	Stopwatch $c = 1$	digital clock $c = 2$	digital watch $c = 3$	Stopwatch $c = 1$	parking meter $c = 4$	digital clock $c = 2$	digital watch $c = 3$	digital clock $c = 2$

Target Set T_τ

$M_\tau = 4$	*1*	*2*	*3*	*4*
$\boldsymbol{x}^{\tau *}$				
$\boldsymbol{y}^{\tau *}$	parking meter $c = 4$	digital watch $c = 3$	stopwatch $c = 1$	digital clock $c = 2$

Number of classes: $C = 4$

Number of shots per class: $k_c = [2, 3, 2, 1]$

Number of queries per class: $q_c = [1, 1, 1, 1]$

Fig. 2.1 A sample meta-learning task consisting of a context set D_τ of size $N_\tau = 8$ and a target set T_τ of size $M_\tau = 4$. Note that the labels $\boldsymbol{y}^{\tau *}$ would only be observed during training. The elements of the context and target sets are drawn from four different classes - stopwatch, digital clock, digital watch, and parking meter. Another task may have different context and target sizes, different classes, a different number of classes, and different number of examples per class. Images are examples from ImageNet (Russakovsky et al., 2015).

2.2.2 Episodic Training

The majority of multi-task, few-shot classification methods that are based on meta-learning concepts use an *episodic* training and testing protocol introduced by Vinyals et al. (2016). The key principle behind this protocol is that the training (referred to as *meta-training*) and testing (referred to as *meta-testing*) conditions should match and the system should be meta-trained with a large number of few-shot learning tasks.

The episodic training and testing protocol is described in the steps below. Figure 2.2 depicts a generic meta-learning-based multi-task classification system that will be referred to in the various steps.

Meta-Training

1. **Create the context and target sets**. Draw a task τ from the training datatset $\mathcal{D}_{train}$ as follows.

    - Select the number of classes C or *way* of the classification task.

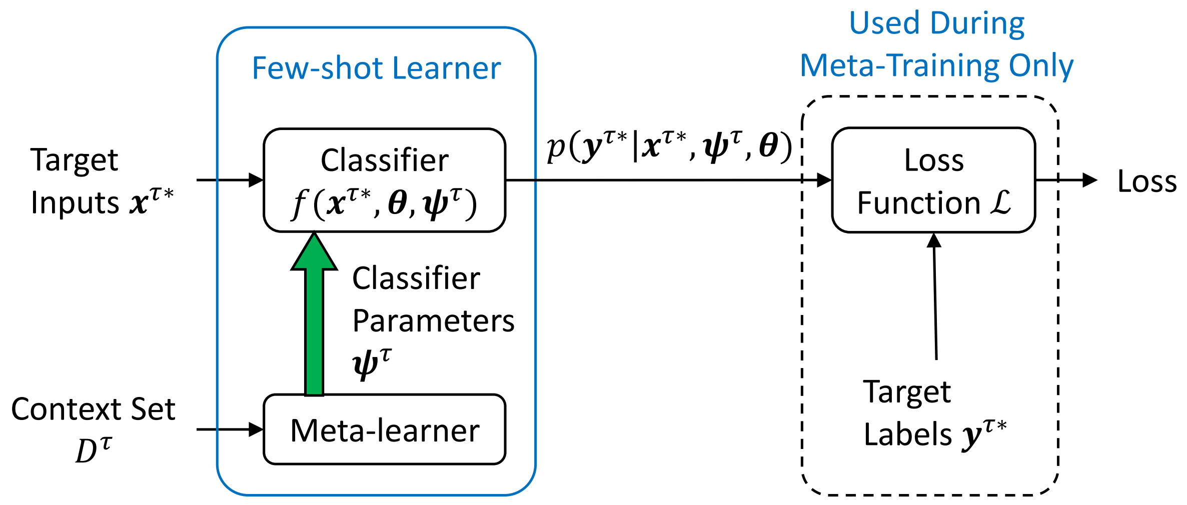

Few-shot Learner
Used During Meta-Training Only
Target Inputs $x^{\tau*}$
Classifier $f(x^{\tau*}, \theta, \psi^{\tau})$
$p(y^{\tau*}|x^{\tau*}, \psi^{\tau}, \theta)$
Loss Function $\mathcal{L}$
Loss
Classifier Parameters ψ^{τ}
Context Set D^{τ}
Meta-learner
Target Labels $y^{\tau*}$

Fig. 2.2 Episodic training. The meta-learner takes the context set D^{τ} as input and induces parameters for the classifier ψ^{τ} that are specific to task τ. In addition, $\boldsymbol{\theta}$ are parameters that are shared across all tasks. Given these parameters, the classifier can now make predictions $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \psi^{\tau}, \boldsymbol{\theta})$ on target inputs $\boldsymbol{x}^{\tau*}$. If in meta-training mode, the predictions and true labels $\boldsymbol{y}^{\tau*}$ are passed into a loss function and the computed loss can be used to improve the meta-learner parameters via back-propagation.

- Select the number of examples k_c or *shots* for each class $c \in C$ in the context set D^{τ}.
- Select the number of examples q_c or *queries* for each class $c \in C$ in the target set T^{τ}.
- Sample C classes from $\mathcal{D}_{train}$ without replacement.
- Form the context set D^{τ} by sampling without replacement from each class $c \in C$, k_c image inputs $\boldsymbol{x}^{\tau}$ with label $\boldsymbol{y}^{\tau} = c$.
- Form the target set T^{τ} by sampling without replacement from each class $c \in C$, q_c image inputs $\boldsymbol{x}^{\tau*}$ with label $\boldsymbol{y}^{\tau*} = c$.

2. **Induce a classifier**. Feed the context set D^{τ} as input to the meta-learner (see Figure 2.2). The goal of the meta-learner is to process D^{τ}, and produce a classifier model with parameters $\boldsymbol{\theta}$ that are shared across tasks and ψ^{τ} that are specific to task τ.
3. **Classify**. The resulting task-specific classifier f (see Figure 2.2) can now make predictions $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \psi^{\tau}, \boldsymbol{\theta})$ for any test inputs $\boldsymbol{x}^{\tau*} \in T^{\tau}$ associated with the task.
4. **Compute the loss and update the parameters**. The predictions $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \psi^{\tau}, \boldsymbol{\theta})$ at the true target labels $\boldsymbol{y}^{\tau*}$ are input to a loss function $\mathcal{L}$ where the output loss can be used to back-propagate through the networks and update the parameters of the meta-learner.
5. **Iterate.** The above steps are repeated until the loss stops decreasing or some other stopping criteria is met.

Meta-testing

1. Same as step 1 in Meta-Training except:
    - The task would be drawn from the test set $\mathcal{D}_{test}$.
    - The target set T^{τ} does not require labels $\boldsymbol{y}^{\tau*}$.
2. **Induce a classifier**. Same as step 2 in Meta-Training.
3. **Classify**. Same as step 3 in Meta-Training.
4. **Iterate**. Repeat for as many tasks that need to be classified.

Often, the assumption is that meta-test tasks will include classes that have not been seen during meta-training, and D^{τ} will contain only a few observations for each class. In the next section, we cast meta-learning in terms of a probabilistic model.

2.2.3 Hierarchical Probabilistic Modelling View of Meta-Learning

In the previous section, we described the computational flow for a meta-learning approach to multi-task, few-shot classification. In this section, we take a more formal perspective on the scenario.

A general and useful view of meta-learning is through the perspective of hierarchical probabilistic modelling (Heskes, 2000; Bakker and Heskes, 2003; Grant et al., 2018; Gordon et al., 2019). A standard graphical representation of this modelling approach is presented in Figure 2.3. Global parameters $\boldsymbol{\theta}$ encode information shared across all tasks, while local parameters $\boldsymbol{\psi}^{\tau}$ encode information specific to task τ. This model introduces a *hierarchy* of latent parameters, corresponding to the hierarchical nature of the data distribution. Let X^{τ} and Y^{τ} denote all the inputs and outputs (both context and target) for task τ. The joint probability of the outputs and task specific parameters for T tasks, given the inputs and global parameters is:

$$p\left(\{Y^{\tau}, \boldsymbol{\psi}^{\tau}\}_{\tau=1}^{\mathrm{T}} | \{X^{\tau}\}_{\tau=1}^{\mathrm{T}}, \boldsymbol{\theta}\right) = \prod_{\tau=1}^{\mathrm{T}} p\left(\boldsymbol{\psi}^{\tau}|\boldsymbol{\theta}\right) \prod_{n=1}^{N_{\tau}} p\left(\boldsymbol{y}_n^{\tau}|\boldsymbol{x}_n^{\tau}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}\right) \prod_{m=1}^{M_{\tau}} p\left(\boldsymbol{y}_m^{\tau*}|\boldsymbol{x}_m^{\tau*}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}\right).$$

The context set D^{τ} and $\boldsymbol{x}^{\tau*}$ are always observed and $\boldsymbol{y}^{\tau*}$ is observed in training. In this thesis, we focus primarily on the posterior predictive distribution $p\left(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, D^{\tau}, \boldsymbol{\theta}\right)$ of the model and we make the following assumptions:

1. $\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau} \perp\!\!\!\perp \boldsymbol{x}^{\tau*}$
2. $p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \perp\!\!\!\perp \boldsymbol{\psi}^{\tau}$
3. $y^{\tau*} \perp\!\!\!\perp D^{\tau}|\boldsymbol{\psi}^{\tau}$
4. $\boldsymbol{x}^{\tau*}$ only affects $\boldsymbol{\psi}^{\tau}$ when $\boldsymbol{y}^{\tau*}$ is observed.

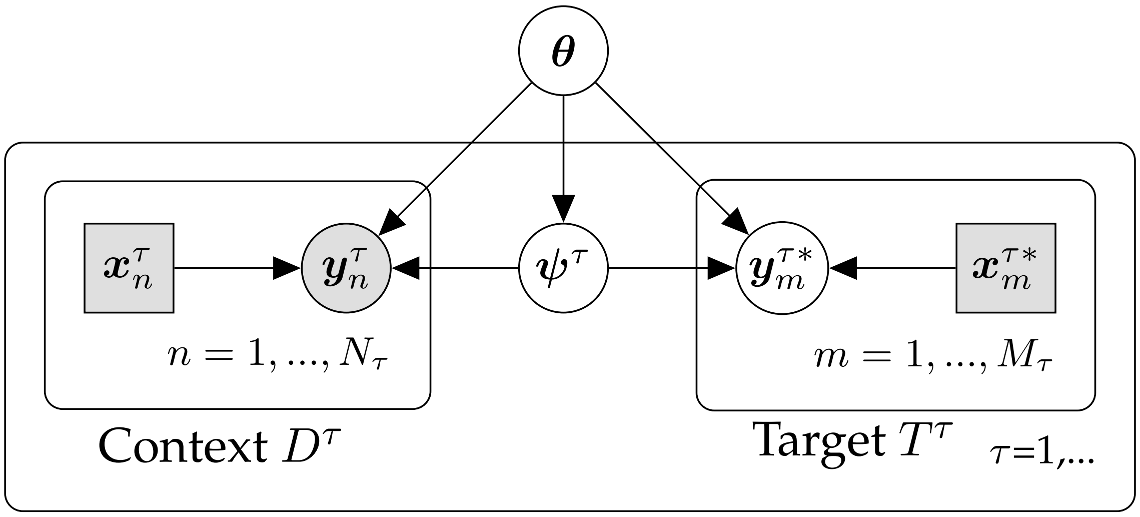


Fig. 2.3 Directed graphical model for multi-task meta-learning that employs shared parameters $\boldsymbol{\theta}$, that are common to all tasks, and task specific parameters $\{\boldsymbol{\psi}^\tau\}_{\tau=1}^{\mathrm{T}}$. The data for a task τ consists of a *context set* $D^\tau = \{(\boldsymbol{x}_n^\tau, \boldsymbol{y}_n^\tau)\}_{n=1}^{N_\tau}$ with N_τ elements with the inputs $\boldsymbol{x}_n^\tau$ and labels $\boldsymbol{y}_n^\tau$ observed, and a *target set* $T^\tau = \{(\boldsymbol{x}_m^{\tau *}, \boldsymbol{y}_m^{\tau *})\}_{m=1}^{M_\tau}$ with M_τ elements for which we wish to make predictions. Here the inputs $\boldsymbol{x}^{\tau *}$ are observed and the labels $\boldsymbol{y}^{\tau *}$ are only observed during training.

A general approach to meta-learning is to design inference procedures for the task-specific parameters $\boldsymbol{\psi}^\tau = \boldsymbol{\psi}_{\boldsymbol{\phi}}(D^\tau)$ conditioned on the context set (Grant et al., 2018; Gordon et al., 2019), where $\boldsymbol{\psi}$ is parameterized by additional parameters $\boldsymbol{\phi}$. Thus, a meta-learning algorithm defines a predictive distribution parameterized by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as $p\left(\boldsymbol{y}_m^{\tau *} | \boldsymbol{x}_m^{\tau *}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right), \boldsymbol{\theta}\right)$. This perspective relates to the *inner* and *outer* loops of meta-learning algorithms (Grant et al., 2018; Rajeswaran et al., 2019): the inner loop represents adaptation to a given task by using $\boldsymbol{\psi}_{\boldsymbol{\phi}}$ to generate the parameters $\boldsymbol{\psi}^\tau$ as a function of the context set; while the outer loop represents the meta-training objective by computing the loss between predictions $p(\boldsymbol{y}^{\tau *} | \boldsymbol{x}^{\tau *}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right), \boldsymbol{\theta})$ for target inputs $\boldsymbol{x}^{\tau *}$ and the true target outputs $\boldsymbol{y}^{\tau *}$ to update the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ that are shared across tasks.

During meta-training, a task τ is drawn from $p(\tau)$ and randomly split into a context set D^τ and target set T^τ. The meta-learning algorithm's inner-loop is then applied to the context set to produce $\boldsymbol{\psi}^\tau$. With $\boldsymbol{\theta}$ and $\boldsymbol{\psi}^\tau$, the algorithm can produce predictions for the target set inputs $\boldsymbol{x}_m^{\tau *}$. Given a differentiable loss function, and assuming that $\boldsymbol{\psi}_{\boldsymbol{\phi}}$ is also differentiable, the meta-learning algorithm can then be trained with stochastic gradient descent algorithms. Using log-likelihood as an example loss function, we may express a meta-learning objective for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \underset{p(\tau)}{\mathbb{E}} \left[\sum_{m=1}^{M_\tau} \log p\left(\boldsymbol{y}_m^{\tau *} | \boldsymbol{x}_m^{\tau *}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right), \boldsymbol{\theta}\right) \right] \text{ where } \{\{(\boldsymbol{x}_m^{\tau *}, \boldsymbol{y}_m^{\tau *})\}_{m=1}^{M_\tau}, D^\tau\} \sim p(\tau). \quad (2.1)$$

In the next section, we use this view to summarize a range of meta-learning approaches.

2.3 Meta-learning Methods

There has been an explosion of few-shot learning algorithms proposed in recent years. For in-depth reviews see Hospedales et al. (2020) and Wang et al. (2019). In this section, we will summarize the methods most relevant to this thesis. Typically, few-shot learning approaches are divided into roughly three main approaches including: (i) Gradient-based (ii) Metric Learning (iii) Amortization / Hypernets. However, in this section, we organize various few-shot image classification systems (including our own work) in terms of i) the choice of the parameterization of the classifier (and in particular the nature of the task-specific parameters), and ii) the function used to compute the task-specific parameters from the context set. This space is illustrated in Figure 2.4.

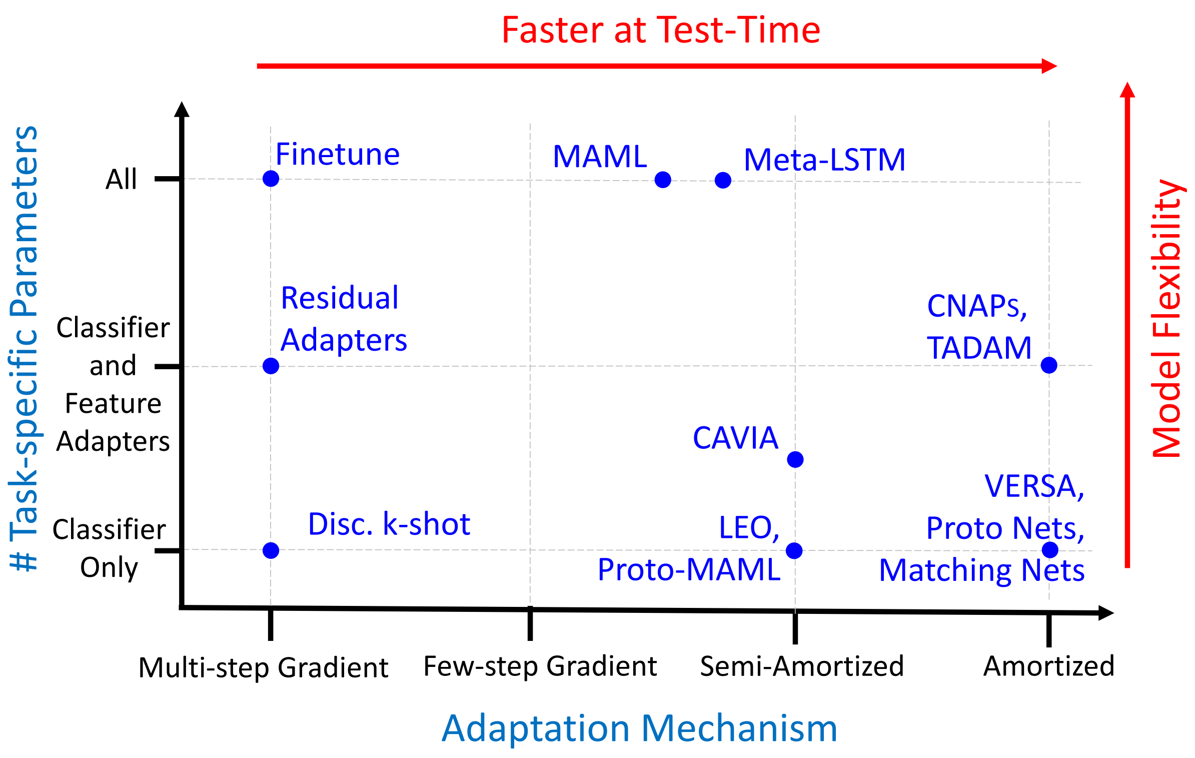


Fig. 2.4 Model design space. The y-axis represents the number of task-specific parameters. Increasing the number task-specific parameters increases model flexibility, but also the propensity to over-fit. The x-axis represents the complexity of the mechanism used to adapt the task-specific parameters to training data. On the right are *amortized* approaches (i.e. using fixed functions). On the left is gradient-based adaptation. Mixed approaches lie between. Computational efficiency increases to the right. Flexibility increases to the left, but with it over-fitting and need for hand tuning.

The choice of task-specific parameters ψ^τ. Clearly, any approach to multi-task classification must adapt, at the very least, the top-level classifier layer of the model. A number of successful models have proposed doing just this with e.g., metric-based approaches (Proto Nets (Snell et al., 2017)), variational inference (Disc. k-shot (Bauer et al., 2017)), or amortized networks (VERSA (Gordon et al., 2019) and Chapter 3). On the other end of the spectrum are models that adapt *all* the parameters of the classifier, e.g., MAML (Finn et al., 2017), Reptile (Nichol et al., 2018), Bayesian MAML (Yoon et al., 2018). The trade-off here is clear: as more parameters are adapted, the resulting model is more flexible, but also slower and more prone to over-fitting. For this reason, in this thesis we will explore methods that modulate only a small portion of the network parameters, following recent work on multi-task learning including Residual Adapters (Rebuffi et al., 2017, 2018), and FiLM layers (Perez et al., 2018).

We argue that just adapting the linear classification layer is sufficient when the task distribution is not diverse, as in the standard benchmarks used for few-shot classification (Omniglot (Lake et al., 2011) and *mini*ImageNet (Ravi and Larochelle, 2017)). However, when faced with a diverse set of tasks, such as that introduced recently by Triantafillou et al. (2020), it is important to adapt the feature extractor on a per-task basis as well.

The adaptation mechanism $\psi\left(D^\tau\right)$. Adaptation varies in the literature from performing full gradient descent learning with D^τ (e.g. Fine-tuning (Yosinski et al., 2014)) to relying on simple operations such as taking the mean of class-specific feature representations (Proto Nets, Matching Nets (Vinyals et al., 2016)). Recent work has focused on reducing the number of required gradient steps by learning a global initialization (MAML, Reptile) or additional parameters of the optimization procedure (Meta-LSTM (Ravi and Larochelle, 2017)). Gradient-based procedures have the benefit of being flexible, but are computationally demanding, and prone to over-fitting in the low-data regime. Another line of work has focused on learning neural networks to output the values of ψ, which we denote as *amortization* (VERSA). Amortization greatly reduces the cost of adaptation and enables sharing of global parameters, but may suffer from the amortization gap (Cremer et al., 2018) (i.e., underfitting), particularly in the large data regime. Recent work has proposed using semi-amortized inference (Proto-MAML Triantafillou et al. (2020), LEO (Rusu et al., 2018)), but have done so while only adapting the classification layer parameters. CNAPs (Requeima et al., 2019b) and Chapter 4 and TADAM (Oreshkin et al., 2018) use an amortized approach to adapt a key set of feature extractor parameters in addition to the classification layer parameters.

In the following, we group few-shot learning methods in terms of the adaptation mechanism, from multi-step gradient through to few-step gradient, and then amortized approaches. We'll then look at semi-amortized approaches that combine few-step gradient and amortization techniques. We will also cover an additional category that encompasses probabilistic methods that span the categorization above.

2.3.1 Multi-step Gradient Approaches

Multi-step gradient-based approaches for few-shot learning "fine tune" some or all of the parameters in a model to a particular task by taking as many gradient descent training steps using the context set data as necessary to optimize classification accuracy, while trying to avoid over-fitting.

Fine-tuning

A non-episodic baseline to measure meta-learning methods against is referred to as *fine-tuning* (Yosinski et al., 2014). Here we pre-train a feature extractor with shared parameters $\boldsymbol{\theta}$ off-line on all the classes of the entire meta-learning training set $\mathcal{D}_{train}$ (i.e. the union of all the training tasks). The final classifier layer of the pre-trained model is then removed and the remaining layers serve as an embedding function. Note that the meta-training phase is not required when fine-tuning. During meta-testing, for each task, the removed final layer is replaced with an untrained fully-connected layer with input size being equal to the embedding function output dimension and output size equal to the way of the task. The context set data from the test task is then used to train the new final classifier layer (and optionally to update the parameters of the pre-trained embedding function as well) using gradient descent or other optimization method. This network trained on the context set can now be evaluated using the target set data from the task.

Residual Adapters In addition to fine-tuning the final fully-connected classifier layer, Rebuffi et al. (2017) also fine-tune a small set of key parameters in the feature extractor. In particular, they devise two efficient parameterizations for modulating the output feature maps of 2D convolutional layers in residual neural networks (He et al., 2016) that they call *residual adapters*. The idea is to add additional task specific convolution and batch normalization layers with a small number of parameters to a standard residual layer. Figure 2.5 depicts both the series and parallel versions of a residual adapter. A compelling aspect is that the new parameterizations result in the majority of the weights of the residual layer being shared across tasks while less than 10% of the total weights in a layer are task specific, greatly reducing the number of parameters that need to be fine-tuned compared to adjusting all of the weights in the feature extractor network. The authors demonstrate excellent results on a "visual decathlon" challenge using this approach.

2.3.2 Few-step Gradient Approaches

Few-step gradient-based methods for few-shot learning attempt to learn a good initialization of model parameters such that only a few gradient steps on the context set data are required to fine tune the model to a particular task.

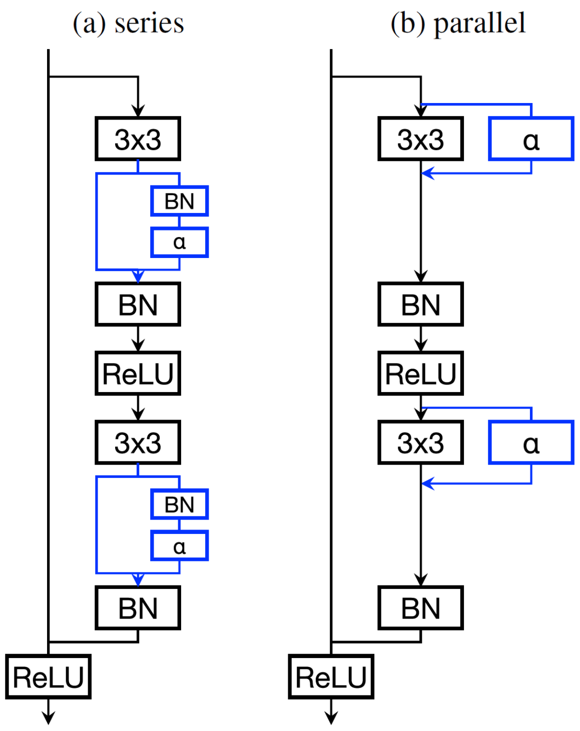


Fig. 2.5 Series (a) versus parallel (b) residual adapters. Standard residual neural network block components are shown in black and the added residual adapter components are shown in blue. Diagram from (Rebuffi et al., 2017)

Model Agnostic Meta-Learning (MAML)

Arguably, the most well-known few-shot learning method is a *gradient-based* approach called Model Agnostic Meta-Learning (MAML) (Finn et al., 2017). The basic idea is that a model is initially trained on a variety of tasks such that it can rapidly adapt to any new task by taking only a few additional gradient steps with only a small number of examples from the new task. In a nutshell, their method trains models that can easily be "fine-tuned" to a new task. In other words (referring to Figure 2.2 and Figure 2.3), MAML sets $\boldsymbol{\theta}$ to be the initialization of all the network parameters and the task specific parameters $\boldsymbol{\psi}^{\tau}$ are the network parameters after applying one or more gradient steps using the examples in the context set D^{τ}. More specifically, to meta-train MAML, let $\boldsymbol{\theta}$ be the (initially random) parameters of the classifier f. Then for each task, compute the task-specific classifier parameters $\boldsymbol{\psi}^{\tau}$ by taking one or more *inner* gradient steps on the context set data D^{τ} with step size α:

$$\boldsymbol{\psi}^{\tau} = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau}), \boldsymbol{y}^{\tau}) \tag{2.2}$$

Then, update the parameters $\boldsymbol{\theta}$ by taking a *meta* (or outer) gradient step with step size β on the target set with the classifier f using the task-specific parameters $\boldsymbol{\psi}^{\tau}$:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\psi}^{\tau}}(\boldsymbol{x}^{\tau *}), \boldsymbol{y}^{\tau *}) \tag{2.3}$$

Note Equation (2.3) requires calculating a gradient through a gradient since f uses the updated parameters $\boldsymbol{\psi}_{\tau}$, which can be computationally and memory intensive. There is a first-order approximation of MAML which performs almost as well with reduced computational complexity.

For meta-testing MAML, equation Equation (2.3) is not necessary and it is replaced with a simple forward pass through the classifier with the task-specific weights:

$$\boldsymbol{y}^{\tau *} = f_{\boldsymbol{\psi}^{\tau}}(\boldsymbol{x}^{\tau *}) \tag{2.4}$$

A key advantage of MAML is that it can work with any model which can be trained with gradient descent. MAML has been applied to classification, regression, and reinforcement learning problems with good success, though recent techniques have surpassed it on standard benchmarks. A downside of MAML is that it requires gradient steps even at meta-test time, which prohibits the use of MAML on mobile devices as common mobile deep learning frameworks currently do not support gradient back-propagation computation. In addition, the original variant of MAML does not provide distributions over parameters or predictions. In Section 2.3.5 we briefly describe some probabilistic variants of MAML, and in Chapter 3, we detail ML-PIP, a distributional model for making predictions that includes MAML as a special case.

Meta-LSTM

(Ravi and Larochelle, 2017) use a Long Short-Term Memory (LSTM) network to learn an optimizer to directly output the task-specific weight updates based on the context set alleviating the need to tune the gradient descent step-size.

Reptile

Nichol et al. (2018) devise an algorithm that is closely related to the first-order variant of MAML, but does not use episodic training (i.e. it does not require the task to be split into context and target sets). Reptile works by computing multiple gradient steps on a task, and moving the network weights $\boldsymbol{\theta}$ towards the fully trained weights on that task so that at test time, only a few gradient steps are required to adapt to the new task.

2.3.3 Amortization via Hypernetworks

A hypernetwork is a network that generates the weights for some or all of another network (Ha et al., 2016). The main challenge in generating the parameters for a high capacity neural network is the large number of weights involved. A fully connected layer with d inputs and k outputs has dk weights plus k biases and a convolutional layer with d input maps, k output maps, and filter kernel extent f has dkf^2 weights plus k biases. The key is to devise a parameterization such that the generation network needs to infer only a fraction of the total number of weights in the network.

More typically, in these methods, $\boldsymbol{\theta}$ parameterizes a shared feature extractor, and $\boldsymbol{\psi}$ a set of parameters used to *adapt* the network to a specific task, which include a linear classifier and possibly additional parameters of the network. These methods are often described as *amortized* as the meta-learner learns to learn how to generate classifier parameters incrementally during meta-training across a large number of training tasks. Note that a well known technique called Prototypical Networks (Snell et al., 2017) (described in detail later in this section) can be viewed in this way as: (i) it employs a shared feature extractor to embed inputs in a feature space; and (ii) the computation of the mean embedding vector for each class in the context set can be construed as a simple hypernetwork (with no parameters) that generates the task-specific classifier parameters.

In the following, we summarize the few-shot learning methods that employ amortization. Prior to that we review two technologies that play an important role in the implementation of amortization networks: deep sets and Feature-wise Linear Modulation (FiLM) layers.

Deep-Sets

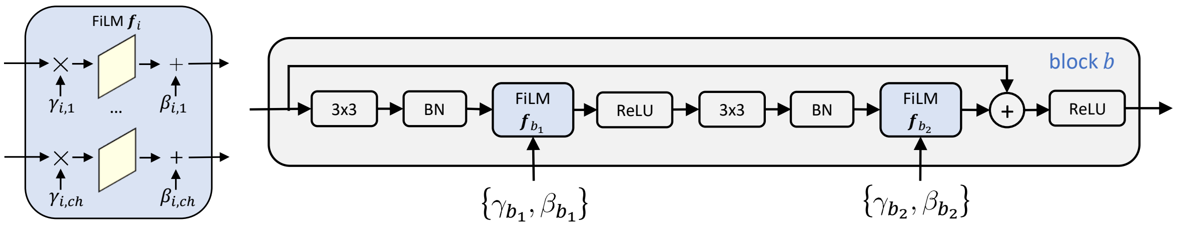
When processing or learning from a set of elements, it is desirable that the result of the processing or learning is invariant to the ordering of the elements in the set. Zaheer et al. (2017) formalizes the result that a function $f(S)$ acting on a set S will be invariant to a permutation in the elements of S (subject to some restrictions) if it can be decomposed into the form:

$$f(S) = \rho(\sum_{s \in S} \phi(s)) \tag{2.5}$$

where ρ and ϕ are transformations. In practice, ρ and ϕ can be arbitrary neural networks and the sum in Equation (2.5) can be replaced without loss of generality by a mean operation. Qi et al. (2017) prove (subject to some restrictions) that the sum may also be replaced by a $\max$ operator that takes an arbitrary number of vectors as input and returns a new vector of the element-wise maximum. This concept will prove to be extremely valuable to meta-learning systems as a way to *summarize* a dataset. Deep sets are a key component of many amortization networks to achieve invariance to the size and ordering of the context set.

FiLM Layers An effective and economical approach to adapting a convolutional neural network based image classifier to a specific task or dataset is to pretrain the network on a large dataset (such as ImageNet (Krizhevsky et al., 2012)), freeze the learned weights, and then insert *adapter* layers with a small number of parameters that will modulate the feature maps at the output of one or more of the convolutional layers in the network. The parameters for the adapter layers can be learned using gradient descent (or some other optimization algorithm) on the context set data or generated by a hypernet using the context set as input. The latter is the approach taken in this research (see Chapter 4).

A specific realization of this concept is a Feature-wise Linear Modulation (FiLM) layer (Perez et al., 2018) that scales and shifts the i^{th} feature map $\boldsymbol{f}_i$ of the the output of a 2D convolutional layer $\text{FiLM}(\boldsymbol{f}_i; \gamma_i, \beta_i) = \gamma_i \boldsymbol{f}_i + \beta_i$ using two parameters, γ_i and β_i. Figure 2.6a illustrates a FiLM layer operating on a convolutional layer, and Figure 2.6b illustrates how a FiLM layer can be added to a standard Residual network block (He et al., 2016). A key advantage of FiLM layers is that they enable expressive feature adaptation while adding even fewer parameters than are required for residual adapters.


FiLM $\boldsymbol{f}_i$
$\gamma_{i,1}$
...
$\beta_{i,1}$
$\gamma_{i,ch}$
$\beta_{i,ch}$
block b
3x3
BN
FiLM $\boldsymbol{f}_{b_1}$
ReLU
3x3
BN
FiLM $\boldsymbol{f}_{b_2}$
ReLU
$\{\gamma_{b_1}, \beta_{b_1}\}$
$\{\gamma_{b_2}, \beta_{b_2}\}$

(a) A FiLM layer. (b) A ResNet basic block with FiLM layers.

Fig. 2.6 (Left) A FiLM layer operating on convolutional feature maps indexed by channel *ch*. (Right) How a FiLM layer is used within a basic Residual network block (He et al., 2016).

We now summarize specific amortization-based few-shot learning methods.

Learnet

Bertinetto et al. (2016) describe a system for generating all the parameters of a feed forward neural network in a single shot. A *learnet* network is trained offline and learns to generate the parameters of a second *pupil* feed-forward network that is optimized for 1-shot learning. The key innovation is in the factorization of fully connected and convolutional layers such that there are drastically fewer parameters to predict in the pupil network. The factorization is analogous to singular value decomposition so that a learnet only needs to predict the diagonal elements of the fully connected weight matrix resulting in a reduction from dk weights to d. The same concept is extended to convolutional layers resulting in only df^2 weights to be generated instead of dkf^2.

HyperNetworks

Ha et al. (2016) describe a hypernetwork that is able to generate the weights of convolutional, residual, Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) networks with a dramatic reduction in the number of parameters that need to be generated. To generate a convolutional layer, the hypernetwork learns the weights of two fully-connected linear layers with m inputs as well as an embedding vector of size z such that the number of learnable weights is $z + md(z+1) + f^2k(m+1)$ compared to dkf^2. For image classification tasks, the hypernetwork approach leads to only a small increase in error rate when compared to a standard approach.

The following two methods, Prototypical Networks and Matching Networks, are normally categorized as *metric-learning* approaches. Metric learning approaches for few-shot classification typically map the context set for each task into an embedding space where similar datapoints are clustered together such that a non-parametric classifier that uses a pre-defined distance metric can separate out individual classes and predict their label correctly. However, in our schema, we see these as amortized methods as the generation of the prototypes by computing the mean of the embedding space vectors for each class of the context set can be viewed as an amortization network with no learnable parameters.

Prototypical Networks

Snell et al. (2017) approach few-shot classification by learning a non-linear clustering mapping $g_{\boldsymbol{\theta}}$ to an embedding space and then computing distances to prototype representations of each class. In particular, the prototype for each class c is simply the mean of the embedding vectors for each class in the context set D_τ of task τ:

$$\mathcal{P}_c = \frac{1}{|D_c^\tau|} \sum_{\boldsymbol{x}_i \in D_c^\tau} g_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right) \tag{2.6}$$

Classification of a new target data point $\boldsymbol{x}^{\tau *}$ is achieved by calculating a softmax over distances to the learned prototypes:

$$p(\boldsymbol{y}^{\tau *} = c | \boldsymbol{x}^{\tau *}, \boldsymbol{\theta}) = \frac{\exp(-d(g_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau *}), \mathcal{P}_c))}{\sum_{c'} \exp(-d(g_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau *}), \mathcal{P}_{c'}))} \tag{2.7}$$

where d is the euclidean distance function. Figure 2.7 is a simple depiction of Equation (2.7) for two-way classification. The objective for learning is to minimize the negative log-probability of the actual class c via gradient descent. Note that other distance functions such as the cosine or Mahalanobis distance could be used in place of the euclidean distance for d.

In the context of Figure 2.2 and Figure 2.3, the parameters θ that are shared across tasks are the parameters of the embedding network g and the parameters ψ are the computed prototypes $\mathcal{P}_c$ and f is a nearest neighbor classifier.

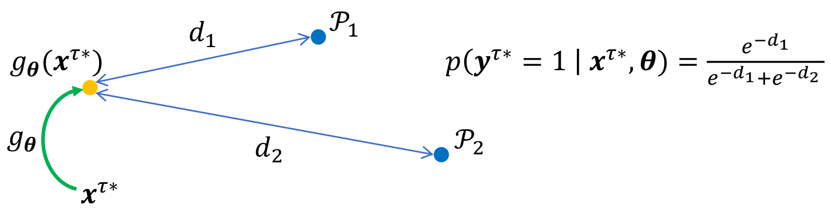

d_1
$\mathcal{P}_1$
$g_{\theta}(\boldsymbol{x}^{\tau *})$
$p(\boldsymbol{y}^{\tau *}=1 \mid \boldsymbol{x}^{\tau *}, \boldsymbol{\theta})=\frac{e^{-d_1}}{e^{-d_1}+e^{-d_2}}$
g_{θ}
d_2
$\mathcal{P}_2$
$\boldsymbol{x}^{\tau *}$

Fig. 2.7 Two-way classification of an image $\boldsymbol{x}^{\tau *}$ using Prototypical Networks. The image $\boldsymbol{x}^{\tau *}$ is first mapped through the feature extractor $g_{\boldsymbol{\theta}}$. Next the distance from the each of the two class prototypes $\mathcal{P}_1$ and $\mathcal{P}_2$ to the mapped point $g_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau *})$ is computed and finally the probability of the image belonging to class 1 can be calculated as shown.

As the classification stage in prototypical networks is non-parametric, it is possible to train the model using a certain number of shots per class and way and test the model with different values of shot and way, which adds significant flexibility. The authors found that training with a higher value of way than was used for testing led to superior classification accuracy. In their experiments, they tune the value used for way during training based on validation set results. They hypothesize that training on higher way forces the model to generalize better. However, they also found that there was no advantage to using a different value for shot in training versus testing.

Interestingly, prototypical networks can be shown to be equivalent to a linear classifier with weights $\mathbf{W}_{c,\cdot} = 2\mathcal{P}_c$ and biases $b_c = -\|\mathcal{P}_c\|^2$ (Snell et al., 2017; Triantafillou et al., 2020). Prototypical Networks can also be viewed as a simple deep sets system - compare the form of the prototype computation in Equation (2.6) to that of deep sets in Equation (2.5).

In summary, prototypical networks are a simple, flexible, efficient approach which achieves state of the art results on certain few-shot learning benchmarks.

Matching Networks

The matching networks model (Vinyals et al., 2016) is conceptually similar to prototypical networks in that a neural network maps examples into a non-linear embedding space. However, instead of using a euclidean distance based nearest neighbor classifier, matching networks use a weighted nearest neighbor classifier where an LSTM-based attention mechanism determines the weights. When the number of shots is equal to one, prototypical networks and matching networks are equivalent. When the number of shots is greater than one, the two approaches differ due to the attention mechanism in matching networks. The approaches also differ in that prototypical networks use a euclidean distance metric while matching networks use a cosine distance metric. Snell et al. (2017) demonstrate superior accuracy results using a euclidean distance in both approaches.

Few-Shot Image Recognition by Predicting Parameters from Activations

In an approach that has many similarities to Prototypical Networks, Qiao et al. (2018) learn a mapping ϕ from the activations in a pre-trained neural network to the weights of a final layer classifier such that the system is capable of high classification accuracy on both the classes it has been pre-trained on and new classes for which it has seen only a few examples.

More formally (Qiao et al., 2018), let $g_{\boldsymbol{\theta}}(\boldsymbol{x})$ be the activations before the fully connected layer for a training example image $\boldsymbol{x}$ on a pre-trained feature extraction network with parameters $\boldsymbol{\theta}$. If $\mathbf{W}_c$ are the fully-connected layer weights for class c and $\mathcal{P}_c$ (refer to Equation (2.6)) are the mean activations for the class c, then $\phi :\rightarrow \mathcal{P}_c \mathbf{W}_c$. Note that the 1D weight vectors $\mathbf{W}_c$ for each class c can be learned independently and subsequently concatenated with each other to form the complete 2D fully connected layer weight matrix. ϕ is learned by minimizing the loss:

$$\mathcal{L}(\phi) = \sum_{(c,x)\in D} \left[-\phi(\mathcal{P}_c) g_{\boldsymbol{\theta}}(\boldsymbol{x}) + \log \sum_{c'\in C} e^{\phi(\mathcal{P}_{c'}) g_{\boldsymbol{\theta}}(\boldsymbol{x})} \right] + \lambda \|\phi\| \tag{2.8}$$

where D is a large training set with many examples of each of the classes C and $\|\phi\|$ is a regularizer with weight λ. Once trained, this mapping was shown to work well for data from unseen classes in a single forward pass at test time. The activation statistic is not restricted to being the mean - the authors found that the maximum activation was the most effective for unseen classes. Note that this system is also extremely flexible in that a different number of classes can be used at test time due to the context independent construction of the fully-connected classification layer and it is also independent of the number of shots at test time due to that fact that an aggregate statistic of the activations is used as opposed to an activation vector of fixed size.

TADAM: Task dependent adaptive metric for improved few-shot learning

Oreshkin et al. (2018) propose a system using a pre-trained convolutional feature extractor and a prototypical networks based task-specific classifier. The distinguishing feature in the system is a hypernet that takes the context set as input and outputs FiLM layer (Perez et al., 2018) parameters that modulate each of the feature maps that result from the convolutional layers in the feature extractor. The modulation of the activations in the feature extractor allow the pre-trained feature extractor to adapt to the current task, enhancing classification accuracy.

2.3.4 Semi-amortized Approaches

In the context of few-shot learning, semi-amortized methods refer to systems that have some inference parameters that are shared / amortized across many tasks and others that require optimization for each task. In reality, MAML could be construed as semi-amortized as the inner learning rate and the initial network parameters $\boldsymbol{\theta}$ before each inner optimization step are shared across tasks. That is why Figure 2.4 depicts MAML to be between few-shot gradient and semi-amortized adaptation approaches. Below we summarize few-shot leaning methods that more fully combine amortization networks and few-step gradient optimization to adapt parameters to each task.

Proto-MAML

Proto-MAML (Triantafillou et al., 2020) combines the best of both MAML and Prototypical Networks and demonstrates superior classification accuracy to either of the component methods. Given that the task specific layer of prototypical networks can be expressed as a linear classifier, the Proto-MAML task-specific layer is initialized from the prototypical networks weights and then these weights are updated with gradient steps on the context set using the MAML algorithm.

Meta-learning with latent embedding optimization (LEO)

Like MAML, LEO (Rusu et al., 2018) is fundamentally a gradient-based learning technique. However, it differs from MAML in that the gradient steps are not taken in weight-space, but in a learned lower dimensional latent space from which the task-specific parameters are generated. The latent space takes as input an encoded version of the context set and outputs a distribution over classifier weights in a class conditional manner. The system uses a pre-trained feature extractor whose weights are fixed and the task-specific gradient optimization in the latent space only affects the top level classifier parameters.

Fast Context Adaptation via Meta-Learning (CAVIA)

CAVIA (Zintgraf et al., 2018) is also a fundamentally gradient-based learning technique. It differs from MAML in that only a small number of task-specific parameters ψ are updated at meta-test time as opposed to MAML where all model parameters are updated. The parameters ψ are extra inputs to the model that modulate its behavior in a task-specific manner. The inner gradient steps only update ψ and the outer meta-gradient updates the remaining parameters $\boldsymbol{\theta}$ in the model that are shared across tasks. Given that only a small number of parameters are updated in the inner gradient loop, CAVIA is much less prone to over-fitting compared to MAML.

2.3.5 Probabilistic Methods

Probabilistic approaches to few-shot learning span various methods for adapting the task specific parameters of a meta-learning system, but we group them together here. Before summarizing various probabilistic methods, we first provide a brief overview of variational inference.

Variational Inference

While not restricted for use in few-shot learning problems, the use of variational inference (Zhang et al., 2017) plays an important part in many high-scale, probabilistic few-shot learning systems. In a nutshell, variational inference is able to convert an intractable Bayesian inference problem into an optimization problem by approximating the true posterior probability distribution by a simpler, parametric distribution. In probabilistic few shot learning, there are many inference problems over a large number tasks, each with a small number of training examples, and it is often impractical to use variational inference to solve each one individually. However, recent advances in amortized variational inference (Kingma and Welling, 2014; Rezende et al., 2014) have brought great power and efficiency to variational inference by learning the parameters of a single neural network (often called a recognition model) to predict each of the unknown variables.

Probabilistic MAML

Finn et al. (2018) extend MAML using variational inference in order to sample a model for a new task from a model distribution. The goal of their meta-training phase is to learn a Gaussian prior specific to each task τ over parameters $\boldsymbol{\theta}$ with shared means $\boldsymbol{\mu_\theta}$ and variances $\boldsymbol{\sigma}_\theta^2$. During the meta-testing phase, an initial set of parameters $\boldsymbol{\theta}_\tau$ are sampled from the learned prior and then are used to compute the gradient on the loss with respect to the context set D_τ. This gradient is then used to compute the parameters adapted to the specific task.

Bayesian MAML

Yoon et al. (2018) take a very different approach to evolving MAML towards a probabilistic framework. Bayesian MAML essentially replaces the inner gradient descent step in standard MAML with Stein Variational Gradient Descent (SVGD) (Liu and Wang, 2016). SVGD is a non-parametric, sampling based variational inference method which allows the approximating distribution to be arbitrary. To obtain M samples from a target from SVGD, M instances of the model parameters (called particles) need to be stored and updated. Bayesian MAML is essentially an ensemble approach with each set of particles representing a sample of weights from the target distribution. If the number of particles is one, Bayesian MAML reverts to standard MAML. Classification accuracy increases with the number of particles, however this comes with the cost of storing additional sets of model parameters. In addition, SVGD performance is known to degrade as the dimensionality of the problem increases (Zhuo et al., 2017), limiting its usefulness in high dimensional scenarios.

Discriminative k-Shot Learning Using Probabilistic Models

Bauer et al. (2017) employs a fixed pre-trained feature extraction network with parameters shared across tasks $\boldsymbol{\theta}$ and learn a mapping between feature activations and top-layer classifier weights in a context independent manner, however the authors take a probabilistic approach to the learning the weights. The top-level weights used in each few-shot classification task are generated by an inference network that utilizes "conceptual information" from pre-trained classes as a prior along with the likelihood of the context set. When coupled with a pre-trained, high capacity feature extraction network such as a ResNet-34 (He et al., 2016), the system achieves excellent classification accuracy results.

2.4 Neural Processes

A Neural Process (NP) (Garnelo et al., 2018a,b) is a scalable and expressive new class of meta-learning model that makes predictions on sets of inputs. The goal of neural processes is to combine the benefits of Gaussian Processes (Rasmussen and Williams, 2005) that utilize prior knowledge to make inferences with the computational efficiency of neural networks. With reference to the previous section on categorizing few-shot learning methods, neural processes can be viewed as an amortized approach. In a neural process, a context set is fed through a deep sets type network and the pooled result is then passed as a context specific parameter into a model that makes predictions. Neural processes emerged concurrently with and are intimately related to the the work in this thesis.

2.4.1 Conditional Neural Processes

The graphical model for a Conditional Neural Process (CNP) is shown on the left side of Figure 2.8 and the computational flow is shown in Figure 2.9. Referring to Figure 2.9, the CNP can be decomposed into an encoder and a decoder. The encoder takes as input the context set D^τ and passes each member of the set through an embedding network with parameters $\boldsymbol{\phi}$. The resulting embeddings from each context set member $r_1, r_2, ..., r_{N_\tau}$ are then combined or *pooled* with a summation or mean operation to produce a representation of the context set $\boldsymbol{r}$. Since the encoder stage takes as input a set, it is often referred to as a *set encoder*. The decoder takes the context set representation $\boldsymbol{r}$, along with the target set inputs $\boldsymbol{x}_1^{\tau *}, \boldsymbol{x}_2^{\tau *}, ...\boldsymbol{x}_{M_\tau}^{\tau *}$ as inputs to a second network with parameters ρ to produce the target set predictions $\boldsymbol{y}_1^{\tau *}, \boldsymbol{y}_2^{\tau *}, ...\boldsymbol{y}_{M_\tau}^{\tau *}$. Note that this architecture is built directly on the deep sets concept (refer to Equation (2.5)). The deep sets connection allows the encoder to handle input context sets of arbitrary size and ensures permutation invariance of the context set members. The CNP target outputs can be stochastic by generating mean and variance outputs from each that can be sampled from and provide a measure of uncertainty. CNPs can be used for both regression and classification, though the original approach to CNP classification has limitations that will be discussed and overcome in Chapter 4.

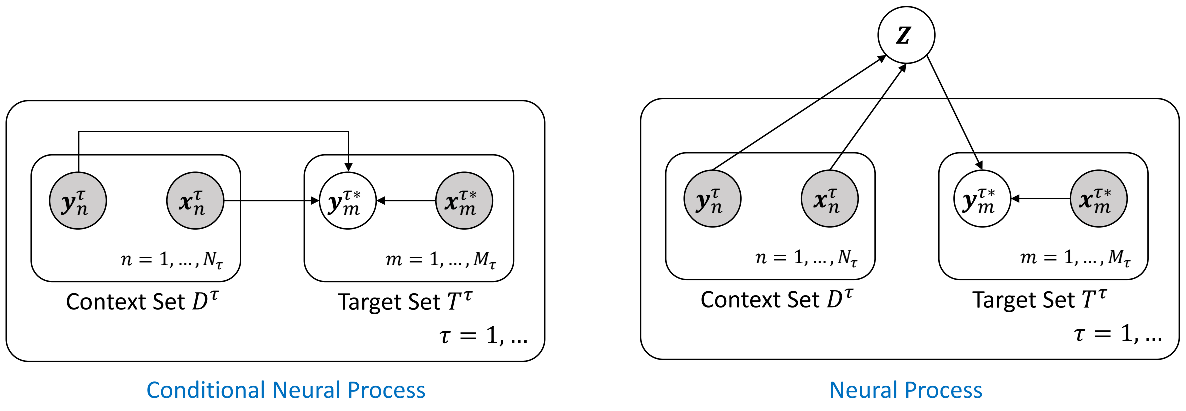


Fig. 2.8 (Left) Graphical model for a conditional neural process. (Right) Graphical model for a neural process with latent variable z.

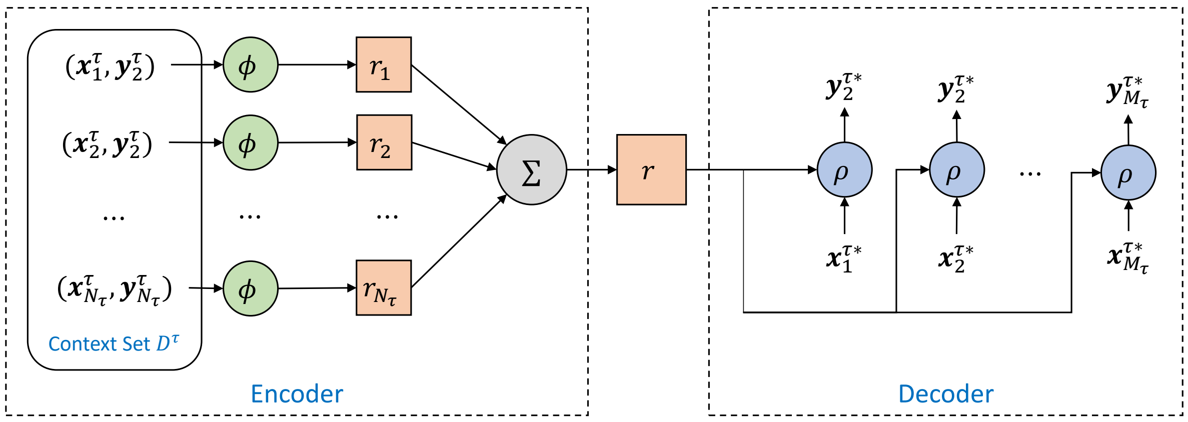


Fig. 2.9 Conditional neural process computational flow.

2.4.2 Neural Processes

A NP is a generalization of a CNP. The primary difference is that a NP adds a global (i.e. shared across tasks) stochastic latent variable z that can be sampled from (refer to Figure 2.8). CNPs make independent predictions at each target input and do not represent uncertainty in the underlying function, whereas the the additional latent variable z in a NP allows for this with better uncertainty estimates as a result.

2.4.3 Generative Query Networks for View Reconstruction

There has been much work in the computer vision research community to reconstruct unseen views or 3-Dimensional (3D) models of a scene from one or more photographs of that scene taken from different viewpoints. See Figure 2.10 for an example (Furukawa et al., 2015). Historically, the methods used to solve these problems have been geometry based - see Hartley and Zisserman (2003) for an excellent overview. More recently, deep learning approaches have been used as solutions to the reconstruction problems that implicitly learn about scene geometry and appearance.

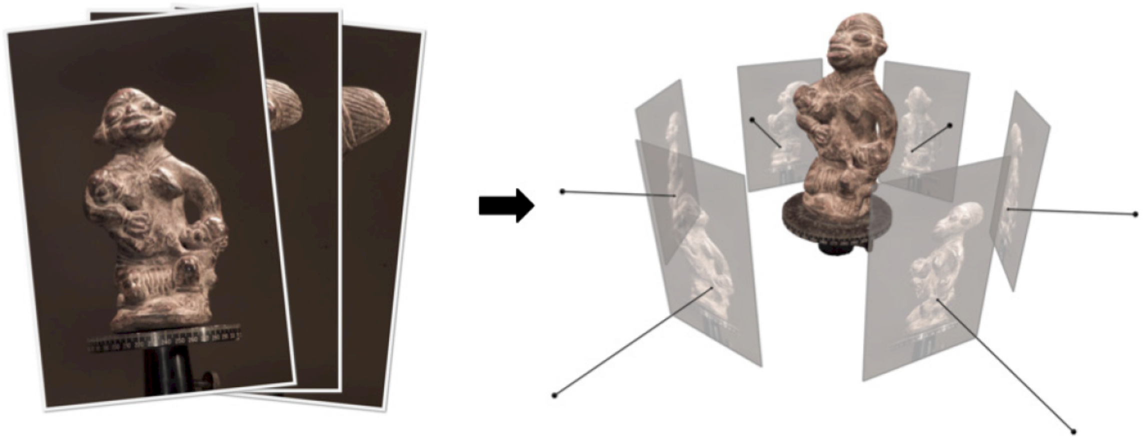


Fig. 2.10 3D model reconstruction from a set of photographs. From Furukawa et al. (2015).

Figure 2.11 shows how view reconstruction can be formulated as a meta-learning regression problem: (i) The context set D^τ for a task τ comprises a small set of viewpoints $\boldsymbol{x}^\tau$ (which may be a viewing angle or position in 2D or 3D space) and images $\boldsymbol{y}^\tau$ that correspond to the rendered 2D views from the viewpoints; (ii) The meta-learner takes as input the context set and generates a set of task-specific view parameters $\boldsymbol{\psi}^\tau$; (iii) The view generator takes the task-specific view parameters and a set of novel viewpoints $\boldsymbol{x}^{\tau *}$ as input along with $\boldsymbol{\theta}$ (the parameters shared across tasks) and generates a predicted view $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta})$; (iv) During meta-training, the predicted view and the ground truth view $\boldsymbol{y}^{\tau *}$ are fed into a loss function $\mathcal{L}$ and the computed loss is then back-propagated through the view generator and meta-learner so that their parameters can be improved; (v) The system is trained on many tasks until the loss no longer decreases at which point the system should generate high fidelity views from unseen viewpoints. In the following, we will summarize how NPs can be used for view reconstruction.

Eslami et al. (2018) introduce the Generative Query Network (GQN) which given a context set of images of a simple 3D scene taken from various viewpoints creates an internal representation of the scene $\boldsymbol{r}$ and can subsequently infer one or more target images of the scene taken from new viewpoints. A GQN is essentially an elaborate NP. Figure 2.12 is a conceptual

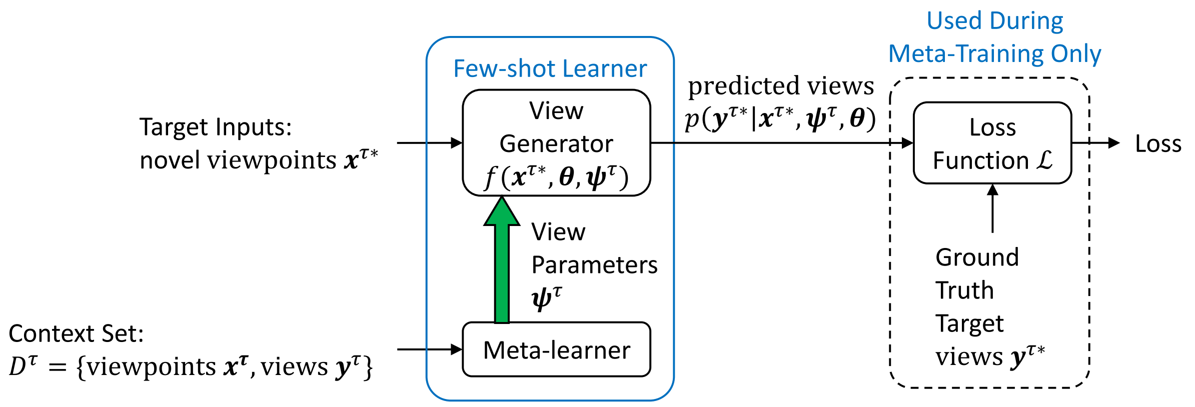

Used During
Meta-Training Only
Few-shot Learner
Target Inputs:
novel viewpoints $\boldsymbol{x}^{\tau*}$
View
Generator
$f(\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, \boldsymbol{\psi}^{\tau})$
predicted views
$p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta})$
Loss
Function $\mathcal{L}$
Loss
View
Parameters
$\boldsymbol{\psi}^{\tau}$
Ground
Truth
Target
views $\boldsymbol{y}^{\tau*}$
Context Set:
$D^{\tau} = \{\text{viewpoints } \boldsymbol{x}^{\tau}, \text{views } \boldsymbol{y}^{\tau}\}$
Meta-learner

Fig. 2.11 Meta-learning perspective of view reconstruction. The meta-learner takes the context set D^{τ} consisting of viewpoints $\boldsymbol{x}^{\tau}$ and corresponding views $\boldsymbol{y}^{\tau}$ as input and induces parameters ψ^{τ} for the view generator that are specific to task τ. In addition, $\boldsymbol{\theta}$ are parameters of the view generator that are shared across all tasks. Given these parameters, the view generator can now predict views $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \psi^{\tau}, \boldsymbol{\theta})$ on target inputs consisting of novel viewpoints $\boldsymbol{x}^{\tau*}$. If in meta-training mode, the predictions and ground truth views $\boldsymbol{y}^{\tau*}$ are passed into a loss function $\mathcal{L}$ (e.g. mean squared error between the predicted and ground truth views) and the computed loss can be used to improve the model parameters via back-propagation.

flow diagram of the GQN system extracted from (Eslami et al., 2018). Referring to Figure 2.12: the left side **A** refers to the context set (denoted with Observations v_i); the central part in **B** pools the context set with a deep sets style network (denoted with f) to derive the scene representation $\boldsymbol{r}$; and finally the right side is the decoder (denoted as the Generation network g) which generates novel target views using a recurrent neural network (RNN) with layers $h_1, h_2, \ldots h_L$ and input from the scene representation $\boldsymbol{r}$, the latent variable $\boldsymbol{z}$, and a target input (denoted as Query). The system is computationally intensive, but generates impressive results.

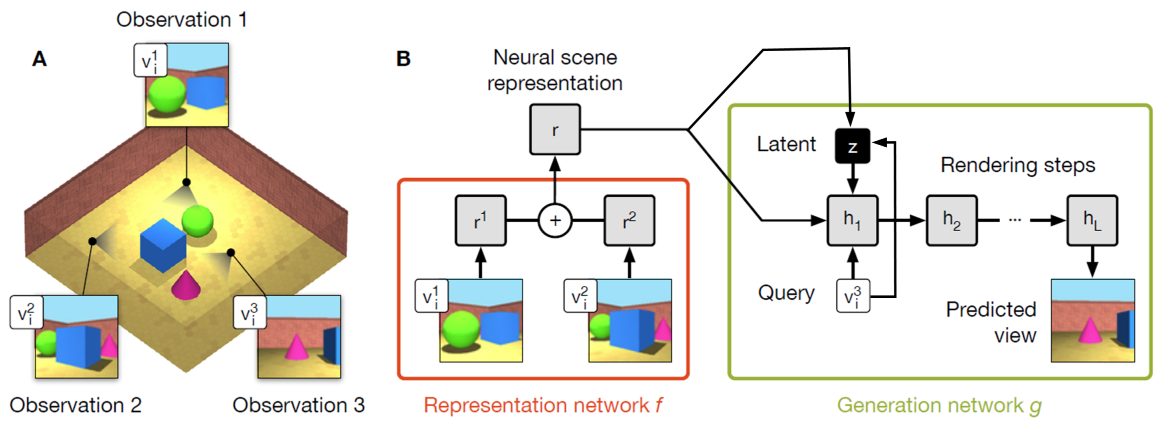


Fig. 2.12 GQN concept and flow diagram. From Eslami et al. (2018) with alterations from Turner (2018).

In Chapter 3, we describe a view reconstruction method that is essentially a CNP model, but our model was published first, though developed concurrently with CNPs.

2.5 Continual Learning and Amortized Inference Methods

Continual or lifelong learning (Ring, 1997) is the ability of a machine learning model to continuously learn from new data, building on past learning, and without forgetting what it has learned in the past. Schwarz et al. (2018) compile a clear desiderata for a continual learning method: (i) It should not catastrophically forget, i.e. it should perform well on previously learned tasks. (ii) It should exhibit *positive forward transfer*, i.e. it should learn new tasks with better performance by building on previously learned tasks. (iii) It should be scalable to a large number of tasks. (iv) It should exhibit *positive backward transfer*, i.e. the performance on previous tasks should improve as it learns new tasks. (v) It should be able to learn without task labels.

Well known approaches for continual learning include Progress and Compress (Schwarz et al., 2018), Elastic Weight Consolidation (Kirkpatrick et al., 2017), Variational Continual Learning (Nguyen et al., 2017), and Riemannian Walk (Chaudhry et al., 2018).

Online learning (Saad, 2009; Hoi et al., 2018) is a simpler form of continual learning where the learner must continuously make predictions based on a small number of data points arriving in a streaming fashion. Meta-trained few-shot learners are well suited to this task as they are trained adapt to a new task with a small context size and make predictions based on that.

Meta-learning can be used as an approach to continual learning (Hospedales et al., 2020). For example, a sequence of meta-training tasks can be designed such that the context set contains only a single new prediction problem, whereas the target set is drawn from all prediction problems seen up until now. This allows a meta-learner to learn new tasks but also encourage it to perform well on previously seen problems.

Few-shot classification methods that employ amortization networks (in particular neural process based methods) can be directly applied to continual learning scenarios since: (i) like any other few-shot learning system they are meta-trained to adapt to new task data; and (ii) its deep sets based amortization network can generate a compact summary of a context set categorized by class, that when persisted can serve as a simple memory of the class data seen to date which can be easily updated as new data is seen.

In Chapter 4 we show that our meta-trained few-shot image classification system performs competitively on standard continual learning benchmarks "out of the box" with no specific training for the continual learning scenario. It does this by effectively adapting to the new tasks presented, along with a small memory component to aid in retaining previous task data.

2.6 Active Learning and Few-shot Learning

Active learning (Cohn et al., 1996; Settles, 2012) is a specific case of machine learning where the learning algorithm has the ability to choose its own training data. The assumption is that the learner has access to a large pool of unlabeled training data and it can select which training inputs that it would like to have labeled by an *oracle* (i.e. a human expert). The goal is to choose the data points that will be the most informative to the learner such that it can make good predictions without having to label a large number of training points (with the assumption that labelling a data point is expensive by some measure). Effective active learning systems require: (i) their predictions to have well-calibrated uncertainty/probability estimates; (ii) the ability to update their parameters incrementally and sequentially.

Due to the requirement to update parameters incrementally based on a small number of data points, active learning systems can be viewed as closely related to meta-learning systems. In particular, a meta-learning view of active learning is as follows (Hospedales et al., 2020): the inner loop is considered to be the conventional supervised learning task (e.g. classification) that uses the currently labeled datapoints as the context set, and the unlabeled pool of datapoints as the target inputs; and the outer loop learns to select the best datapoints to label next.

In Chapter 4, we demonstrate that our meta-learning approach to few-shot image classification improves significantly over random acquisition of data points to label without any special training for active learning indicating that our model produces high quality probability predictions.

2.7 Datasets for Few-Shot Classification

This section describes the standard datasets for evaluating few-shot learning methods. We start by describing Omniglot, a small and relatively simple dataset of hand drawn characters. In the standard train-validation-test dataset split for Omniglot, the validation and test sets contain the same character classes as the train set, and require only a minimal amount of generalization by the few-shot learning method being evaluated. Next is a more challenging dataset of natural images called *mini*ImageNet. The dataset is larger as is the size of the constituent images. The validation and test sets consist of different classes than the train set, so a few-shot learner needs to generalize to a greater degree compared to Omniglot. However, the overall content of the dataset is homogeneous, i.e. the character of the test set does not differ significantly from the train set. Finally, we cover META-DATASET, which is currently one of the the most challenging few-shot image classification benchmarks. META-DATASET is a dataset of diverse datasets. The test set not only contains classes that are held out of the train set, but entire datasets are held out, requiring a few-shot learner to adapt and generalize to data that is very different to what it was trained on. Also, unlike Omniglot and *mini*ImageNet, the tasks have random shot and way and present both low and high, shot and way scenarios.

2.7.1 Omniglot

Lake et al. (2011) hypothesize that the sharing of the constituent parts of objects is the key to the one-shot recognition of new objects that may share some of the same component parts of known objects. In order to test their methods, the authors have created a dataset called Omniglot that consists of characters culled from 50 different alphabets. Unlike the popular MNIST (LeCun et al., 2010) handwritten character dataset that has just 10 characters with thousands of examples of each, Omniglot has over 1600 characters with only 20 examples of each. As a result, Omniglot has become a defacto standard dataset for benchmarking few-shot learning methods. The content of Omniglot images is homogeneous - all of the characters are strokes on a constant color background. The defacto standard Omniglot benchmark includes the following classification task configurations: (i) 5-way, 1-shot; (ii) 5-way, 5-shot; (iii) 20-way, 1-shot; (iv) 20-way, 5-shot. See Figure 2.13 for several samples from the dataset.

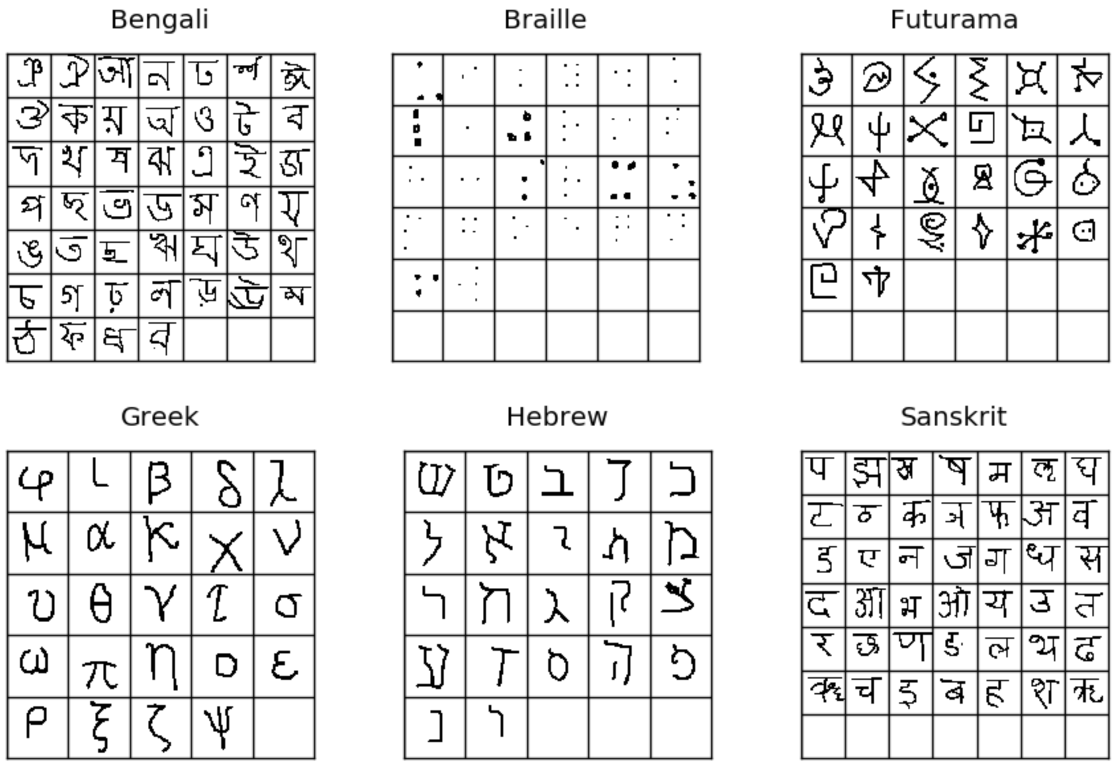


Fig. 2.13 Samples from six Omniglot alphabets. From Bouma (2017).

2.7.2 *mini*ImageNet

*mini*ImageNet is a subset of the larger Imagenet dataset (Russakovsky et al., 2015) created by Vinyals et al. (2016). It consists of 60,000 color images that is sub-divided into 100 classes, each with 600 instances. The images have dimensions of 84×84 pixels. Ravi and Larochelle (2017) standardized the 64 training, 16 validation, and 20 test class splits. Along with Omniglot, *mini*ImageNet has become a defacto standard dataset for benchmarking few-shot image classification methods. *mini*ImageNet is significantly more difficult than Omniglot due to the fact that the images are color and consist of photographic imagery as opposed to hand-written characters. The defacto standard *mini*ImageNet benchmark includes the following classification task configurations: (i) 5-way, 1-shot; (ii) 5-way, 5-shot.

2.7.3 META-DATASET

Despite the ubiquity of Omniglot and *mini*ImageNet protocols for evaluating few-shot classification algorithms, they exhibit several significant disadvantages. The first is that they consider only homogeneous tasks (i.e. the tasks consist of a fixed number of classes and a fixed number of training examples per class), whereas real-life learning tasks are heterogeneous and consist of a variable number of classes and and an unbalanced number of examples per class. The second is that the Omniglot and *mini*ImageNet benchmarks only measure *in* distribution generalization. Ideally, a meta-trained model should be able to generalize to new distributions at meta-test time. The third is that Omniglot and *mini*ImageNet benchmarks are saturated in being able to differentiate the advantages of competing methods. Almost all approaches can achieve high classification accuracy on Omniglot and while *mini*ImageNet benchmark is more challenging than Omniglot, most of the recent methods score similarly on it (assuming the network capacity is held constant), making it difficult to determine which methods are superior.

Recently, Triantafillou et al. (2020) proposed META-DATASET, a few-shot classification benchmark that addresses the issue of homogeneous train and test-time tasks and more closely resembles real-world few-shot multi-task learning by evaluating methods not only on held-out classes, but also on entirely held-out datasets. Many of the approaches that achieved excellent performance on simpler benchmarks struggle with this collection of diverse tasks.

META-DATASET is composed of ten (eight train, two test) image classification datasets. The challenge constructs few-shot learning tasks by drawing from the following distribution. First, one of the datasets is sampled uniformly; second, the “way” and “shot” are sampled randomly according to a fixed procedure; third, the classes and context / target instances are sampled. Where a hierarchical structure exists in the data (ImageNet or Omniglot), task-sampling respects the hierarchy. In the meta-test phase, the identity of the original dataset is not revealed and the tasks must be treated independently (i.e. no information can be transferred between them). Notably, the meta-training set comprises a disjoint and dissimilar set of classes from those used for meta-test. META-DATASET is presently, the "gold standard" for evaluating few-shot classification methods.

Referring to Figure 2.14, the ten datasets that comprise META-DATASET are: (a) ImageNet ILSVRC 2012 (Russakovsky et al., 2015), (b) Omniglot (Lake et al., 2011), (c) Aircraft (Maji et al., 2013), (d) Birds (Wah et al., 2011), (e) Describable Textures (DTD) (Cimpoi et al., 2014), (f) Quick Draw (Ha and Eck, 2017), (g) Fungi (Schroeder and Cui, 2018), (h) VGG Flower (Nilsback and Zisserman, 2008), (i) Traffic Signs (Houben et al., 2013), and (j) MSCOCO (Lin et al., 2014).

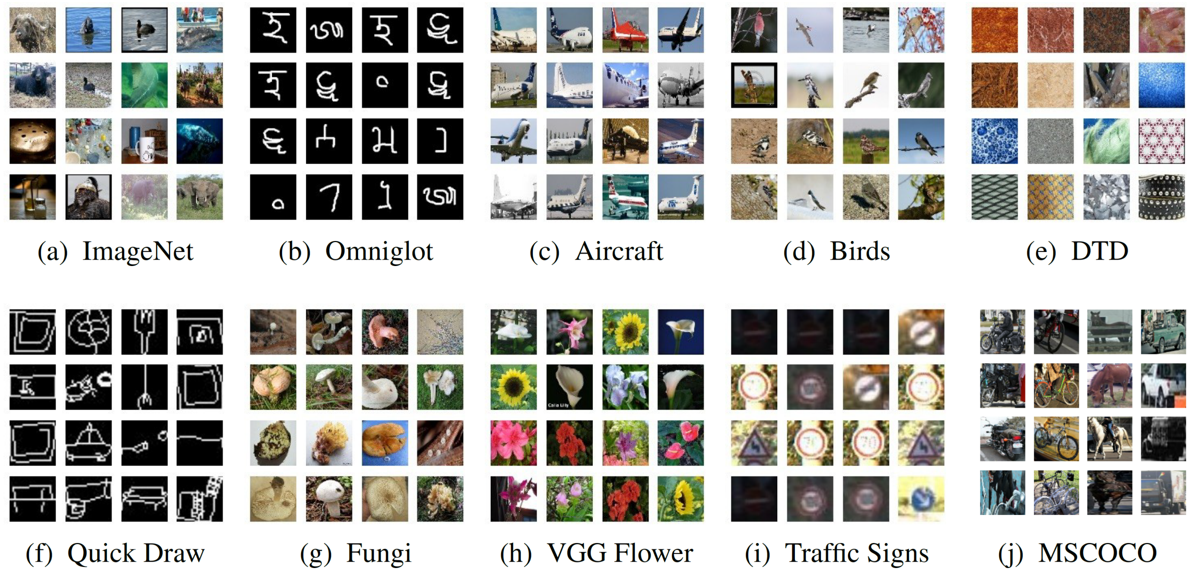


Fig. 2.14 Samples from the ten constituent datasets that form META-DATASET. From Triantafillou et al. (2020).

2.8 Conclusion

In this chapter, we introduced various background concepts relevant to this thesis including meta-learning, multi-task learning, transfer learning, and few-shot learning. Focusing on few-shot image classification, we described a range of methods and categorized them in terms of the mechanism used to adapt model parameters to a given task and which parameters of the model are task-specific.

There has been spectacular progress in the field of few-shot learning research in recent years, spawning a wide array of ever improving techniques. However, there is still much to be done. In particular:

- There exists no unified theoretical view of few-shot learning methods;
- Most few-shot learning methods are designed for and evaluated on datasets that are homogeneous in terms of content, requiring only minimal generalization to diverse tasks;
- Little or no attention has been paid to the computational efficiency of few-shot learning algorithms, especially at test-time, but also at training time.

In response to the above deficiencies, in the following three chapters, we:

- Unify many existing few-shot learning methods, including MAML and Prototypical Networks, under a general framework that we call *Meta-learning approximate Probabilistic Inference for Prediction* or ML-PIP;
- Introduce VERSA and CNAPS, two novel, expressive and flexible amortization based few-shot image classification algorithms that efficiently adapt to diverse new tasks at meta-test time;
- Rethink batch normalization for few-shot learning algorithms by: (i) first investigating the effects of different normalization methods on several few-shot image classification algorithms in terms of both classification accuracy and training efficiency; and then (ii) introduce TASKNORM, a new normalization technique that consistently improves classification accuracy and training efficiency across few-shot learning methods.

Chapter 3

VERSA: Meta-Learning Probabilistic Inference For Prediction

3.1 Introduction

Due to the large quantity of recent work in few-shot learning (Wang et al., 2019; Hospedales et al., 2020), notably in meta-learning based approaches (Ravi and Larochelle, 2017; Vinyals et al., 2016; Edwards and Storkey, 2017; Finn et al., 2017; Lacoste et al., 2018), a unifying view is needed to understand and improve these methods. Existing frameworks (Grant et al., 2018; Finn et al., 2018) are limited to specific families of approaches. In this chapter we develop a framework for meta-learning approximate probabilistic inference for prediction (ML-PIP), providing this view in terms of amortizing posterior predictive distributions. In Section 3.5, we show that ML-PIP re-frames and extends existing point-estimate probabilistic interpretations of meta-learning (Grant et al., 2018; Finn et al., 2018) to cover a broader class of methods, including gradient based meta-learning (Finn et al., 2017; Ravi and Larochelle, 2017), metric based meta-learning (Snell et al., 2017), amortized MAP inference (Qiao et al., 2018) and conditional probability modelling (Garnelo et al., 2018a,b).

The framework incorporates three key elements. First, we leverage shared statistical structure between tasks via hierarchical probabilistic models developed for multi-task and transfer learning (Heskes, 2000; Bakker and Heskes, 2003). Second, we share information between tasks about how to learn and perform inference using meta-learning (Naik and Mammone, 1992; Thrun and Pratt, 2012; Schmidhuber, 1987). Since uncertainty is rife in small datasets, we provide a procedure for meta-learning probabilistic inference. Third, we enable fast learning that can flexibly handle a wide range of tasks and learning settings via amortization (Kingma and Welling, 2014; Rezende et al., 2014).

Building on the framework, we propose a new method – VERSA – which substitutes optimization procedures at test time with forward passes through inference networks. This amortizes the cost of inference, resulting in faster test-time performance, and relieves the need for second derivatives during training. VERSA employs a flexible amortization network that takes few-shot learning datasets, and outputs a distribution over task-specific parameters in a single forward pass. The network can handle arbitrary numbers of shots, and for classification, arbitrary numbers of classes at train and test time (see Section 3.3). In Section 3.6, we evaluate VERSA on (i) standard benchmarks where (at the time) the method set new state-of-the-art results, (ii) settings where test conditions (shot and way) differ from training, and (iii) a challenging one-shot view reconstruction task.

This chapter is based on the conference publication entitled 'Meta-learning Probabilistic Inference for Prediction' (Gordon et al., 2019). As one of two first authors, I contributed jointly to all aspects of the work including the development the the model, devising the set of experiments, preparing the datasets, writing the code, performing the experiments, analyzing the results, and writing the paper.

3.2 Meta-Learning Probabilistic Inference For Prediction

We now present the framework that consists of (i) a multi-task probabilistic model, and (ii) a method for meta-learning probabilistic inference.

3.2.1 Probabilistic Model

Two principles guide the choice of model. First, the use of discriminative models to maximize predictive performance on supervised learning tasks (Ng and Jordan, 2002). Second, the need to leverage shared statistical structure between tasks (i.e. multi-task learning). These criteria are met by the standard multi-task directed graphical model described earlier in Figure 2.3 that employs shared parameters $\boldsymbol{\theta}$, which are common to all tasks, and task specific parameters $\{\boldsymbol{\psi}^{\tau}\}_{\tau=1}^{\mathrm{T}}$. Inputs are denoted $\boldsymbol{x}$ and outputs $\boldsymbol{y}$. Context set $D^{\tau} = \{(\boldsymbol{x}_n^{\tau}, \boldsymbol{y}_n^{\tau})\}_{n=1}^{N_{\tau}}$, and target set $T^{\tau} = \{(\boldsymbol{x}_m^{\tau *}, \boldsymbol{y}_m^{\tau *})\}_{m=1}^{M_{\tau}}$ are explicitly distinguished for each task τ, as this is key for few-shot learning.

Let X^{τ} and Y^{τ} denote all the inputs and outputs (both context and target) for task τ. As shown in Section 2.2.3, the joint probability of the outputs and task specific parameters for T tasks, given the inputs and global parameters is:

$$p\left(\{Y^{\tau}, \boldsymbol{\psi}^{\tau}\}_{\tau=1}^{\mathrm{T}} | \{X^{\tau}\}_{\tau=1}^{\mathrm{T}}, \boldsymbol{\theta}\right) = \prod_{\tau=1}^{\mathrm{T}} p\left(\boldsymbol{\psi}^{\tau}|\boldsymbol{\theta}\right) \prod_{n=1}^{N_{\tau}} p\left(\boldsymbol{y}_n^{\tau}|\boldsymbol{x}_n^{\tau}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}\right) \prod_{m=1}^{M_{\tau}} p\left(\boldsymbol{y}_m^{\tau *}|\boldsymbol{x}_m^{\tau *}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}\right).$$

In this work we are interested in the posterior predictive distribution of our model:

$$\begin{aligned}
p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^{\tau}, \boldsymbol{\theta}) &= p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{y}^{\tau}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \quad \text{(expanding } D^{\tau}\text{)} \\
&= \frac{p(\boldsymbol{y}^{\tau *}, \boldsymbol{x}^{\tau *}|\boldsymbol{y}^{\tau}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta})}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau *}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad \text{(definition of conditional probability)} \\
&= \frac{p(\boldsymbol{y}^{\tau *}, \boldsymbol{x}^{\tau *}|\boldsymbol{y}^{\tau}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta})}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad (\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau} \perp\!\!\!\perp \boldsymbol{x}^{\tau *}) \\
&= \frac{\int p(\boldsymbol{y}^{\tau *}, \boldsymbol{y}^{\tau}, \boldsymbol{\psi}^{\tau}|\boldsymbol{x}^{\tau *}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad \text{(numerator contains the joint probability)} \\
&= \frac{\int p(\boldsymbol{y}^{\tau *}, \boldsymbol{\psi}^{\tau}|\boldsymbol{x}^{\tau *}, \boldsymbol{x}^{\tau}, \boldsymbol{y}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau *}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad \text{(chain rule)} \\
&= \frac{\int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{\psi}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{y}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau *}, \boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad \text{(chain rule)} \\
&= \frac{\int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{\psi}^{\tau}|D^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad (\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau} \perp\!\!\!\perp \boldsymbol{x}^{\tau *}) \\
&= \frac{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{\psi}^{\tau}|D^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}}{p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta})} \quad (p(\boldsymbol{y}^{\tau}|\boldsymbol{x}^{\tau}, \boldsymbol{\theta}) \perp\!\!\!\perp \boldsymbol{\psi}^{\tau}) \\
&= \int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^{\tau}, \boldsymbol{\theta}) p(\boldsymbol{\psi}^{\tau}|D^{\tau}, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^{\tau}
\end{aligned} \tag{3.1}$$

In the next section, the goal is to meta-learn fast and accurate approximations to the posterior predictive distribution for unseen tasks τ.

3.2.2 Probabilistic Inference

This section provides a framework for meta-learning approximate inference that is a simple reframing and extension of existing approaches (Finn et al., 2017; Grant et al., 2018). We will employ point estimates for the shared parameters $\boldsymbol{\theta}$ since data across all tasks will pin down their value. Distributional estimates will be used for the task-specific parameters since only a few shots constrain them.

Once the shared parameters are learned, the probabilistic solution to few-shot learning in the model above comprises two steps. First, form the posterior distribution over the task-specific parameters $p(\boldsymbol{\psi}^{\tau}|\boldsymbol{x}^{\tau *}, D^{\tau}, \boldsymbol{\theta})$. Second, compute the posterior predictive $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^{\tau}, \boldsymbol{\theta})$. These steps will require approximation and the emphasis here is on performing this quickly at test time. We will describe the form of the approximation, the optimization problem used to learn it, and how to implement this efficiently below.

Specification of the approximate posterior predictive distribution. Our framework approximates the posterior predictive distribution by an amortized distribution $q_{\phi}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^{\tau}, \boldsymbol{\theta})$. That is, we learn a feed-forward inference network with parameters ϕ that takes any training dataset D^τ and test input $\boldsymbol{x}^{\tau *}$ as inputs and returns the predictive distribution over the test output $\boldsymbol{y}^{\tau *}$. We construct this by amortizing the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau | \boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ and then form the approximate posterior predictive distribution using:

$$q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) = \int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta}) q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^\tau. \tag{3.2}$$

This step may require additional approximation e.g. Monte Carlo sampling. The amortization will enable fast predictions at test time. The form of these distributions is identical to those used in amortized variational inference (Edwards and Storkey, 2017; Kingma and Welling, 2014). In this work, we use a factorized Gaussian distribution for $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ with means and variances set by the amortization network. However, the training method described next is different.

Meta-learning the approximate posterior predictive distribution. The quality of the approximate posterior predictive for a single task will be measured by the KL-divergence between the true and approximate posterior predictive distribution $\mathrm{KL}\left[p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) \| q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\right]$. The goal of learning will be to minimize the expected value of this KL averaged over tasks,

$$\begin{aligned}
\boldsymbol{\phi}^* &= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \underset{p(D^\tau)}{\mathbb{E}} \left[\mathrm{KL}\left[p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) \| q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\right]\right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \underset{p(D^\tau)}{\mathbb{E}} \left[\underset{p(\boldsymbol{y}^{\tau *}|D^\tau)}{\mathbb{E}} \left[\log(p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) - \log(q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}))\right]\right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \underset{p(\boldsymbol{y}^{\tau *}, D^\tau)}{\mathbb{E}} \left[\log(q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}))\right] \\
&= \underset{\boldsymbol{\phi}}{\operatorname{argmax}} \underset{p(\boldsymbol{y}^{\tau *}, D^\tau)}{\mathbb{E}} \left[\log \int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta}) q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^\tau\right].
\end{aligned} \tag{3.3}$$

Training will therefore return parameters $\boldsymbol{\phi}$ that best approximate the posterior predictive distribution in an average KL sense. So, if the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ is rich enough, *global* optimization will recover the true posterior $p(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ (assuming $p(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ obeys identifiability conditions (Casella and Berger, 2002)).[1] Thus, the amortized procedure meta-learns approximate inference that supports accurate prediction. Section A.1.1 provides a generalized derivation of the framework, grounded in Bayesian decision theory (Jaynes, 2003).

The right hand side of Equation (3.3) indicates how training could proceed: (i) select a task τ at random, (ii) sample some training data D^τ, (iii) form the posterior predictive $q_{\boldsymbol{\phi}}(\cdot|D^\tau)$ and, (iv) compute the log-density $\log q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ at test data $\boldsymbol{y}^{\tau *}$ *not included in* D^τ. Repeating this process many times and averaging the results would provide an unbiased estimate of the

[1]Note that the true *predictive* posterior $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ is recovered regardless of the identifiability of $p(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$.

objective which can then be optimized. This perspective also makes it clear that the procedure is scoring the approximate inference procedure by simulating approximate Bayesian held-out log-likelihood evaluation. Importantly, while an inference network is used to approximate posterior distributions, the training procedure differs significantly from standard variational inference. In particular, rather than minimizing $\mathrm{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\|p(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta}))$, our objective function directly focuses on the posterior predictive distribution and minimizes $\mathrm{KL}\left[p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\|q_{\boldsymbol{\phi}}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\right]$.

End-to-end stochastic training. Armed by the insights above we now layout the full training procedure. We reintroduce inputs and shared parameters θ and the objective becomes:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\phi}) &= -\underset{p(D^\tau, \boldsymbol{y}^{\tau *}, \boldsymbol{x}^{\tau *})}{\mathbb{E}}\left[\log q_{\phi}(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})\right] \\ &= -\underset{p(D, \boldsymbol{y}^{\tau *}, \boldsymbol{x}^{\tau *})}{\mathbb{E}}\left[\log \int p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta}) q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|D^\tau, \boldsymbol{\theta}) \mathrm{d}\boldsymbol{\psi}^\tau\right].\end{aligned} \tag{3.4}$$

We optimize the objective over the shared parameters $\boldsymbol{\theta}$ as this will maximize predictive performance (i.e., Bayesian held out likelihood). An end-to-end stochastic training objective for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ is:

$$\hat{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{1}{M\mathrm{T}} \sum_{M,\mathrm{T}} \log \frac{1}{L} \sum_{l=1}^{L} p\left(\boldsymbol{y}_m^{\tau *}|\boldsymbol{x}_m^{\tau *}, \boldsymbol{\psi}_l^\tau, \boldsymbol{\theta}\right), \quad \text{with } \boldsymbol{\psi}_l^\tau \sim q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|D^\tau, \boldsymbol{\theta}) \tag{3.5}$$

and $\{\boldsymbol{y}_m^{\tau *}, \boldsymbol{x}_m^{\tau *}, D^\tau\} \sim p(\boldsymbol{y}^*, \boldsymbol{x}^*, D)$, where p represents the data distribution (e.g., sampling tasks and splitting them into disjoint training data D^τ and test data $T^\tau = \{(\boldsymbol{x}_m^{\tau *}, \boldsymbol{y}_m^{\tau *})\}_{m=1}^{M_\tau}$). This type of training therefore uses episodic train / test splits at meta-train time. We have also approximated the integral over $\boldsymbol{\psi}$ using L Monte Carlo samples. Interestingly, the learning objective does not require an explicit specification of the prior distribution over parameters, $p(\boldsymbol{\psi}^\tau|\boldsymbol{\theta})$, learning it implicitly through $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|\boldsymbol{x}^{\tau *}, D^\tau, \boldsymbol{\theta})$ instead.

We use the local reparametrization trick (Kingma et al., 2015) for efficient computation. The 'trick' takes advantage of the fact that for a fully factorized Gaussian posterior on the inferred classifier weights $\boldsymbol{w}_\tau$, the posterior activations $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta})$ after multiplying the feature extractor outputs $h_\theta(\boldsymbol{x}^{\tau *})$ by the weights will also be a fully-factorized Gaussian. As a result, instead of sampling the Gaussian classifier weights individually and then multiplying them by the feature extractor outputs to get a sample from $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta})$, the idea is to sample $p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\psi}^\tau, \boldsymbol{\theta})$ directly from the distribution over the resulting activations, leading to a significant computational savings as many fewer variables are sampled. For example, if the dimension of the feature extractor output is $d_{\boldsymbol{\theta}}$ and the task is C-way classification, the local reparameterization trick reduces the number of samples required from $C \times d_{\boldsymbol{\theta}}$ to C (a factor of $d_{\boldsymbol{\theta}}$ improvement). Kingma et al. (2015) also show that using the local reparameterization trick also leads to a lower variance gradient estimator which results in faster and more stable training.

In summary, we have developed an approach for Meta-Learning Probabilistic Inference for Prediction (ML-PIP). A simple investigation of the inference method with synthetic data is provided in Section 3.6.1. In Section 3.5 we will show that this formulation unifies a number of existing approaches, but first we discuss a particular instance of the ML-PIP framework that supports versatile learning.

3.3 Versatile Amortized Inference

A versatile system is one that makes inferences both rapidly *and* flexibly. By rapidly we mean that test-time inference involves only simple computation such as a feed-forward pass through a neural network. By flexibly we mean that the system supports a variety of tasks – including variable numbers of shots or numbers of classes in classification problems – without retraining. Rapid inference comes automatically with the use of a deep neural network to amortize the approximate posterior distribution q. However, it typically comes at the cost of flexibility: amortized inference is usually limited to a single specific task. Below, we discuss design choices that enable us to retain flexibility.

Inference with sets as inputs. The amortization network takes data sets of variable size as inputs whose ordering we should be invariant to. We use permutation-invariant *mean pooling* operations to process these sets as formalized in Zaheer et al. (2017) and described in Section 2.3.3. The deep sets mean pooling operation ensures that the network can process any number of training observations.

VERSA for Few-Shot Image Classification. For few-shot image classification, our parameterization of the probabilistic model is inspired by early work from Heskes (2000); Bakker and Heskes (2003) and recent extensions to deep learning (Bauer et al., 2017; Qiao et al., 2018). A feature extractor neural network $h_{\boldsymbol{\theta}}(x) \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$, shared across all tasks, feeds into a set of task-specific linear classifiers with softmax outputs and weights and biases $\boldsymbol{\psi}^{\tau} = \{W^{\tau}, b^{\tau}\}$ (see Figure 3.1).

A naive amortization requires the approximate posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D, \boldsymbol{\theta})$ to model the distribution over full weight matrices in $\mathbb{R}^{d_{\boldsymbol{\theta}} \times C}$ (and biases). This requires the specification of the number of few-shot classes C ahead of time and limits inference to this chosen number. Moreover, it is difficult to meta-learn systems that directly output large matrices as the output dimensionality is high. Inspired by Qiao et al. (2018), we therefore propose specifying $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D, \boldsymbol{\theta})$ in a *context independent* manner such that each weight vector $\boldsymbol{\psi}_c$ depends only on examples from class c, by amortizing individual weight vectors associated with a single

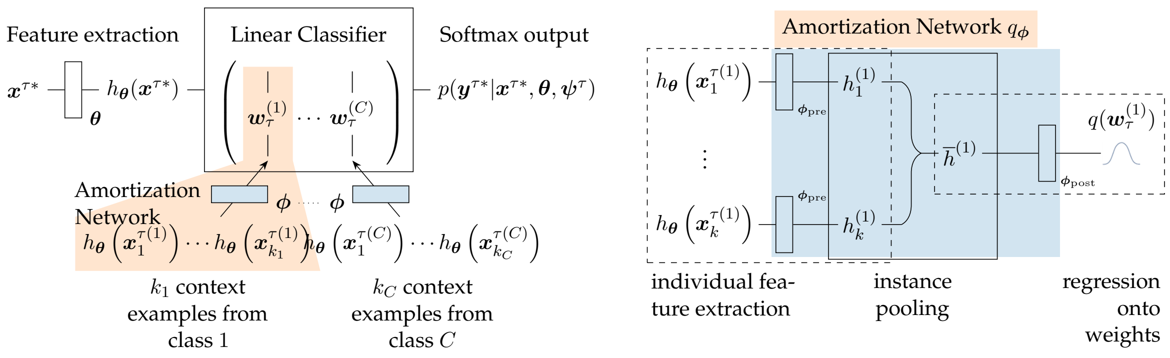

Feature extraction
Linear Classifier
Softmax output
$\boldsymbol{x}^{\tau*}$
$\boldsymbol{\theta}$
$h_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau*})$
$\boldsymbol{w}_{\tau}^{(1)} \cdots \boldsymbol{w}_{\tau}^{(C)}$
$p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, \boldsymbol{\psi}^{\tau})$
Amortization Network
$\boldsymbol{\phi}$
$\boldsymbol{\phi}$
$h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_1^{\tau(1)}\right) \cdots h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k_1}^{\tau(1)}\right) h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_1^{\tau(C)}\right) \cdots h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_{k_C}^{\tau(C)}\right)$
k_1 context examples from class 1
k_C context examples from class C
Amortization Network $q_{\boldsymbol{\phi}}$
$h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_1^{\tau(1)}\right)$
$h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_k^{\tau(1)}\right)$
$\boldsymbol{\phi}_{\text{pre}}$
$\boldsymbol{\phi}_{\text{pre}}$
$h_1^{(1)}$
$h_k^{(1)}$
$\overline{h}^{(1)}$
$\boldsymbol{\phi}_{\text{post}}$
$q(\boldsymbol{w}_{\tau}^{(1)})$
individual feature extraction
instance pooling
regression onto weights

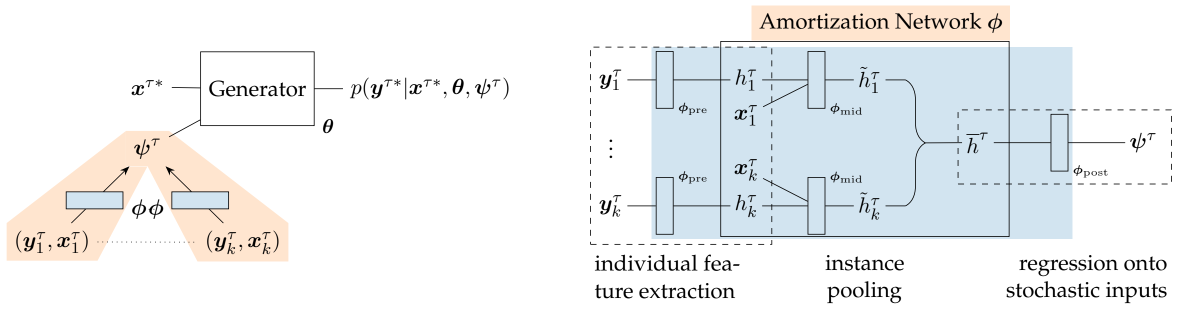
Fig. 3.1 Computational flow of VERSA for few-shot classification with the context-independent approximation. *Left:* A test point $\boldsymbol{x}^{\tau*}$ is mapped to its softmax output through a feature extractor neural network and a linear classifier (fully connected layer). The global parameters $\boldsymbol{\theta}$ of the feature extractor are shared between tasks whereas the weight vectors $\boldsymbol{w}_{\tau}^{(c)}$ of the linear classifier are task specific and inferred through an amortization network with parameters $\boldsymbol{\phi}$. *Right:* Amortization network that maps the extracted features of the k training examples of a particular class to the corresponding weight vector of the linear classifier. The additional superscript in parentheses associated with each context set example indicates the class.

softmax output instead of the entire weight matrix directly. To reduce the number of learned parameters, the amortization network operates directly on the extracted features $h_{\boldsymbol{\theta}}(\boldsymbol{x})$:

$$q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D, \boldsymbol{\theta}) = \prod_{c=1}^{C} q_{\boldsymbol{\phi}}\left(\boldsymbol{\psi}_c|\{h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n^c\right)\}_{n=1}^{k_c}, \theta\right). \tag{3.6}$$

Note that in our implementation, end-to-end training is employed, i.e., we backpropagate to $\boldsymbol{\theta}$ through the inference network. Here k_c is the number of observed examples in class c and $\boldsymbol{\psi}_c = \{\boldsymbol{w}_c, b_c\}$ denotes the weight vector and bias of the linear classifier associated with that class. Thus, we construct the classification matrix $\boldsymbol{\psi}^{\tau}$ by performing C feed-forward passes through the inference network $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D, \boldsymbol{\theta})$ (see Figure 3.1).

The assumption of context independent inference is an approximation. In Section A.1.2, we provide theoretical and empirical justification for its validity. Our theoretical arguments use insights from Density Ratio Estimation (Mohamed, 2018; Sugiyama et al., 2012), and we empirically demonstrate that full approximate posterior distributions are close to their context independent counterparts. Critically, the context independent approximation addresses all the limitations of a naive amortization mentioned above: (i) the inference network needs to amortize far fewer parameters whose number does not scale with number of classes C (a single weight vector instead of the entire matrix); (ii) the amortization network can be meta-trained with different numbers of classes per task, and (iii) the number of classes C can vary at test-time.



$\boldsymbol{x}^{\tau *}$
Generator
$p(\boldsymbol{y}^{\tau *}|\boldsymbol{x}^{\tau *}, \boldsymbol{\theta}, \boldsymbol{\psi}^{\tau})$
$\boldsymbol{\theta}$
$\boldsymbol{\psi}^{\tau}$
$\boldsymbol{\phi}$ $\boldsymbol{\phi}$
$(\boldsymbol{y}_1^{\tau}, \boldsymbol{x}_1^{\tau})$
$(\boldsymbol{y}_k^{\tau}, \boldsymbol{x}_k^{\tau})$
Amortization Network $\boldsymbol{\phi}$
$\boldsymbol{y}_1^{\tau}$
$\boldsymbol{y}_k^{\tau}$
ϕ_{pre}
ϕ_{pre}
h_1^{τ}
$\boldsymbol{x}_1^{\tau}$
$\boldsymbol{x}_k^{\tau}$
h_k^{τ}
ϕ_{mid}
ϕ_{mid}
$\tilde{h}_1^{\tau}$
$\tilde{h}_k^{\tau}$
$\bar{h}^{\tau}$
ϕ_{post}
$\boldsymbol{\psi}^{\tau}$
individual feature extraction
instance pooling
regression onto stochastic inputs

Fig. 3.2 Computational flow of VERSA for few-shot view reconstruction. *Left:* A set of training images and angles $\{(\boldsymbol{y}_n^{\tau}, \boldsymbol{x}_n^{\tau})\}_{n=1}^{k}$ are mapped to a stochastic input $\boldsymbol{\psi}^{\tau}$ through the amortization network $q_{\boldsymbol{\phi}}$. $\boldsymbol{\psi}^{\tau}$ is then concatenated with a test angle $\boldsymbol{x}^{\tau *}$ and mapped onto a new image through the generator $\boldsymbol{\theta}$. *Right:* Amortization network that maps k image/angle examples of a particular object-instance to the corresponding stochastic input. The k view images in the context set $\{\boldsymbol{y}_n^{\tau}\}_{n=1}^{k}$ are fed through a feature extraction network with parameters $\boldsymbol{\phi}_{pre}$ to produce outputs $\{h_n^{\tau}\}_{n=1}^{k}$. These outputs are then are concatenated with the k context view angles $\{\boldsymbol{x}_n^{\tau}\}_{n=1}^{k}$. The concatenated vectors are then passed through a network with parameters $\boldsymbol{\phi}_{mid}$ to produce outputs $\{\tilde{h}_n^{\tau}\}_{n=1}^{k}$. The outputs of this network are then combined using mean pooling. Finally, the pooled output is fed through a network with parameters $\boldsymbol{\phi}_{post}$ to produce the task specific stochastic output $\boldsymbol{\psi}^{\tau}$.

VERSA for Few-Shot Image Reconstruction (Regression). We consider a challenging few-shot learning task with a complex (high dimensional and continuous) output space. We define view reconstruction as the ability to infer how an object looks from any desired angle based on a small set of observed views. We frame this as a multi-output regression task from a set of training images with known orientations to output images with specified orientations. Refer to Figure 2.11 and Section 2.4.3 for a meta-learning perspective on view reconstruction.

Our generative model is similar to the generator of a GAN or the decoder of a VAE: A latent vector $\boldsymbol{\psi}^{\tau} \in \mathbb{R}^{d_{\psi}}$, which acts as an object-instance level input to the generator, is concatenated with a view angle representation and mapped through the generator to produce an image at the specified orientation. In this setting, we treat all parameters $\boldsymbol{\theta}$ of the generator network as global parameters (see Section C.2 for full details of the architecture), whereas the latent inputs $\boldsymbol{\psi}^{\tau}$ are the task-specific parameters. We use a Gaussian likelihood in pixel space for the outputs of the generator. To ensure that the output means are between zero and one, we use a sigmoid activation after the final layer. $\boldsymbol{\phi}$ parameterizes an amortization network that first processes the image representations of an object, concatenates them with their associated view orientations, and processes them further before mean pooling. From the pooled representations, $q_{\boldsymbol{\phi}}(\psi|D, \boldsymbol{\theta})$ produces a distribution over vectors $\boldsymbol{\psi}^{\tau}$. This process is illustrated in Figure 3.2.

3.4 Variational Inference Derivations for the Model

We derive a Variational Inference (VI) based objective for our probabilistic model. VI does not distinguish between context and target data for each task, consequently we define a VI task as the union of the context and target data: $U^\tau = D^\tau \cup T^\tau$. By *amortized* VI we mean that $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta})$ is parameterized by a neural network with a fixed-sized $\boldsymbol{\phi}$ (Kingma and Welling, 2014; Rezende et al., 2014; Kingma et al., 2015; Blundell et al., 2015). Conversely, *non-amortized* VI refers to local parameters $\boldsymbol{\phi}^\tau$ that are optimized independently (at test time) for each new task τ, such that $q(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\psi}^\tau|\mu_{\boldsymbol{\phi}^\tau}, \Sigma_{\boldsymbol{\phi}^\tau})$. However, the derivation of the objective function does not change between these options. For a single task τ, an evidence lower bound (ELBO; (Wainwright and Jordan, 2008)) may be expressed as:

$$\mathcal{L}_\tau = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau,\boldsymbol{\theta})}\left[\sum_{(\boldsymbol{x}^\tau,\boldsymbol{y}^\tau)\in U^\tau} \log p(\boldsymbol{y}^\tau|\boldsymbol{x}^\tau, \boldsymbol{\psi}^\tau, \boldsymbol{\theta})\right] - \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta}) \| p(\boldsymbol{\psi}^\tau|\boldsymbol{\theta})\right]. \tag{3.7}$$

We can then derive a stochastic estimator to optimize Equation (3.7) by sampling $U^\tau \sim p(D)$ (approximated with a training set of tasks) and simple Monte Carlo integration over $\boldsymbol{\psi}^\tau$ such that $\boldsymbol{\psi}_l^\tau \sim q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta})$:

$$\hat{\mathcal{L}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \frac{1}{\mathrm{T}}\sum_{\tau=1}^{\mathrm{T}}\left(\sum_{(\boldsymbol{x}^\tau,\boldsymbol{y}^\tau)\in U^\tau}\left(\frac{1}{L}\sum_{l=1}^{L}\log p(\boldsymbol{y}^\tau|\boldsymbol{x}^\tau, \boldsymbol{\psi}_l^\tau, \boldsymbol{\theta})\right) - \mathrm{KL}\left[q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta}) \| p(\boldsymbol{\psi}^\tau|\boldsymbol{\theta})\right]\right), \tag{3.8}$$

Equation (3.8) differs from our objective function in Equation (3.5) in two important ways: (i) Equation (3.5) does not contain a KL term for $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^\tau|U^\tau, \boldsymbol{\theta})$ (nor any other form of prior distribution over $\boldsymbol{\psi}$), and (ii) Equation (3.7) does not distinguish between training and test data within a task, and therefore does not explicitly encourage the model to generalize in any way.

In Section 3.6, we show that VERSA significantly improves over standard VI in the few-shot classification case and compare to recent VI/meta-learning hybrids.

3.5 ML-PIP Unifies Disparate Related Work

In this section, we continue in the spirit of Grant et al. (2018), and recast a broader class of meta-learning approaches as approximate inference in hierarchical models. We show that ML-PIP unifies a number of important approaches to meta-learning, including *both* gradient and metric based variants, as well as amortized MAP inference and conditional modelling approaches (Garnelo et al., 2018a). We lay out these connections, most of which rely on point estimates for the task-specific parameters corresponding to $q(\boldsymbol{\psi}^\tau|D^\tau, \boldsymbol{\theta}) = \delta\left(\boldsymbol{\psi}^\tau - \boldsymbol{\psi}^*(D^\tau, \boldsymbol{\theta})\right)$. In addition, we compare previous approaches to VERSA.

Gradient-Based Meta-Learning. Let the task-specific parameters ψ^{τ} be all the parameters in a neural network. Consider a point estimate formed by taking a step of gradient ascent of the training loss, initialized at ψ_0 and with learning rate η.

$$\psi^*(D^{\tau}, \boldsymbol{\theta}) = \psi_0 + \eta \frac{\partial}{\partial \psi} \sum_{n=1}^{N_{\tau}} \log p(y_n^{\tau} | x_n^{\tau}, \psi, \boldsymbol{\theta}) \bigg|_{\psi_0}. \tag{3.9}$$

This is an example of semi-amortized inference (Kim et al., 2018), as the only shared inference parameters are the initialization and learning rate, and optimization is required for each task (albeit only for one step). Importantly, Equation (3.9) recovers MAML (Finn et al., 2017), providing a perspective as semi-amortized ML-PIP. This perspective is complementary to that of Grant et al. (2018) who justify the one-step gradient parameter update employed by MAML through MAP inference and the form of the prior $p(\psi|\boldsymbol{\theta})$. Note that the episodic meta-train / meta-test splits do not fall out of this perspective. Instead we view the update choice as one of amortization which is trained using the predictive KL and naturally recovers the test-train splits. More generally, multiple gradient steps could be fed into an RNN to compute ψ^* which recovers Ravi and Larochelle (2017). In comparison to these methods, besides being distributional over ψ, VERSA relieves the need to back-propagate through gradient based updates during training and compute gradients at test time, as well as enables the treatment of both local and global parameters which simplifies inference.

Metric-Based Few-Shot Learning. Let the task-specific parameters be the top layer softmax weights and biases of a neural network $\psi^{\tau} = \{\boldsymbol{w}_c^{\tau}, b_c^{\tau}\}_{c=1}^{C}$. The shared parameters are the lower layer weights. Consider amortized point estimates for these parameters constructed by averaging the top-layer activations for each class,

$$\psi^{\tau *}(D^{\tau}, \boldsymbol{\theta}) = \{\boldsymbol{w}_c^{\tau *}, b_c^{\tau *}\}_{c=1}^{C} = \left\{\mu_c^{\tau}, -\|\mu_c^{\tau}\|^2/2\right\}_{c=1}^{C} \quad \text{where} \quad \mu_c^{\tau} = \frac{1}{k_c} \sum_{n=1}^{k_c} h_{\boldsymbol{\theta}}(\boldsymbol{x}_n^{\tau(c)}) \tag{3.10}$$

These choices lead to the following predictive distribution:

$$p(\boldsymbol{y}^{\tau *} = c | \boldsymbol{x}^{\tau *}, \boldsymbol{\theta}) \propto \exp\left(-d(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau *}), \mu_c^{\tau})\right) = \exp\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{\tau *})^{\mathrm{T}} \mu_c^{\tau} - \frac{1}{2}\|\mu_c^{\tau}\|^2\right), \tag{3.11}$$

which recovers prototypical networks (Snell et al., 2017) using a Euclidean distance function d with the final hidden layer being the embedding space. In comparison, VERSA is distributional and it uses a more flexible amortization function that goes beyond averaging of activations.

Amortized MAP inference. Qiao et al. (2018) proposed a method for predicting weights of classes from activations of a pre-trained network to support i) online learning on a single task to which new few-shot classes are incrementally added, ii) transfer from a high-shot classification task to a separate low-shot classification task. This is an example usage of hyper-networks (Ha et al., 2016) to amortize learning about weights, and can be recovered by the ML-PIP framework by pre-training $\boldsymbol{\theta}$ and performing MAP inference for $\boldsymbol{\psi}$. VERSA goes beyond point estimates and although its amortization network is similar in spirit, it is more general, employing end-to-end training and supporting full multi-task learning by sharing information between many tasks.

Conditional models trained via maximum likelihood. In cases where a point estimate of the task-specific parameters are used the predictive becomes

$$q_{\boldsymbol{\phi}}(\boldsymbol{y}^*|D,\boldsymbol{\theta}) = \int p(\boldsymbol{y}^*|\boldsymbol{\psi},\boldsymbol{\theta})q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D,\boldsymbol{\theta})\mathrm{d}\boldsymbol{\psi} = p(\boldsymbol{y}^*|\boldsymbol{\psi}^*(D,\boldsymbol{\theta}),\boldsymbol{\theta}). \tag{3.12}$$

In such cases the amortization network that computes $\boldsymbol{\psi}^*(D,\boldsymbol{\theta})$ can be equivalently viewed as part of the model specification rather than the inference scheme. From this perspective, the ML-PIP training procedure for $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ is equivalent to training a conditional model $p(\boldsymbol{y}^*|\boldsymbol{\psi}^*_{\boldsymbol{\phi}}(D,\boldsymbol{\theta}),\boldsymbol{\theta})$ via maximum likelihood estimation, establishing a strong connection to neural processes (Garnelo et al., 2018a,b).

3.6 Experiments and Results

We evaluate VERSA on several few-shot learning tasks. We begin with toy experiments to investigate the properties of the amortized posterior inference achieved by VERSA. We then report few-shot classification results using the Omniglot and *mini*ImageNet datasets in Section 3.6.2, and demonstrate VERSA's ability to retain high accuracy as the shot and way are varied at test time. [2]

3.6.1 Posterior Inference with Toy Data

To investigate the approximate inference performed by our training procedure, we run the following experiment. We first generate data from a Gaussian distribution with a mean that varies across tasks:

$$p(\boldsymbol{\theta}) = \delta(\boldsymbol{\theta} - 0); \quad p\left(\boldsymbol{\psi}^\tau|\boldsymbol{\theta}\right) = \mathcal{N}\left(\boldsymbol{\psi}^\tau;\boldsymbol{\theta},\sigma^2_{\boldsymbol{\psi}}\right); \quad p\left(\boldsymbol{y}^\tau_n|\boldsymbol{\psi}^\tau\right) = \mathcal{N}\left(\boldsymbol{y}^\tau_n;\boldsymbol{\psi}^\tau,\sigma^2_{\boldsymbol{y}}\right). \tag{3.13}$$

We generate $\mathrm{T} = 250$ tasks in two separate experiments, having $N \in \{5, 10\}$ train observations and $M = 15$ test observations. We introduce the inference network $q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D^\tau) = \mathcal{N}(\boldsymbol{\psi};\mu^\tau_q,\sigma^{\tau 2}_q)$,

[2]Source code for the experiments is available at https://github.com/Gordonjo/versa.

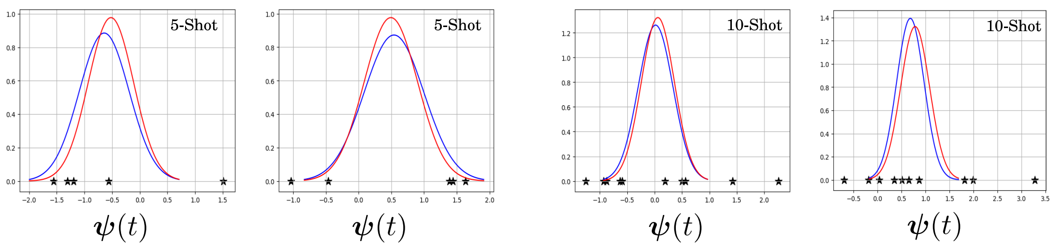


Fig. 3.3 True posteriors $p(\psi|D)$ (—) and approximate posteriors $q_{\phi}(\psi|D)$ (—) for unseen test sets ($\star$) in the experiment. In both cases (five and ten shot), the approximate posterior closely resembles the true posterior given the observed data.

amortizing inference as:

$$\mu_q^\tau = \boldsymbol{w}_\mu \sum_{n=1}^{N} \boldsymbol{y}_n^\tau + \boldsymbol{b}_\mu, \quad \sigma_q^{\tau 2} = \exp\left(\boldsymbol{w}_\sigma \sum_{n=1}^{N} \boldsymbol{y}_n^\tau + \boldsymbol{b}_\sigma \right). \tag{3.14}$$

The learnable parameters $\boldsymbol{\phi} = \{\boldsymbol{w}_\mu, \boldsymbol{b}_\mu, \boldsymbol{w}_\sigma, \boldsymbol{b}_\sigma\}$ are trained with the objective function in Equation (3.5). The model is trained to convergence with Adam (Kingma and Ba, 2015) using mini-batches of tasks from the generated dataset. Then, a separate set of tasks is generated from the same generative process, and the posterior $q_{\boldsymbol{\phi}}(\psi|D)$ is inferred with the learned amortization parameters. The true posterior over ψ is Gaussian with a mean that depends on the task, and may be computed analytically. Figure 3.3 shows the approximate posterior distributions inferred for unseen test sets by the trained amortization networks. The evaluation shows that the inference procedure is able to recover accurate posterior distributions over ψ, despite minimizing a predictive KL divergence in data space.

3.6.2 Few-shot Classification

We evaluate VERSA on standard few-shot classification tasks in comparison to previous work. Specifically, we consider the Omniglot (Lake et al., 2011) and *mini*ImageNet (Ravi and Larochelle, 2017) datasets which are C-way classification tasks with k_c examples per class. VERSA follows the implementation in Sections 3.2 and 3.3, and the approximate inference scheme in Equation (3.6). We follow the experimental protocol established by Vinyals et al. (2016) for Omniglot and Ravi and Larochelle (2017) for *mini*Imagenet, using equivalent architectures for $h_{\boldsymbol{\theta}}$. Training is carried out in an episodic manner: for each task, k_c examples are used as training inputs to infer $q_{\boldsymbol{\phi}}(\psi^{(c)}|D, \boldsymbol{\theta})$ for each class, and an additional set of examples is used to evaluate the objective function. Full details of data preparation and network architectures are provided in Section B.1 and Section C.1, respectively.

Table 3.1 details few-shot classification performance for VERSA as well as competitive approaches.

The tables include results for only those approaches with comparable training procedures and convolutional feature extraction architectures. Approaches that employ pre-training and/or residual networks (Bauer et al., 2017; Qiao et al., 2018; Rusu et al., 2018; Gidaris and Komodakis, 2018; Oreshkin et al., 2018; Satorras and Estrach, 2018; Lacoste et al., 2018) have been excluded so that the quality of the learning algorithm can be assessed separately from the power of the underlying discriminative model. For Omniglot, the training, validation, and test splits have not been specified for previous methods, affecting the comparison.

VERSA achieves state-of-the-art results at the time (67.37% - up 1.38% over the previous best) on 5-way - 5-shot classification on the *mini*ImageNet benchmark and (97.66% - up 0.02%) on the 20-way - 1 shot Omniglot benchmark for systems using a similar feature extractor architecture and an end-to-end training procedure. VERSA is within error bars of state-of-the-art (at the time) on three other benchmarks including 5-way - 1-shot *mini*ImageNet, 5-way - 5-shot Omniglot, and 5-way - 1-shot Omniglot. Results on the Omniglot 20 way - 5-shot benchmark are very competitive with, but lower than other approaches. While most of the methods evaluated in Table 3.1 adapt all of the learned parameters for new tasks, VERSA is able to achieve state-of-the-art performance despite adapting only the weights of the top-level classifier.

Comparison to standard and amortized VI. To investigate the performance of our inference procedure, we compare it in terms of log-likelihood (Table 3.2) and accuracy (Table 3.1) to training the same model using both amortized and non-amortized VI (i.e., Equation (3.8)). Derivations and details are provided in Section 3.4. VERSA improves substantially over amortized VI even though the same amortization network is used for both. This is due to VI's tendency to under-fit, especially for small numbers of data points (Trippe and Turner, 2018; Turner and Sahani, 2011) which is compounded when using inference networks (Cremer et al., 2018). Using non-amortized VI improves performance substantially, but does not reach the level of VERSA and forming the posterior is significantly slower as it requires many forward / backward passes through the network. This is similar in spirit to MAML (Finn et al., 2017), though MAML dramatically reduces the number of required iterations by finding good global initializations e.g., five gradient steps for *mini*ImageNet. This is in contrast to the single forward pass required by VERSA.

Versatility. VERSA allows us to vary the number of classes C and shots k_c between training and testing (Equation (3.6)). Figure 3.4a shows that a model trained for a particular C-way retains very high accuracy as C is varied. For example, when VERSA is trained for the 20-Way, 5-Shot condition, at test-time it can handle $C = 100$ way conditions and retain an accuracy of approximately 94%. Figure 3.4b shows similar robustness as the number of shots k_c is varied. VERSA therefore demonstrates considerable flexibility and robustness to the test-time conditions, but at the same time it is efficient as it only requires forward passes through the

	Omniglot				*miniImageNet*	
	5-way accuracy (%)		**20-way accuracy (%)**		**5-way accuracy (%)**	
Method	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Siamese Nets (Koch et al., 2015)	97.3	98.4	88.1	97.0		
Matching Nets (Vinyals et al., 2016)	98.1	98.9	93.8	98.5	46.6	60.0
Neural Statistician (Edwards and Storkey, 2017)	98.1	99.5	93.2	98.1		
Memory Mod (Kaiser et al., 2017)	98.4	99.6	95.0	98.6		
Meta LSTM (Ravi and Larochelle, 2017)					43.44 ± 0.77	60.60 ± 0.71
MAML (Finn et al., 2017)	98.7 ± 0.4	$\mathbf{99.9 \pm 0.1}$	95.8 ± 0.3	98.9 ± 0.2	48.7 ± 1.84	63.11 ± 0.92
Prototypical Nets[3] (Snell et al., 2017)	97.4	99.3	95.4	98.7	46.61 ± 0.78	65.77 ± 0.70
mAP-SSVM (Triantafillou et al., 2017)	98.6	99.6	95.2	98.6	50.32 ± 0.80	63.94 ± 0.72
mAP-DLM (Triantafillou et al., 2017)	98.8	99.6	95.4	98.6	50.28 ± 0.80	63.70 ± 0.70
LLAMA (Grant et al., 2018)					49.40 ± 1.83	
PLATIPUS (Finn et al., 2018)					50.13 ± 1.86	
Meta-SGD (Li et al., 2017)	$\mathbf{99.53 \pm 0.26}$	$\mathbf{99.93 \pm 0.09}$	95.93 ± 0.38	98.97 ± 0.19	50.47 ± 1.87	64.03 ± 0.94
SNAIL (Mishra et al., 2018)	99.07 ± 0.16	99.78 ± 0.09	$\mathbf{97.64 \pm 0.30}$	$\mathbf{99.36 \pm 0.18}$	45.1	55.2
Relation Net (Sung et al., 2018)	$\mathbf{99.6 \pm 0.2}$	$\mathbf{99.8 \pm 0.1}$	$\mathbf{97.6 \pm 0.2}$	$\mathbf{99.1 \pm 0.1}$	50.44 ± 0.82	65.32 ± 0.70
Reptile (Nichol et al., 2018)	97.68 ± 0.04	99.48 ± 0.06	89.43 ± 0.14	97.12 ± 0.32	49.97 ± 0.32	$\mathbf{65.99 \pm 0.58}$
BMAML (Yoon et al., 2018)					$\mathbf{53.8 \pm 1.46}$	
Amortized VI	97.77 ± 0.55	98.71 ± 0.22	90.56 ± 0.54	96.12 ± 0.23	44.13 ± 1.78	55.68 ± 0.91
Non-Amortized VI	98.77 ± 0.18	99.74 ± 0.06	95.28 ± 0.19	98.84 ± 0.09		
VERSA (Ours)	$\mathbf{99.70 \pm 0.20}$	$\mathbf{99.75 \pm 0.13}$	$\mathbf{97.66 \pm 0.29}$	98.77 ± 0.18	$\mathbf{53.40 \pm 1.82}$	$\mathbf{67.37 \pm 0.86}$

Table 3.1 Accuracy results for different few-shot settings on Omniglot and *mini*ImageNet. The ± sign indicates the 95% confidence interval over tasks using a Student's t-distribution approximation. Bold text indicates the highest scores that overlap in their confidence intervals. Blank entries indicate that no result was computed or published for that configuration. Results for Non-Amortized VI were not computed on *mini*ImageNet due to the lengthy computation time.

	Omniglot				*miniImageNet*	
	5-way NLL		**20-way NLL**		**5-way NLL**	
Method	1-shot	5-shot	1-shot	5-shot	1-shot	5-shot
Amortized VI	0.179 ± 0.009	0.137 ± 0.004	0.456 ± 0.010	0.253 ± 0.004	1.328 ± 0.024	1.165 ± 0.010
Non-Amortized VI	0.144 ± 0.005	0.025 ± 0.001	0.393 ± 0.005	0.078 ± 0.002		
VERSA	0.010 ± 0.005	0.007 ± 0.003	0.079 ± 0.009	0.031 ± 0.004	1.183 ± 0.023	0.859 ± 0.015

Table 3.2 Negative Log-likelihood (NLL) results for different few-shot settings on Omniglot and *mini*ImageNet. The ± sign indicates the 95% confidence interval over tasks using a Student's t-distribution approximation. Note that results for Non-Amortized VI were not computed on *mini*ImageNet due to the lengthy computation time.

network. The time taken to evaluate 1000 test tasks with a 5-way, 5-shot *mini*ImageNet trained model using MAML (https://github.com/cbfinn/maml) is 302.9 seconds whereas VERSA took 53.5 seconds on a NVIDIA Tesla P100-PCIE-16GB GPU. This is more than $5\times$ speed advantage in favor of VERSA while bettering MAML in accuracy by 4.26%.

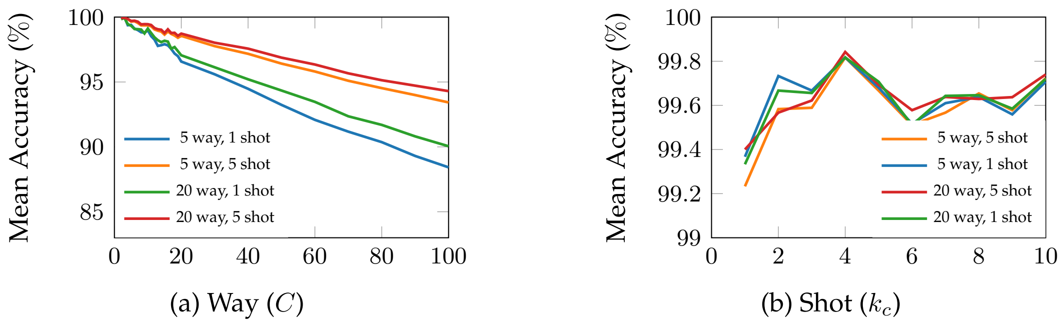


Fig. 3.4 Test accuracy on Omniglot when varying (a) way (fixing shot to be that used for training) and (b) shot. In Figure 3.4b, all models are evaluated on 5-way classification. Colors indicate models trained with different way-shot episodic combinations.

3.6.3 ShapeNet View Reconstruction

ShapeNetCore v2 (Chang et al., 2015) is an annotated database of 3D objects covering 55 common object categories with ~51,300 unique objects. For our experiments, we use 12 of the largest object categories. Refer to Table 3.3 for a complete list. We concatenate all instances from all 12 of the object categories together to obtain a dataset of 37,108 objects. This concatenated dataset is then randomly shuffled and we use 70% of the objects (25,975 in total) for training, 10% for validation (3,710 in total) , and 20% (7423 in total) for testing. For each object, we generate $V = 36$, 128×128 pixel image views spaced evenly every 10 degrees in azimuth around the object. We then convert the rendered images to gray-scale and reduce their size to be 32×32 pixels. Again, we train our model in an episodic manner. Each training iteration consists a batch of one or more tasks. For each task an object is selected at random from the training set. We train on a single view selected at random from the $V = 36$ views associated with each object and use the remaining 35 views to evaluate the objective function. We then generate 36 views of the object with a modified version of our amortization network which is shown diagrammatically in Figure 3.2. To evaluate the system, we generate views and compute quantitative metrics over the entire test set. Tables C.5 to C.7 describe the network architectures for the encoder, amortization, and generator networks, respectively. To train, we use the Adam (Kingma and Ba, 2015) optimizer with a constant learning rate of 0.0001 with 24 tasks per batch for 500,000 training iterations. In addition, we set $d_\phi = 256$, $d_\psi = 256$ and number of ψ samples to 1.

Object Category	**sysnet ID**	**Instances**
airplane	02691156	4045
bench	02828884	1813
cabinet	02933112	1571
car	02958343	3533
phone	02992529	831
chair	03001627	6778
display	03211117	1093
lamp	03636649	2318
speaker	03691459	1597
sofa	04256520	3173
table	04379243	8436
boat	04530566	1939

Table 3.3 List of ShapeNet categories used in the VERSA view reconstruction experiments.

We evaluate VERSA view reconstruction by comparing it to a conditional variational autoencoder (C-VAE) with view angles as labels (Kingma et al., 2014; Narayanaswamy et al., 2017) and identical architectures. We train VERSA in an episodic manner and the C-VAE in batch-mode on all 12 object classes at once. We train on a single view selected at random and use the remaining views to evaluate the objective function. Figure 3.5 shows views of unseen objects from the test set generated from a single shot with VERSA as well as a C-VAE and compares both to ground truth views. Both VERSA and the C-VAE capture the correct orientation of the object in the generated images. However, VERSA produces images that contain much more detail and are visually sharper than the C-VAE images. Although important information is missing due to occlusion in the single shot, VERSA is often able to accurately impute this information presumably due to learning the statistics of these objects. Table 3.4 provides quantitative comparison results between VERSA with varying shot and the C-VAE. The quantitative metrics all show the superiority of VERSA over a C-VAE. As the number of shots increase to 5, the measurements show a corresponding improvement.

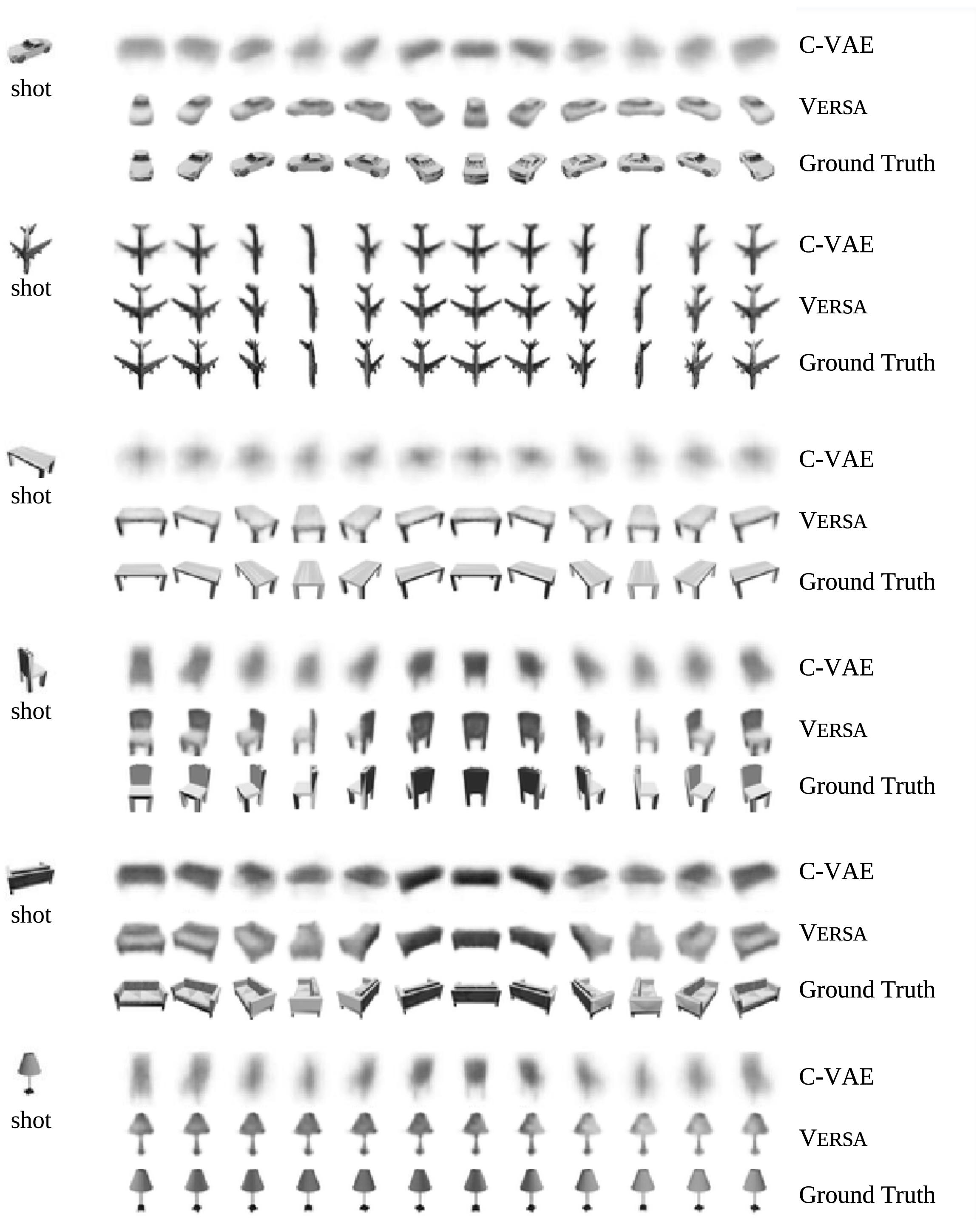


Fig. 3.5 Results for ShapeNet view reconstruction for unseen objects from the test set (shown left). The model was trained to reconstruct views from a single orientation. *Top row:* images/views generated by a C-VAE model; *middle row* images/views generated by VERSA; *bottom row:* ground truth images. Views are spaced evenly every 30 degrees in azimuth.

Model	MSE	SSIM
C-VAE 1-shot	0.0269	0.5705
VERSA 1-shot	0.0108	0.7893
VERSA 5-shot	0.0069	0.8483

Table 3.4 View reconstruction test results. Mean squared error (MSE – lower is better) and the structural similarity index (SSIM - higher is better) (Wang et al., 2004) are measured between the generated and ground truth images. Error bars not shown as they are insignificant.

3.7 Summary

In this chapter, we have introduced ML-PIP, a probabilistic framework for meta-learning. ML-PIP unifies a broad class of recently proposed meta-learning methods, and suggests alternative approaches. Building on ML-PIP, we developed VERSA, a few-shot learning algorithm that avoids the use of gradient based optimization at test time by amortizing posterior inference of task-specific parameters. We evaluated VERSA on several few-shot learning tasks and at the time, demonstrated state-of-the-art performance.

3.8 Epilogue

In this section, we capture follow-on work related to ML-PIP and VERSA.

Few-shot classification accuracy continues to climb Since the writing of this paper, there have been numerous papers describing new approaches to few-shot image classification with continuously improving results on the *mini*ImageNet benchmark. Most of the improvements can be attributed to more capable feature extractors that produce superior embeddings. In fact, two very recent papers (Tian et al., 2020; Chen et al., 2020) argue that using a good pre-trained embedding combined with fine-tuning a linear classifier or a metric-based classifier can yield better results than a sophisticated meta-learning algorithm. This is a very interesting result and certainly demands close scrutiny. However, a potential confounder in these experiments is that they used settings where these conclusions were drawn using tasks and evaluation conditions that were homogeneous (i.e. they used fixed way and shot), by only testing on in-distribution tasks, and in the case of fine tuning, by not considering the cost of doing extensive optimization at test time. In Chapter 4, we consider much more complex tasks and show that fine-tuning and metric-learning approaches (such as prototypical networks) can be significantly outperformed.

VERSA view reconstruction compared to CNPs, NPs, and GQN Our work on view reconstruction was published prior to the Neural Processes (NP), Conditional Neural Process (CNP), and Generative Query Network (GQN) work summarized in Section 2.4, however the development was independent and concurrent. In retrospect, our VERSA view reconstruction model is almost identical to a CNP. Compare Figure 3.2 and Figure 2.9. The ϕ network that pools the input context set with a deep sets network in the two models is identical. The representation $\boldsymbol{r}$ in the CNP maps directly to ψ^{τ} in VERSA with the exception that VERSA retains uncertainty in the parameters, and the CNP decoder with network ρ is the same as the VERSA view reconstruction decoder with parameters $\boldsymbol{\theta}$. The two systems are precisely the same when there is no uncertainty in the VERSA q_{ϕ} distribution over parameters. When there is uncertainty, VERSA is closer to a NP model, although the training objective is different because VERSA does not use variational inference, while a NP does.

GQN view reconstruction was based on a NP (as opposed to a CNP) model and took advantage of the additional latent variable $\boldsymbol{z}$ to offer superior uncertainty estimates. The GQN approach also used a much more elaborate decoder based on a recurrent neural network. The GQN experimental work went considerably further than ours. GQN demonstrated view reconstruction on simple 3D scenes containing several objects whereas the VERSA view reconstruction only dealt with a single 3D object.

Variance collapse in ML-PIP In a recent paper that builds on the ML-PIP and VERSA concepts, Iakovleva et al. (2020) show that training with the approximation in Equation (3.5) tends to severely underestimate the variance in the inferred task specific weights ψ^{τ}, effectively reducing the VERSA model to be deterministic. We also noticed this in our image classification experiments. This can occur because the ML-PIP training objective Equation (3.5) is exclusively focused on the predictive distribution as opposed to the distribution over the learned classifier weights. In the simple experiment in Section 3.6.1 that uses Gaussian data, we see that in Figure 3.3 the weight distribution is properly inferred and does not collapse since the true weight distribution and true predictive distribution are also Gaussian (and identifiable). However, when doing larger scale image classification experiments, where the distribution is identifiable to a lesser degree, uncertainty is pushed into label noise instead of parameter uncertainty, resulting in a collapse of the weight variance.

To avoid this problem, Iakovleva et al. (2020) take a different approach in their system called SAMOVAR and use amortized variational inference to learn the distribution over the task specific weights using a loss identical to that used in NPs. When compared directly to VERSA, the classification accuracy of SAMOVAR is almost identical, though the variance of the distribution over the weights did not collapse in SAMOVAR as it does in VERSA. A downside of SAMOVAR is that it requires orders of magnitude greater number of Monte Carlo samples to achieve optimal classification accuracy, making adaptation to unseen tasks considerably slower than VERSA.

In another recent paper, Foong et al. (2020) present a modified version of the ML-PIP loss that should avoid the variance collapse:

$$\hat{\mathcal{L}}\left(\boldsymbol{\theta}, \boldsymbol{\phi}\right) = \frac{1}{\mathrm{T}} \sum_{\mathrm{T}} \log \left[\frac{1}{L} \sum_{l=1}^{L} \exp \left(\sum_{M} p\left(\boldsymbol{y}_m^{\tau *} | \boldsymbol{x}_m^{\tau *}, \boldsymbol{\psi}_l^{\tau}, \boldsymbol{\theta} \right) \right) \right], \quad \text{with } \boldsymbol{\psi}_l^{\tau} \sim q_{\boldsymbol{\phi}}(\boldsymbol{\psi}^{\tau} | D^{\tau}, \boldsymbol{\theta}) \quad (3.15)$$

This improved version of the ML-PIP loss computes the joint likelihood over the target set compared to the simple average as in Equation (3.5). This small modification in the new formulation captures correlations in the target set and structure in the predictive distribution, potentially avoiding the weight variance collapse, though this has yet to be validated in image classification experiments.

Chapter 4

CNAPS: Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes

4.1 Introduction

In this chapter, we consider the development of general purpose image classification systems that can handle tasks from a broad range of data distributions, in both the low and high data regimes, without the need for costly retraining when new tasks are encountered. We argue that such systems require mechanisms that adapt to each task, and that these mechanisms should themselves be learned from a diversity of datasets and tasks at training time. This general approach relates to methods for meta-learning (Schmidhuber, 1987; Thrun and Pratt, 2012) and few-shot learning (Lake et al., 2015). However, existing work in this area typically considers homogeneous task distributions at train and test-time that therefore require only minimal adaptation. To handle the more challenging case of different task distributions we design a fully adaptive system, requiring specific design choices in the model and training procedure.

Current approaches to meta-learning and few-shot learning for classification are characterized by two fundamental trade-offs (refer back to Figure 2.4). (i) The number of parameters that are adapted to each task. One approach adapts only the top, or head, of the classifier leaving the feature extractor fixed (Snell et al., 2017; Gordon et al., 2019). While useful in simple settings, this approach is prone to under-fitting when the task distribution is heterogeneous (Triantafillou et al., 2020). Alternatively, we can adapt all parameters in the feature extractor (Finn et al., 2017; Nichol et al., 2018) thereby increasing fitting capacity, but incurring a computation cost and opening the door to over-fitting in the low-shot regime. What is needed is a middle ground which strikes a balance between model capacity and reliability of the adaptation. (ii) The adaptation mechanism. Many approaches use gradient-based adaptation (Finn et al., 2017; Yosinski et al., 2014). While this approach can incorporate training data in a very flexible way, it is computationally inefficient at test-time, may require expertise to tune the optimization procedure, and is again prone to over-fitting. Conversely, function approximators can be used to directly map training data to the desired parameters (we refer to this as *amortization*) (Gordon et al., 2019; Qiao et al., 2018). This yields fixed-cost adaptation mechanisms, and enables greater sharing across training tasks. However, it may under-fit if the function approximation is not sufficiently flexible. On the other hand, high-capacity function approximators require a large number of training tasks to be learned.

We introduce a modelling class that is well-positioned with respect to these two trade-offs for the multi-task classification setting called Conditional Neural Adaptive Processes (CNAPS).[1] CNAPS directly model the desired predictive distribution (Geisser, 1983, 2017), thereby introducing a *conditional neural processes* (CNPs) (Garnelo et al., 2018a) approach to the multi-task classification setting. CNAPS handles varying way classification tasks and introduces a parametrization and training procedure enabling the model to *learn to adapt* the feature representation for classification of diverse tasks at test time. CNAPS utilize i) a classification model with shared global parameters and a small number of task-specific parameters. We demonstrate that by identifying a small set of key parameters, the model can balance the trade-off between flexibility and robustness. ii) A rich adaptation neural network with a novel auto-regressive parameterization that avoids under-fitting while proving easy to train in practice with existing datasets (Triantafillou et al., 2020). In Section 4.5 we evaluate CNAPS. Recently, Triantafillou et al. (2020) proposed META-DATASET, a few-shot classification benchmark that addresses the issue of homogeneous train and test-time tasks and more closely resembles real-world few-shot multi-task learning. Many of the approaches that achieved excellent performance on simple benchmarks struggle with this collection of diverse tasks. In contrast, we show that CNAPS achieved (at the time) state-of-the-art performance on the META-DATASET benchmark, often by comfortable margins and at a fraction of the time required by competing methods. Finally, we showcase the versatility of the model class by demonstrating that CNAPS can be applied "out of the box" to continual learning and active learning.

This chapter is based on the conference publication entitled 'Fast and Flexible Multi-Task Classification Using Conditional Neural Adaptive Processes' (Requeima et al., 2019b). I was one of the joint-first authors and we all contributed equally to all aspects of the work including the development the the model, devising the set of experiments, preparing the datasets, writing the code, performing the experiments, analyzing the results, and writing the paper.

[1]Source code available at https://github.com/cambridge-mlg/cnaps.

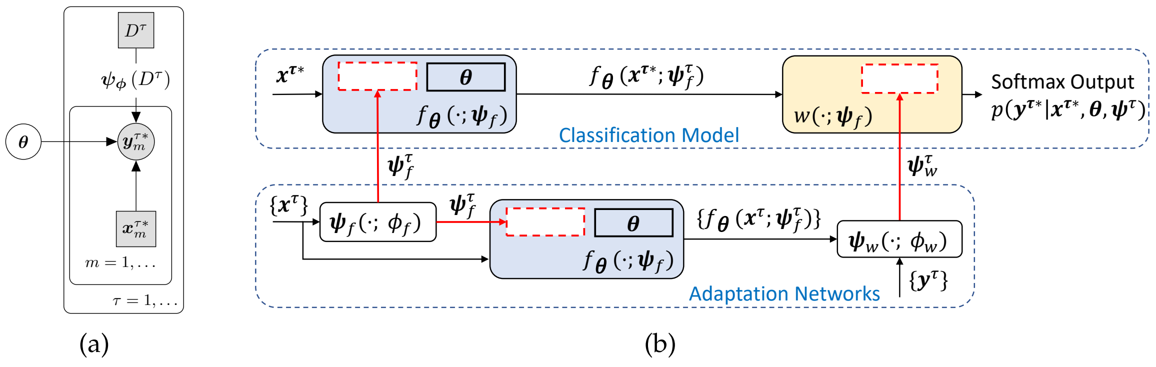


Fig. 4.1 (a) Probabilistic graphical model detailing the CNP (Garnelo et al., 2018a) framework. Shaded nodes indicate variables that are observed. The data for a task τ consists of a *context set* $D^\tau = \{(\boldsymbol{x}_n^\tau, \boldsymbol{y}_n^\tau)\}_{n=1}^{N_\tau}$ with N_τ elements with the inputs $\boldsymbol{x}_n^\tau$ and labels $\boldsymbol{y}_n^\tau$ observed, and a *target set* $\{(\boldsymbol{x}_m^{\tau*}, \boldsymbol{y}_m^{\tau*})\}_{m=1}^{M_\tau}$ with M_τ elements for which we wish to make predictions. Here the inputs $\boldsymbol{x}^{\tau*}$ are observed and the labels $\boldsymbol{y}^{\tau*}$ are only observed during training. $\boldsymbol{\theta}$ are global classifier parameters shared across tasks. $\boldsymbol{\psi}^\tau$ are local task-specific parameters, produced by a function $\boldsymbol{\psi}_{\boldsymbol{\phi}}(\cdot)$ that acts on the context set D^τ. $\boldsymbol{\psi}_{\boldsymbol{\phi}}(\cdot)$ has another set of global adaptation network parameters $\boldsymbol{\phi}$. $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are the learnable parameters in the model. (b) Computational diagram depicting the CNAPS model class. Red boxes imply parameters in the model architecture supplied by adaptation networks $\boldsymbol{\psi}_f$ and $\boldsymbol{\psi}_w$. Blue shaded boxes depict the feature extractor and the gold box depicts the linear classifier. In the lower Adaptation Networks box, the context set inputs $\{\boldsymbol{x}^\tau\}$ are fed into the feature extractor adaptation network $\boldsymbol{\psi}_f$ to generate the parameters to adapt the feature extractor to the current task. The context set inputs are then fed into the adapted feature extractor whose output is passed to the linear classifier adaptation network $\boldsymbol{\psi}_w$ which generates the weights for the linear classifier for the current task. In the upper Classification Model box, the target inputs $\{\boldsymbol{x}^{\tau*}\}$ are classified by the adapted feature extractor and linear classifier to produce the labels $\boldsymbol{y}^{\tau*}$ at the Softmax Output.

4.2 Model Design

We consider a setup where a large number of training tasks are available, each composed of a set of inputs $\boldsymbol{x}$ and labels $\boldsymbol{y}$. The data for task τ includes a *context set* $D^\tau = \{(\boldsymbol{x}_n^\tau, \boldsymbol{y}_n^\tau)\}_{n=1}^{N_\tau}$, with inputs and outputs observed, and a *target set* $\{(\boldsymbol{x}_m^{\tau*}, \boldsymbol{y}_m^{\tau*})\}_{m=1}^{M_\tau}$ for which we wish to make predictions ($\boldsymbol{y}^{\tau*}$ are only observed during training). CNPs (Garnelo et al., 2018a) construct predictive distributions given $\boldsymbol{x}^{\tau*}$ as:

$$p\left(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, D^\tau\right) = p\left(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau = \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right)\right). \tag{4.1}$$

Here $\boldsymbol{\theta}$ are global classifier parameters shared across tasks. $\boldsymbol{\psi}^\tau$ are local task-specific parameters, produced by a function $\boldsymbol{\psi}_{\boldsymbol{\phi}}(\cdot)$ that acts on D^τ. $\boldsymbol{\psi}_{\boldsymbol{\phi}}(\cdot)$ has another set of global parameters $\boldsymbol{\phi}$ called *adaptation network parameters*. $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are the learnable parameters in the model (see Figure 4.1a).

CNAPS is a model class that specializes the CNP framework for the multi-task classification setting. The model-class is characterized by a number of design choices, made specifically for the multi-task image classification setting. CNAPS employ global parameters $\boldsymbol{\theta}$ that are trained offline to capture high-level features, facilitating transfer and multi-task learning. Whereas CNPs define $\boldsymbol{\psi}^{\tau}$ to be a fixed dimensional vector used as an input to the model, CNAPS instead let $\boldsymbol{\psi}^{\tau}$ be specific parameters of the model itself. This increases the flexibility of the classifier, enabling it to model a broader range of input / output distributions. We discuss our choices (and associated trade-offs) for these parameters below. Finally, CNAPS employ a novel auto-regressive parameterization of $\boldsymbol{\psi}_{\boldsymbol{\phi}}(\cdot)$ that significantly improves performance. An overview of CNAPS and its key components is illustrated in Figure 4.1b.

4.2.1 Specification of the classifier: global $\boldsymbol{\theta}$ and task-specific parameters $\boldsymbol{\psi}^{\tau}$

We begin by specifying the classifier's global parameters $\boldsymbol{\theta}$ followed by how these are adapted by the local parameters $\boldsymbol{\psi}^{\tau}$.

Global Classifier Parameters. The global classifier parameters will parameterize a feature extractor $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ whose output is fed into a linear classifier, described below. A natural choice for $f_{\boldsymbol{\theta}}(\cdot)$ in the image setting is a convolutional neural network, e.g., a ResNet (He et al., 2016). In what follows, we assume that the global parameters $\boldsymbol{\theta}$ are fixed and known. In Section 4.3 we discuss the training of $\boldsymbol{\theta}$.

Task-Specific Classifier Parameters: Linear Classification Weights. The final classification layer must be task-specific as each task involves distinguishing a potentially unique set of classes. We use a task specific affine transformation of the feature extractor output, followed by a softmax. The task-specific weights are denoted $\boldsymbol{\psi}_w^{\tau} \in \mathbb{R}^{d_f \times C^{\tau}}$ (suppressing the biases to simplify notation), where d_f is the dimension of the feature extractor output $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ and C^{τ} is the number of classes in task τ.

Task-Specific Classifier Parameters: Feature Extractor Parameters. A sufficiently flexible model must have capacity to adapt its feature representation $f_{\boldsymbol{\theta}}(\cdot)$ as well as the classification layer (e.g. compare the optimal features required for ImageNet versus Omiglot). We therefore introduce a set of local feature extractor parameters $\boldsymbol{\psi}_f^{\tau}$, and denote $f_{\boldsymbol{\theta}}(\cdot)$ the *unadapted* feature extractor, and $f_{\boldsymbol{\theta}}(\cdot; \boldsymbol{\psi}_f^{\tau})$ the feature extractor adapted to task τ.

It is critical in few-shot multi-task learning to adapt the feature extractor in a parameter-efficient manner. Unconstrained adaptation of all the feature extractor parameters (e.g. by fine-tuning (Yosinski et al., 2014)) gives flexibility, but it is also slow and prone to over-fitting (Triantafillou et al., 2020). Instead, we employ linear modulation of the convolutional feature maps as proposed by Perez et al. (2018), which adapts the feature extractor through a relatively small number of task specific parameters.

A FiLM layer (refer back to Section 2.3.3) scales and shifts the i^{th} unadapted feature map $\boldsymbol{f}_i$ in the feature extractor $\text{FiLM}(\boldsymbol{f}_i; \gamma_i^{\tau}, \beta_i^{\tau}) = \gamma_i^{\tau} \boldsymbol{f}_i + \beta_i^{\tau}$ using two task specific parameters, γ_i^{τ}

and β_i^τ. FiLM layers enable expressive feature adaptation while adding only a small number of parameters (Perez et al., 2018). For example, in our implementation we use a ResNet18 with FiLM layers after every convolutional layer. The set of task specific FiLM parameters ($\boldsymbol{\psi}_f^\tau = \{\boldsymbol{\gamma}_i^\tau, \boldsymbol{\beta}_i^\tau\}$) constitute fewer than 0.7% of the parameters in the model. Despite this, as we show in Section 5.5, they allow the model to adapt to a broad class of datasets.

4.2.2 Computing the local parameters via adaptation networks

The previous sections have specified the form of the classifier $p\left(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau\right)$ in terms of the global and task specific parameters, $\boldsymbol{\theta}$ and $\boldsymbol{\psi}^\tau = \{\boldsymbol{\psi}_f^\tau, \boldsymbol{\psi}_w^\tau\}$. The local parameters could now be learned separately for every task τ via optimization. While in practice this is feasible for small numbers of tasks (see e.g., (Rebuffi et al., 2017, 2018)), this approach is computationally demanding, requires expert oversight (e.g. for tuning early stopping), and can over-fit in the low-data regime.

Instead, CNAPs uses a function, such as a neural network, that takes the context set D^τ as an input and returns the task-specific parameters, $\boldsymbol{\psi}^\tau = \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right)$. The adaptation network has parameters $\boldsymbol{\phi}$ that will be trained on multiple tasks to learn how to produce local parameters that result in good generalization, a form of meta-learning. Sacrificing some of the flexibility of the optimisation approach, this method is comparatively cheap computationally (only involving a forward pass through the adaptation network), automatic (with no need for expert oversight), and employs explicit parameter sharing (via $\boldsymbol{\phi}$) across the training tasks.

Adaptation Network: Linear Classifier Weights. CNAPs represents the linear classifier weights $\boldsymbol{\psi}_w^\tau$ as a parameterized function of the form $\boldsymbol{\psi}_w^\tau = \boldsymbol{\psi}_w(D^\tau; \boldsymbol{\phi}_w, \boldsymbol{\psi}_f, \boldsymbol{\theta})$, denoted $\boldsymbol{\psi}_w(D^\tau)$ for brevity. There are three challenges with this approach: first, the dimensionality of the weights depends on the task ($\boldsymbol{\psi}_w^\tau$ is a matrix with a column for each class, see Figure 4.2) and thus the network must output parameters of different dimensionalities; second, the number of datapoints in D^τ will also depend on the task and so the network must be able to take inputs of variable cardinality; third, we would like the model to support continual learning. To handle the first two challenges we follow Gordon et al. (2019). First, each column of the weight matrix is generated independently from the context points from that class $\boldsymbol{\psi}_w^\tau = \left[\boldsymbol{\psi}_w\left(D_1^\tau\right), \quad \ldots, \quad \boldsymbol{\psi}_w\left(D_C^\tau\right)\right]$, an approach which scales to arbitrary numbers of classes. Second, we employ a permutation invariant architecture (Zaheer et al., 2017; Qi et al., 2017) for $\boldsymbol{\psi}_w(\cdot)$ to handle the variable input cardinality (see Section C.3 for details). Third, as permutation invariant architectures can be incrementally updated (Vartak et al., 2017), continual learning is supported (as discussed in Section 4.5).

Intuitively, the classifier weights should be determined by the representation of the data points emerging from the adapted feature extractor. We therefore input the adapted feature representation of the data points into the network, rather than the raw data points (hence the dependency of $\boldsymbol{\psi}_w$ on $\boldsymbol{\psi}_f$ and $\boldsymbol{\theta}$). To summarize, $\boldsymbol{\psi}_w(\cdot)$ is a function *on sets* that accepts as input a set of *adapted* feature representations from D_c^τ, and outputs the c^{th} column of the linear classification matrix, i.e.,

$$\boldsymbol{\psi}_w\left(D_c^\tau; \boldsymbol{\phi}_w, \boldsymbol{\psi}_f, \boldsymbol{\theta}\right) = \boldsymbol{\psi}_w\left(\{f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_m; \boldsymbol{\psi}_f\right) | \boldsymbol{x}_m \in D^\tau, \boldsymbol{y}_m = c\}; \boldsymbol{\phi}_w\right). \tag{4.2}$$

Here $\boldsymbol{\phi}_w$ are learnable parameters of $\boldsymbol{\psi}_w(\cdot)$. See Figure 4.2 for an illustration.

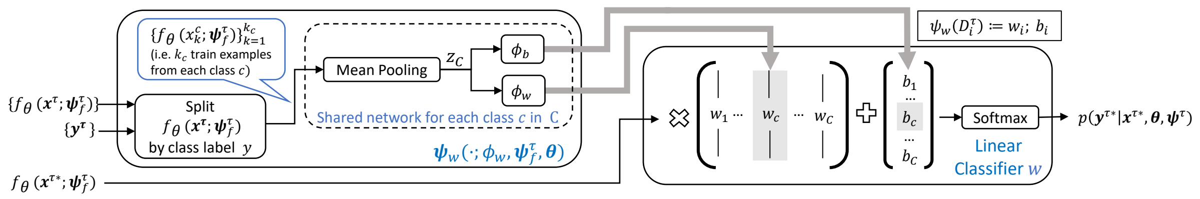

$\{f_\theta(x_k^c; \boldsymbol{\psi}_f^\tau)\}_{k=1}^{k_c}$
(i.e. k_c train examples from each class c)
$\{f_\theta(\boldsymbol{x}^\tau; \boldsymbol{\psi}_f^\tau)\}$
$\{\boldsymbol{y}^\tau\}$
Split $f_\theta(\boldsymbol{x}^\tau; \boldsymbol{\psi}_f^\tau)$ by class label y
Mean Pooling
z_C
ϕ_b
ϕ_w
Shared network for each class c in C
$\boldsymbol{\psi}_w(\cdot; \boldsymbol{\phi}_w, \boldsymbol{\psi}_f^\tau, \boldsymbol{\theta})$
$f_\theta(\boldsymbol{x}^{\tau*}; \boldsymbol{\psi}_f^\tau)$
$\psi_w(D_i^\tau) \coloneqq w_i;\ b_i$
$w_1 \cdots w_c \cdots w_C$
$b_1 \cdots b_c \cdots b_C$
Softmax
Linear Classifier w
$p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau)$

Fig. 4.2 Implementation of functional representation of the class-specific parameters $\boldsymbol{\psi}_w$. In this parameterization, $\boldsymbol{\psi}_w^c$ are the linear classification parameters for class c, and $\boldsymbol{\phi}_w$ are the learnable parameters.

Adaptation Network: Feature Extractor Parameters. CNAPS represents the task-specific feature extractor parameters $\boldsymbol{\psi}_f^\tau$, comprising the parameters of the FiLM layers $\boldsymbol{\gamma}^\tau$ and $\boldsymbol{\beta}^\tau$ in our implementation, as a parameterized function of the context-set D^τ. Thus, $\boldsymbol{\psi}_f(\cdot; \boldsymbol{\phi}_f, \boldsymbol{\theta})$ is a collection of functions (one for each FiLM layer) with parameters $\boldsymbol{\phi}_f$, many of which are shared across functions. We denote the function generating the parameters for the i^{th} FiLM layer $\boldsymbol{\psi}_f^i(\cdot)$ for brevity.

Our experiments (Section 4.5) show that this mapping requires careful parameterization. We propose a novel parameterization that improves performance in complex settings with diverse datasets. Our implementation contains two components: a task-specific representation that provides context about the task to all layers of the feature extractor (denoted $\boldsymbol{z}_{\text{G}}^\tau$), and an auto-regressive component that provides information to deeper layers in the feature extractor concerning how shallower layers have adapted to the task (denoted $\boldsymbol{z}_{\text{AR}}^i$). The input to the $\boldsymbol{\psi}_f^i(\cdot)$ network is $\boldsymbol{z}_i = (\boldsymbol{z}_{\text{G}}^\tau, \boldsymbol{z}_{\text{AR}}^i)$. $\boldsymbol{z}_{\text{G}}^\tau$ is computed for every task τ by passing the inputs $\boldsymbol{x}_n^\tau$ through a global set encoder g with parameters in $\boldsymbol{\phi}_f$.

To adapt the l^{th} layer in the feature extractor, it is useful for the system to have access to the representation of task-relevant inputs from layer $l-1$. While $\boldsymbol{z}_G$ could in principle encode how layer $l-1$ has adapted, we opt to provide this information directly to the adaptation network adapting layer l by passing the adapted activations from layer $l-1$. The auto-regressive component $\boldsymbol{z}_{\text{AR}}^i$ is computed by processing the *adapted* activations of the previous convolutional block with a layer-specific set encoder (except for the first residual block, whose auto-regressive component is given by the *un-adapted* initial pre-processing stage in the ResNet). Both the global and all layer-specific set-encoders are implemented as permutation invariant functions (Zaheer et al., 2017; Qi et al., 2017) (see Section C.3 for details). The full parameterization is illustrated in Figure 4.3, and the architecture of $\boldsymbol{\psi}_f^i(\cdot)$ networks is illustrated in Figure 4.4.

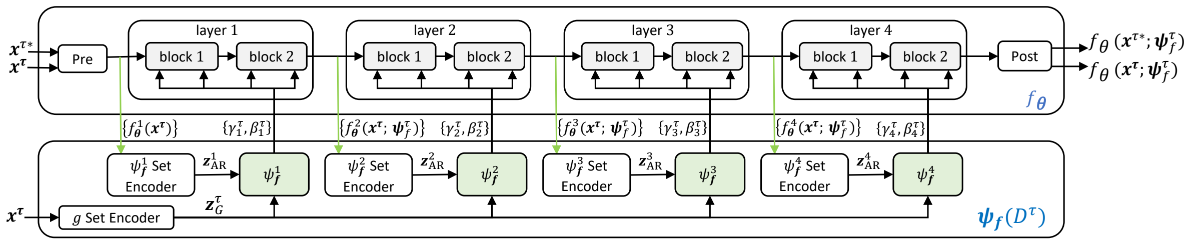


Fig. 4.3 Implementation of the feature-extractor: an independently learned set encoder g provides a fixed context that is concatenated to the (processed) activations of $\boldsymbol{x}$ from the previous ResNet block. The inputs $\boldsymbol{z}_i = (\boldsymbol{z}_{\mathrm{G}}^{\tau}, \boldsymbol{z}_{\mathrm{AR}}^{i})$ are then fed to $\psi_f^i(\cdot)$, which outputs the FiLM parameters for layer i. Green arrows correspond to propagation of auto-regressive representations. Note that the auto-regressive component $\boldsymbol{z}_{\mathrm{AR}}^{i}$ is computed by processing the *adapted* activations $\{f_\theta^i(\boldsymbol{x}; \psi_f^\tau)\}$ of the previous convolutional block.

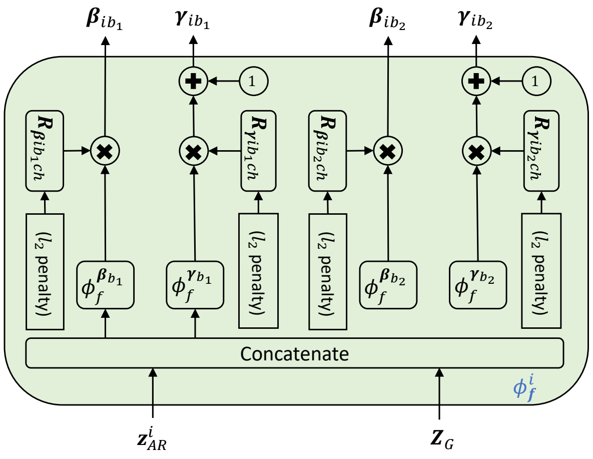


Fig. 4.4 Adaptation network ϕ_f. $\boldsymbol{R}_{\gamma_i b_j ch}$ and $\boldsymbol{R}_{\beta_i b_j ch}$ denote a vector of regularization weights that are learned with an l_2 penalty.

4.3 Model Training

The previous section has specified the model (see Figure 4.1b for a schematic). We now describe how to train the global classifier parameters $\boldsymbol{\theta}$ and the adaptation network parameters $\boldsymbol{\phi} = \{\boldsymbol{\phi}_f, \boldsymbol{\phi}_w\}$.

Training the global classifier parameters $\boldsymbol{\theta}$. A natural approach to training the model (originally employed by CNPs (Garnelo et al., 2018a)) would be to maximize the likelihood of the training data jointly over $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. However, experiments (detailed in Section A.2.1) showed that it is crucially important to adopt a two stage process instead. In the first stage, $\boldsymbol{\theta}$ are trained on a large dataset (e.g., the training set of ImageNet (Russakovsky et al., 2015; Triantafillou et al., 2020)) in a full-way classification procedure, mirroring standard pre-training. Second, $\boldsymbol{\theta}$ are fixed and $\boldsymbol{\phi}$ are trained using episodic training over all meta-training datasets in the multi-task setting. We hypothesize that two-stage training is important for two reasons: (i) during the second stage, $\boldsymbol{\phi}_f$ are trained to adapt $f_{\boldsymbol{\theta}}(\cdot)$ to tasks τ by outputting $\boldsymbol{\psi}_f^\tau$. As $\boldsymbol{\theta}$ has far more capacity than $\boldsymbol{\psi}_f^\tau$, if they are trained in the context of all tasks, there is no need for $\boldsymbol{\psi}_f^\tau$ to adapt the feature extractor, resulting in little-to-no training signal for $\boldsymbol{\phi}_f$ and poor generalization. (ii) Allowing $\boldsymbol{\theta}$ to adapt during the second phase violates the principle of "train as you test", i.e., when test tasks are encountered, $\boldsymbol{\theta}$ will be fixed, so it is important to simulate this scenario during training. Finally, fixing $\boldsymbol{\theta}$ during meta-training is desirable as it results in a dramatic decrease in training time.

Training the adaptation network parameters $\boldsymbol{\phi}$. Following the work of Garnelo et al. (2018a), we train $\boldsymbol{\phi}$ with maximum likelihood. An unbiased stochastic estimator of the log-likelihood is:

$$\hat{\mathcal{L}}\left(\boldsymbol{\phi}\right) = \frac{1}{MT} \sum_{\tau=1}^{T} \sum_{m=1}^{M_\tau} \log p\left(\boldsymbol{y}_m^{*\tau} | \boldsymbol{x}_m^{*\tau}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right), \boldsymbol{\theta}\right), \tag{4.3}$$

where $M = \sum_{\tau=1}^{T} M_\tau$ and $\{\boldsymbol{y}_m^{*\tau}, \boldsymbol{x}_m^{*\tau}, D^\tau\} \sim \hat{P}$, with $\hat{P}$ representing the data distribution (e.g., sampling tasks and splitting them into disjoint context (D^τ) and target data $\{(\boldsymbol{x}_m^{*\tau}, \boldsymbol{y}_m^{*\tau})\}_{m=1}^{M_\tau}$). Maximum likelihood training therefore naturally uses episodic context / target splits often used in meta-learning. In our experiments we use the protocol defined by Triantafillou et al. (2020) and META-DATASET for this sampling procedure.

Algorithm for Constructing Stochastic Estimator An algorithm for constructing the stochastic training objective $\hat{\mathcal{L}}(\boldsymbol{\phi}; \tau)$ for a single task τ is given in Algorithm 1. CAT$(\cdot; \boldsymbol{\pi})$ denotes a the likelihood of a categorical distribution with parameter vector $\boldsymbol{\pi}$. This algorithm can be used on a batch of tasks to construct an unbiased estimator for the auto-regressive likelihood of the task outputs.

Algorithm 1 Stochastic Objective Estimator for Meta-Training.

```
1: procedure META-TRAINING($\{\boldsymbol{x}_m^{\tau*}, \boldsymbol{y}_m^{\tau*}\}_{m=1}^M, D^\tau, \boldsymbol{\theta}, \boldsymbol{\phi}$)
2:     $\boldsymbol{\psi}_f^\tau \leftarrow \boldsymbol{\psi}_f(\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_n^\tau)|\boldsymbol{x}^\tau \in D^\tau\}; \boldsymbol{\phi}_f)$
3:     $\boldsymbol{\psi}_c^\tau \leftarrow \boldsymbol{\psi}_w(\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_n^\tau; \boldsymbol{\psi}_f^\tau)|\boldsymbol{x}^\tau \in D^\tau, \boldsymbol{y}_n^\tau = c\}; \boldsymbol{\phi}_w) \quad \forall c \in C^\tau$
4:     for $m \in 1, ..., M$ do
5:         $\boldsymbol{\pi}_m \leftarrow f_{\boldsymbol{\theta}}(\boldsymbol{x}_m^{\tau*}; \boldsymbol{\psi}_f^\tau)^T \boldsymbol{\psi}_w^\tau$
6:         $\log p(\boldsymbol{y}_m^{\tau*}|\boldsymbol{\pi}_m) \leftarrow \log \text{CAT}(\boldsymbol{y}_m^{\tau*}; \boldsymbol{\pi}_m)$
7:     end for
8:     return $\hat{\mathcal{L}}(\boldsymbol{\phi}; \tau) \leftarrow \frac{1}{M} \sum_M \log p(\boldsymbol{y}_m^{\tau*}|\boldsymbol{\pi}_m)$
9: end procedure
```

4.4 Related Work

Our work frames multi-task classification as directly modelling the predictive distribution $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\psi}(D^\tau))$. The perspective allows previous work (Finn et al., 2017; Gordon et al., 2019; Perez et al., 2018; Ravi and Larochelle, 2017; Rebuffi et al., 2017, 2018; Rusu et al., 2018; Snell et al., 2017; Triantafillou et al., 2020; Vinyals et al., 2016; Yosinski et al., 2014; Zintgraf et al., 2018; Bauer et al., 2017) to be organised in terms of i) the choice of the parameterization of the classifier (and in particular the nature of the local parameters), and ii) the function used to compute the local parameters from the training data. Refer back to the illustration of this space in Figure 2.4.

One of the inspirations for our work is conditional neural processes (CNPs) (Garnelo et al., 2018a). CNPs directly model the predictive distribution $p(\boldsymbol{y}^{\tau*}|\boldsymbol{x}^{\tau*}, \boldsymbol{\psi}(D^\tau))$ and train the parameters using maximum likelihood. Whereas previous work on CNPs has focused on homogeneous regression and classification datasets and fairly simple models, here we study multiple heterogeneous classification datasets and use a more complex model to handle this scenario. Similarly, our work can be viewed as a deterministic limit of ML-PIP (Gordon et al., 2019) which employs a distributional treatment of the local-parameters $\boldsymbol{\psi}$.

A model with design choices closely related to CNAPs is TADAM (Oreshkin et al., 2018). TADAM employs a similar set of local parameters, allowing for adaptation of both the feature extractor and classification layer. However, it uses a far simpler adaptation network (lacking auto-regressive structure) and an expensive and ad-hoc training procedure. Moreover, TADAM was applied to simple few-shot learning benchmarks (e.g. CIFAR100 and mini-ImageNet) and sees little gain from feature extractor adaptation. In contrast, we see a large benefit from adapting the feature extractor. This may in part reflect the differences in the two models, but we observe that feature extractor adaptation has the largest impact when used to adapt to *different datasets* and that two stage training is required to see this. Further differences are our usage of the CNP framework and the flexible deployment of CNAPs to continual learning and active learning (see Section 4.5).

4.5 Experiments and Results

The experiments target three key questions: (i) Can CNAPS improve performance in multi-task few-shot learning? (ii) Does the use of an adaptation network benefit computational-efficiency and data-efficiency? (iii) Can CNAPS be deployed directly to complex learning scenarios like continual learning and active learning? The experiments use the following modelling choices (see Section C.3 for full details). While CNAPS can utilize any feature extractor, a ResNet18 (He et al., 2016) is used throughout to enable fair comparison with Triantafillou et al. (2020). To ensure that each task is handled independently, batch normalization statistics (Ioffe and Szegedy, 2015) are learned (and fixed) during the pre-training phase for $\boldsymbol{\theta}$. Actual batch statistics of the test data are never used during meta-training or testing.

Few Shot Classification. The first experiment tackles a demanding few-shot classification challenge called META-DATASET (Triantafillou et al., 2020). META-DATASET is composed of ten (eight train, two test) image classification datasets. The challenge constructs few-shot learning tasks by drawing from the following distribution. First, one of the datasets is sampled uniformly; second, the "way" and "shot" are sampled randomly according to a fixed procedure; third, the classes and context / target instances are sampled. Where a hierarchical structure exists in the data (ILSVRC or OMNIGLOT), task-sampling respects the hierarchy. In the meta-test phase, the identity of the original dataset is not revealed and the tasks must be treated independently (i.e. no information can be transferred between them). Notably, the meta-training set comprises a disjoint and dissimilar set of classes from those used for meta-test. Full training details are available in Section B.2.1 and (Triantafillou et al., 2020).

Triantafillou et al. (2020) consider two stage training: an initial stage that trains a feature extractor in a standard classification setting, and a meta-training stage of all parameters in an episodic regime. For the meta-training stage, they consider two settings: meta-training only on the META-DATASET version of ILSVRC, and on all meta-training data. We focus on the latter as CNAPS rely on training data from a variety of training tasks to learn to adapt, but provide results for the former in Section 4.5. We pre-train $\boldsymbol{\theta}$ on the meta-training set of the META-DATASET version of ILSVRC, and meta-train $\boldsymbol{\phi}$ in an episodic fashion using all meta-training data. We compare CNAPS to models considered by Triantafillou et al. (2020), including their proposed method (Proto-MAML) in Table 4.1. We meta-test CNAPS on three additional held-out datasets: MNIST (LeCun et al., 2010), CIFAR10 (Krizhevsky and Hinton, 2009), and CIFAR100 (Krizhevsky and Hinton, 2009). As an ablation study, we compare a version of CNAPS that does not make use of the auto-regressive component $\boldsymbol{z}_{AR}$, and a version that uses no feature extractor adaptation. In our analysis of Table 4.1, we distinguish between two types of generalization: (i) unseen tasks (classes) in meta-training datasets, and (ii) unseen datasets.

Dataset	Finetune	MatchingNet	ProtoNet	fo-MAML	Proto-MAML	CNAPS (no ψ_f)	CNAPS (no z_{AR})	CNAPS
ILSVRC	43.1 ± 1.1	36.1 ± 1.0	44.5 ± 1.1	32.4 ± 1.0	47.9 ± 1.1	43.8 ± 1.0	**51.3** $\pm$ **1.0**	**52.3** $\pm$ **1.0**
Omniglot	71.1 ± 1.4	78.3 ± 1.0	79.6 ± 1.1	71.9 ± 1.2	82.9 ± 0.9	60.1 ± 1.3	**88.0** $\pm$ **0.7**	**88.4** $\pm$ **0.7**
Aircraft	72.0 ± 1.1	69.2 ± 1.0	71.1 ± 0.9	52.8 ± 0.9	74.2 ± 0.8	53.0 ± 0.9	76.8 ± 0.8	**80.5** $\pm$ **0.6**
Birds	59.8 ± 1.2	56.4 ± 1.0	67.0 ± 1.0	47.2 ± 1.1	70.0 ± 1.0	55.7 ± 1.0	**71.4** $\pm$ **0.9**	**72.2** $\pm$ **0.9**
Textures	**69.1** $\pm$ **0.9**	61.8 ± 0.7	65.2 ± 0.8	56.7 ± 0.7	67.9 ± 0.8	60.5 ± 0.8	62.5 ± 0.7	58.3 ± 0.7
Quick Draw	47.0 ± 1.2	60.8 ± 1.0	64.9 ± 0.9	50.5 ± 1.2	66.6 ± 0.9	58.1 ± 1.0	**71.9** $\pm$ **0.8**	**72.5** $\pm$ **0.8**
Fungi	38.2 ± 1.0	33.7 ± 1.0	40.3 ± 1.1	21.0 ± 1.0	42.0 ± 1.1	28.6 ± 0.9	**46.0** $\pm$ **1.1**	**47.4** $\pm$ **1.0**
VGG Flower	85.3 ± 0.7	81.9 ± 0.7	86.9 ± 0.7	70.9 ± 1.0	**88.5** $\pm$ **0.7**	75.3 ± 0.7	**89.2** $\pm$ **0.5**	86.0 ± 0.5
Traffic Signs	**66.7** $\pm$ **1.2**	55.6 ± 1.1	46.5 ± 1.0	34.2 ± 1.3	52.3 ± 1.1	55.0 ± 0.9	60.1 ± 0.9	60.2 ± 0.9
MSCOCO	35.2 ± 1.1	28.8 ± 1.0	39.9 ± 1.1	24.1 ± 1.1	**41.3** $\pm$ **1.0**	**41.2** $\pm$ **1.0**	**42.0** $\pm$ **1.0**	**42.6** $\pm$ **1.1**
MNIST						76.0 ± 0.8	88.6 ± 0.5	**92.7** $\pm$ **0.4**
CIFAR10						**61.5** $\pm$ **0.7**	**60.0** $\pm$ **0.8**	**61.5** $\pm$ **0.7**
CIFAR100						44.8 ± 1.0	**48.1** $\pm$ **1.0**	**50.1** $\pm$ **1.0**

Table 4.1 Few-shot classification results on META-DATASET (Triantafillou et al., 2020) using models trained on all training datasets. All figures are percentages and the $\pm$ sign indicates the 95% confidence interval over tasks. Bold text indicates the scores within the confidence interval of the highest score. Tasks from datasets below the dashed line were not used for training. Competing methods' results from (Triantafillou et al., 2020).

Unseen tasks: CNAPS achieve significant improvements over existing methods on seven of the eight datasets. The exception is the TEXTURES dataset, which has only seven test classes and accuracy is highly sensitive to the train / validation / test class split. The ablation study demonstrates that removing z_{AR} from the feature extractor adaptation degrades accuracy in most cases, and that removing all feature extractor adaptation results in drastic reductions in accuracy.

Unseen datasets: CNAPS-models outperform all competitive models at the time with the exception of FINETUNE on the TRAFFIC SIGNS dataset. Removing z_{AR} from the feature extractor decreases accuracy and removing the feature extractor adaptation entirely significantly impairs performance. The degradation is particularly pronounced when the held out dataset differs substantially from the dataset used to pretrain $\boldsymbol{\theta}$, e.g. for MNIST.

Note that the superior results when using the auto-regressive component can not be attributed to increased network capacity alone. In Section A.2.2 we demonstrate that CNAPS yields superior classification accuracy when compared to parallel residual adapters (Rebuffi et al., 2018) even though CNAPS requires significantly less network capacity in order to adapt the feature extractor to a given task.

Few-Shot Classification Results When Training on ILSVRC-2012 only Table 4.2 shows few-shot classification results on META-DATASET when trained on ILSVRC-2012 only. We emphasize that this scenario does not capture the key focus of our work, and that these results are provided mainly for completeness and compatibility with the work of Triantafillou et al.

Dataset	Finetune	MatchingNet	ProtoNet	fo-MAML	Proto-MAML	CNAPs
ILSVRC	45.8±1.1	45.0±1.1	**50.5±1.1**	36.1±1.0	**51.0±1.1**	**50.6±1.1**
Omniglot	**60.9±1.6**	52.3±1.3	60.0±1.4	38.7±1.4	**63.0±1.4**	45.2±1.4
Aircraft	**68.7±1.3**	49.0±0.9	53.1±1.0	34.5±0.9	55.3±1.0	36.0±0.8
Birds	57.3±1.3	62.2±1.0	**68.8±1.0**	49.1±1.2	**66.9±1.0**	60.7±0.9
Textures	**69.1±0.9**	64.2±0.9	66.6±0.8	56.5±0.8	**67.8±0.8**	**67.5±0.7**
Quick Draw	42.6±1.2	42.9±1.1	49.0±1.1	27.2±1.2	**53.7±1.1**	42.3±1.0
Fungi	**38.2±1.0**	34.0±1.0	**39.7±1.1**	23.5±1.0	**38.0±1.1**	30.1±0.9
VGG Flower	**85.5±0.7**	80.1±0.7	**85.3±0.8**	66.4±1.0	**86.9±0.8**	70.7±0.7
Traffic Signs	**66.8±1.3**	47.8±1.1	47.1±1.1	33.2±1.3	51.2±1.1	53.3±0.9
MSCOCO	34.9±1.0	35.0±1.0	41.0±1.1	27.5±1.1	**43.4±1.1**	**45.2±1.1**
MNIST						**70.4±0.8**
CIFAR10						**65.2±0.8**
CIFAR100						**53.6±1.0**

Table 4.2 Few-shot classification results on META-DATASET (Triantafillou et al., 2020) using models trained on ILSVRC-2012 only. All figures are percentages and the ± sign indicates the 95% confidence interval. Bold text indicates the highest scores that overlap in their confidence intervals. Results from competitive methods from (Triantafillou et al., 2020)

(2020). In particular, our method relies on training the parameters ϕ to adapt the conditional predictive distribution to new datasets. In this setting, the model is never presented with data that has not been used to pre-train $\boldsymbol{\theta}$, and therefore cannot learn to appropriately adapt the network to new datasets. Despite this, CNAPs demonstrate competitive results with the methods evaluated by Triantafillou et al. (2020) even in this scenario.

Feature Extractor Parameter Learning Figure 4.5 shows t-SNE (Maaten and Hinton, 2008) plots that visualize the output of the set encoder z_G and the FiLM layer parameters following the first and last convolutional layers of the feature extractor at test time. Even with unseen test data, the set encoder has learned to clearly separate examples arising from diverse datasets. The FiLM generators learn to generate feature extractor adaptation parameters unique to each dataset. The only significant overlap in the FiLM parameter plots is between CIFAR10 and CIFAR100 datasets which are closely related. This visualization demonstrates that the model is able to learn meaningful task and dataset level representations and parameterizations. The results support the hypothesis that learning to adapt key parts of the network is more robust and achieves significantly better performance than existing approaches.

FiLM Parameter Learning Performance: Speed-Accuracy Trade-off. CNAPs generate FiLM layer parameters for each task τ at test time using the adaptation network $\psi_f(D^\tau)$. It is also possible to learn the FiLM parameters via gradient descent (see (Rebuffi et al., 2017, 2018)). Here we compare CNAPs to this approach. Figure 4.6 shows plots of 5-way classification accuracy versus time for four held out data sets as the number of shots was varied. For gradient

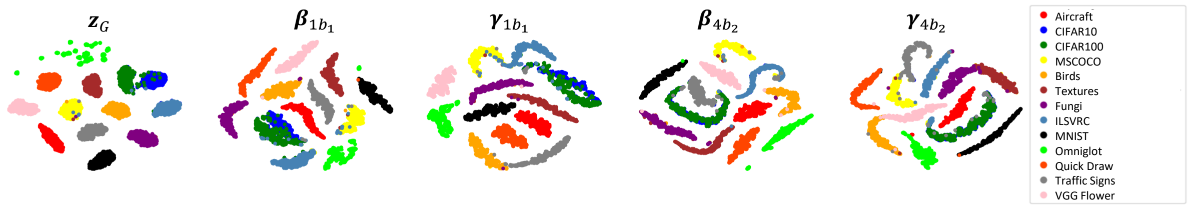


Fig. 4.5 t-SNE plots of the output of the set encoder z_G and the FiLM layer parameters at the start ($\beta_{1b1}, \gamma_{1b1}$) and end ($\beta_{4b2}, \gamma_{4b2}$) of the feature extraction process at test time.

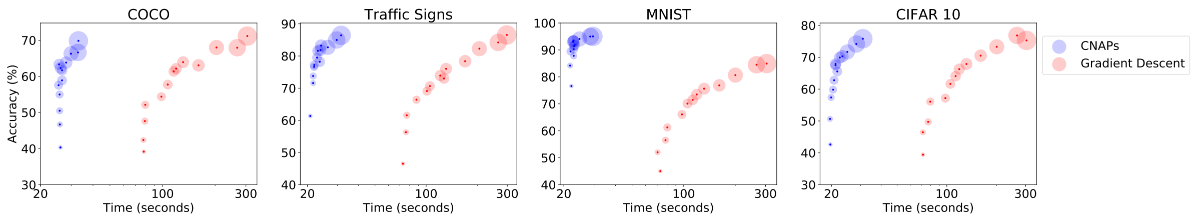


Fig. 4.6 Comparing CNAPs to gradient based feature extractor adaptation: accuracy on 5-way classification tasks from withheld datasets as a function of processing time. Dot size reflects shot number (1 to 25 shots).

descent, we used a fixed learning rate of 0.001 and took 25 steps for each point. The overall time required to produce the plot was 1274 and 7214 seconds for CNAPs and gradient approaches, respectively, on a NVIDIA Tesla P100-PCIE-16GB GPU. CNAPs is at least 5 times faster at test time than gradient-based optimization requiring only a single forward pass through the network while gradient based approaches require multiple forward and backward passes. Further, the accuracy achieved with adaptation networks is significantly higher for fewer shots as it protects against over-fitting. For large numbers of shots, gradient descent catches up, albeit slowly.

4.6 Continual Learning

In continual learning (Ring, 1997), new tasks appear over time and existing tasks may change. The goal is to adapt accordingly, but without retaining old data which is challenging for artificial systems. To demonstrate the the versatility CNAPs we show that, although it has not been explicitly trained for continual learning, we are able to apply the same model trained for the few-shot classification experiments (without the auto-regressive component) to standard continual learning benchmarks on held out datasets: Split MNIST (Zenke et al., 2017) and Split CIFAR100 (Chaudhry et al., 2018). Refer to Section B.2.2 for more details on these benchmarks.

We modify the model to compute running averages for the representations of both ψ_w^τ and ψ_f^τ, in this way it performs incremental updates using the new data and the old model, and does not need to access old data. In particular, we store a compact representation of our training data that can be updated at each step of the continual learning procedure. Notice that Figure 4.2 indicates that the functional representation of our linear classification layer $\psi_w^\tau(\cdot)$ contains a mean pooling layer that combines the per-class output of our feature extractor $\{f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n^\tau; \boldsymbol{\psi}_f\right) | \boldsymbol{x}_n^\tau \in D^\tau, \boldsymbol{y}_n^\tau = c\}$. The result of this pooling,

$$\boldsymbol{z}_c = \frac{1}{N} \sum f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n^\tau; \boldsymbol{\psi}_f\right) \tag{4.4}$$

where $N = |\{f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n^\tau; \boldsymbol{\psi}_f\right) | \boldsymbol{x}_n^\tau \in D^\tau, \boldsymbol{y}_n^\tau = c\}|$, is supplied as input to the network $\psi_w(\cdot)$. This network yields the class conditional parameters of the linear classifier $\boldsymbol{\psi}_w^\tau$, resulting in (along with the feature extractor parameters $\boldsymbol{\psi}_f^\tau$) the full paramterization of $\boldsymbol{\psi}^\tau$. We store $\boldsymbol{z}_c$ as the training dataset representation for, class c.

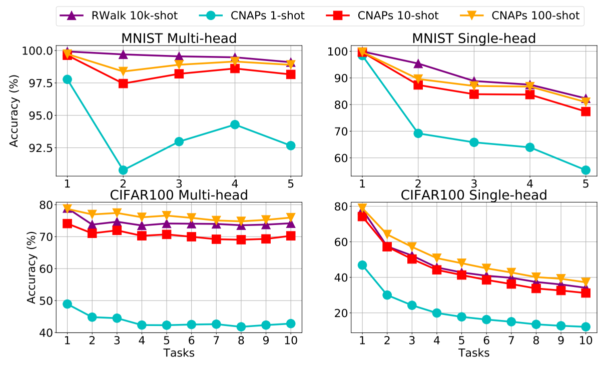
If at any point in our continual learning procedure we observe new training data for class c we can update our representation for class c by computing $\boldsymbol{z}_c' = \frac{1}{N'} \sum f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_n^{\tau\prime}; \boldsymbol{\psi}_f\right)$ the pooled average resulting from N' new training examples $\boldsymbol{x}_m^{\tau\prime}$ for class c. We then update $\boldsymbol{z}_c$ with the weighted average: $\boldsymbol{z}_c \leftarrow \frac{N\boldsymbol{z}_c + N'\boldsymbol{z}_c'}{N+N'}$. At prediction time, we supply $\boldsymbol{z}_c$ to $\boldsymbol{\psi}_w(\cdot)$ to produce classification parameters for class c.

Similar to the input to $\boldsymbol{\psi}_w^\tau(\cdot)$, the input to $\boldsymbol{\psi}_f^\tau(\cdot)$ also contains a mean-pooled representation, this time of the entire training dataset $\boldsymbol{z}_G^\tau$. This representation is also stored and updated in the same way.

One issue with our procedure is that it is not completely invariant to the order in which we observe the sequence of training data during our continual learning procedure. The feature extractor adaptation parameters are only conditioned on the most recent training data, meaning that if data from class c is not present in the most recent training data, z_c was generated using "old" feature extractor adaptation parameters (from a previous time step). This creates a potential disconnect between the classification parameters from previous time steps and the feature extractor output. Fortunately, in our experiment we noticed little within dataset variance for the adaptation parameters. Since all of our experiments on continual learning were within a single dataset, this did not seem to be an issue as CNAPS was able to achieved good performance. However, for continual learning experiments that contain multiple datasets, we anticipate that this issue will need to be addressed.

Figure 4.7 (left) shows the accumulated multi- and single-head (Chaudhry et al., 2018) test accuracy averaged over 30 runs. Figure 4.7 (right) shows average results at the final task comparing to SI (Zenke et al., 2017), EWC (Kirkpatrick et al., 2017), VCL (Nguyen et al., 2017), and Riemannian Walk (Chaudhry et al., 2018). Please refer to Section B.2.2 for details on how the experiments were executed.

Figure 4.7 demonstrates that CNAPS naturally resists catastrophic forgetting (Kirkpatrick et al., 2017) and compares favourably to competing methods, despite the fact that it was not exposed to these datasets during training, observes orders of magnitude fewer examples, and was not trained explicitly to perform continual learning. CNAPS performs similarly to,



Method	*MNIST* Multi	*MNIST* Single	*CIFAR100* Multi	*CIFAR100* Single
SI	99.3	57.6	73.2	22.8
EWC	99.3	55.8	72.8	23.1
VCL	98.5 $\pm$ 0.4	-	-	-
RWalk	99.3	82.5	74.2	34.0
CNAPs	98.9 $\pm$ 0.2	80.9 $\pm$ 0.9	76.0 $\pm$ 0.5	37.2 $\pm$ 0.6

Fig. 4.7 Continual learning classification results on Split MNIST and Split CIFAR100 using a model trained on all training datasets. (Left) The plots show accumulated accuracy averaged over 30 runs for both single- and multi-head scenarios. (Right) Average accuracy at final task computed over 30 experiments (all figures are percentages). Errors are one standard deviation. Additional results from (Chaudhry et al., 2018; Swaroop et al., 2019).

or better than, the state-of-the-art Riemannian Walk method which departs from the pure continual learning setting by maintaining a small number of training samples across tasks. Conversely, CNAPs has the advantage of being exposed to a larger range of datasets and can therefore leverage task transfer. We emphasize that this is not meant to be an "apples-to-apples" comparison, but rather, the goal is to demonstrate the out-of-the-box versatility and strong performance of CNAPs in new domains and learning scenarios.

Figure 4.7 showed the average performance as more tasks were observed for the single and multi head settings. Figure 4.8 and Figure 4.9 provide more complete results, detailing the performance through "time" at the task level. Figure 4.8 details the performance of CNAPs

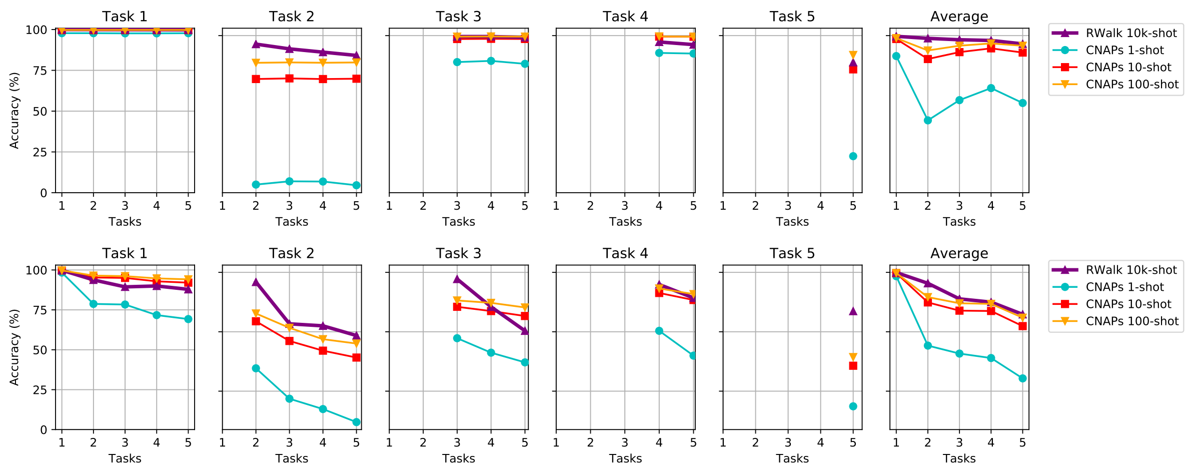


Fig. 4.8 Continual learning results on Split MNIST. Top row is multi-head, bottom row is single-head.

(with varying number of observed examples) and Riemannian Walk (RWalk) (Chaudhry et al., 2018) on the five tasks of Split MNIST through time. Note that RWalk makes explicit use of training data from previous time steps when new data is observed, while CNAPs do not.

Figure 4.8 implies that CNAPs is competitive with RWalk in this scenario, despite seeing far less data per task, and not using old data to retrain the model at every time-step. Further, we see that CNAPs is naturally resistant to forgetting, as it uses internal task representations to maintain important information about tasks seen at previous time-steps.

Figure 4.9 demonstrates that CNAPs maintains similar results when scaling up to considerably more difficult datasets such as CIFAR100. Here too, CNAPs has not been trained on this dataset, yet demonstrates performance comparable to (and even better than) RWalk, a method explicitly trained for this task that makes use of samples from previous tasks at each time step.

4.7 Active Learning

Active learning Cohn et al. (1996); Settles (2012) requires accurate data-efficient learning that returns well-calibrated uncertainty estimates. Figure 4.10 compares the performance of CNAPs and prototypical networks using two standard active learning acquisition functions (variation ratios and predictive entropy (Cohn et al., 1996)) against random acquisition on the FLOWERS dataset and three representative held-out languages from OMNIGLOT. Figure 4.10 show that CNAPs achieves higher accuracy on average than prototypical networks. Moreover, CNAPs achieves significant improvements over random acquisition, whereas prototypical networks do not. These tests indicates that CNAPs is more accurate and suggest that CNAPs has better calibrated uncertainty estimates than prototypical networks. It is not clear why this is the case, but a plausible explanation is that CNAPs employs FiLM layers to effectively adapt the feature extractor to unseen tasks which has shown to increase accuracy (see Table 4.1), whereas prototypical networks has no such mechanism.

For completeness, Figure 4.11 shows the active learning results from all twenty held-out languages in Omniglot. Figure 4.11 demonstrates that in almost all held-out languages, using the predictive distribution of CNAPs not only improves overall performance, but also enables the model to make use of standard acquisition functions (Cohn et al., 1996) to improve data efficiency over random acquisition. In contrast, we see that in most cases, random acquisition performs as well or better than acquisition functions that rely on the predictive distribution of Prototypical Networks. This provides empirical evidence that in addition to achieving overall better performance, the predictive distribution of CNAPs is more calibrated, and thus better suited to tasks such as active learning that require uncertainty in predictions.

4.8 Summary

This chapter has introduced CNAPS, an automatic, fast and flexible modelling approach for multi-task classification. We have demonstrated that at the time CNAPS achieve state-of-the-art performance on the META-DATASET challenge, and can be deployed "out-of-the-box" to diverse learning scenarios such as continual and active learning where they are competitive with the state-of-the-art. Future avenues of research are to consider the exploration of the design space by introducing gradients and function approximation to the adaptation mechanisms, as well as generalizing the approach to distributional extensions of CNAPS (Garnelo et al., 2018b; Kim et al., 2019).

4.9 Epilogue

Three recent publications have recently built directly on the CNAPS work. We discuss each one briefly below. Following that, we summarize two very recent papers, one that demonstrates superior results on Meta-Dataset and another that presents a novel approach to learn FiLM layer parameters to modulate feature extractor activations.

Improved Few-Shot Visual Classification Bateni et al. (2019) improve on the CNAPS classification accuracy results by replacing the CNAPS linear classifier stage with a Prototypical Networks (Snell et al., 2017) based classifier that uses the Mahalanobis distance metric instead of the usual Euclidean distance metric. Their 'Simple CNAPS' system improves upon the CNAPS classification accuracy overall by 6.1% and reduces the number of learnable parameters by 9.2%. The parameter savings are due to the fact that the 'Simple CNAPS' metric based classifier stage has no learnable parameters and they do not require the linear classifier adaptation network to generate any classifier parameters. A possible explanation for the superior classification accuracy of Simple CNAPs is that it is based on the principles of linear discriminant analysis (LDA) (Fisher, 1936; Duda et al., 2012). If the assumption that the data is normally distributed holds, LDA can be more effective than logistic regression (Hastie et al., 2009; Efron, 1975), especially if the sample size is small (Pohar et al., 2004) (which is the case in few-shot learning).

The authors also show that the prototypical networks classifier with the simpler Euclidean distance metric yields slightly better classification results than the VERSA classifier used in CNAPS. This warrants further investigation as both are conceptually similar, with the main difference being that prototypical networks with the Euclidean distance metric uses a simpler hypernet to generate the linear classifier weights. Their work appeared in the proceedings of CVPR 2020.

LEEP: A New Measure to Evaluate Transferability of Learned Representations Nguyen et al. (2020) develop a measure to quantify the transferability of representations learned by classifiers. The measure is called 'Log Expected Empirical Prediction' or LEEP. LEEP is computationally efficient as the bottleneck step is simply a forward pass of the target data through the source network and retraining the source network on the target data is not required. It is also the first method to to measure transferability in meta-transfer learning algorithms. As such, the authors demonstrate that LEEP can predict the performance of CNAPS, showing that a higher LEEP score corresponds to higher test accuracies in CNAPS. Their work appeared in the proceedings of ICML 2020.

Meta-learning from Poor-Quality Videos for Personalised Object Recognition (Note: I am a co-author on this work). The goal of this paper is to develop a personal object recognizer to allow blind or visually impaired users to locate objects of interest to them. CNAPS is ideal for this scenario because the system can be 'personalized' by merely performing a forward pass through the system with a small number of examples of each of the personal objects to be recognized which form the context set. This forward pass will generate the FiLM layer parameters and linear classifier weights and biases such that the system can now classify arbitrary inputs, potentially on a mobile device equipped with a camera. The authors posit that it is more effective for the blind or visually impaired user to record videos of their personal objects to be recognized as opposed to a still image, as a video will likely contain at least a few frames of a good capture of the object and provide a variety of views of the object. As a result, this work extends CNAPS to support: (i) video (as opposed to still image) input; and (ii) a task-specific frame weighter that filters out (via down-weighting) irrelevant frames from a video sequence to be classified. The resulting system is called 'V-CNAPs' and the experimental results evaluated on poor quality videos produced by visually impaired and blindfolded users demonstrate the effectiveness of the basic CNAPS model and the new task-specific frame weighter. This work is under review for ECCV 2020.

A Universal Representation Transformer Layer for Few-Shot Image Classification In a very recent paper, Liu et al. (2020) demonstrate the best results so far on the *in-domain* datasets on the Meta-Dataset benchmark, but fall short of the results by Bateni et al. (2019) on the more important *out-of-domain* datasets. They achieve this by pretraining a feature extractor for each of the the eight in-domain datasets and then meta-learn a dot-product, multi-head attention layer that weights the feature embeddings from each of the eight feature extractors. The weighted embeddings are then fed into a classification layer. While this concept yields superior accuracy on the in-domain datasets due to the additional pretraining (CNAPS pretrains on only a single dataset), it does not generalize well to datasets that it has not pretrained on. One piece of empirical evidence that supports this view is the work of Bateni et al. (2019) which extends CNAPS and achieves superior results compared to Liu et al. (2020) on the held-out datasets demonstrating superior generalization via FiLM layers added to a feature extractor pretrained on a single dataset.

Cross-domain few-shot classification via learned feature-wise transformation In another recent paper, like CNAPS, Tseng et al. (2020) use FiLM layers (Perez et al., 2018) (though they refer to them as *feature-wise transformation* layers) to adapt a feature extractor to diverse tasks unseen during meta-training. However, the FiLM layer coeficients (γ and $\boldsymbol{\beta}$) are parameterized by a normal distribution where the mean is fixed to one for γ and to zero for $\boldsymbol{\beta}$ and the standard deviations are meta-learned via stochastic gradient descent during meta-training. At meta-test time, the FiLM layer parameters are sampled from the learned distribution. This differs significantly from the CNAPS approach where the FiLM layer parameters are generated by a hypernetwork as a function of the task context set. The authors did not evaluate their approach on META-DATASET, so the performance of the two systems cannot be directly compared.

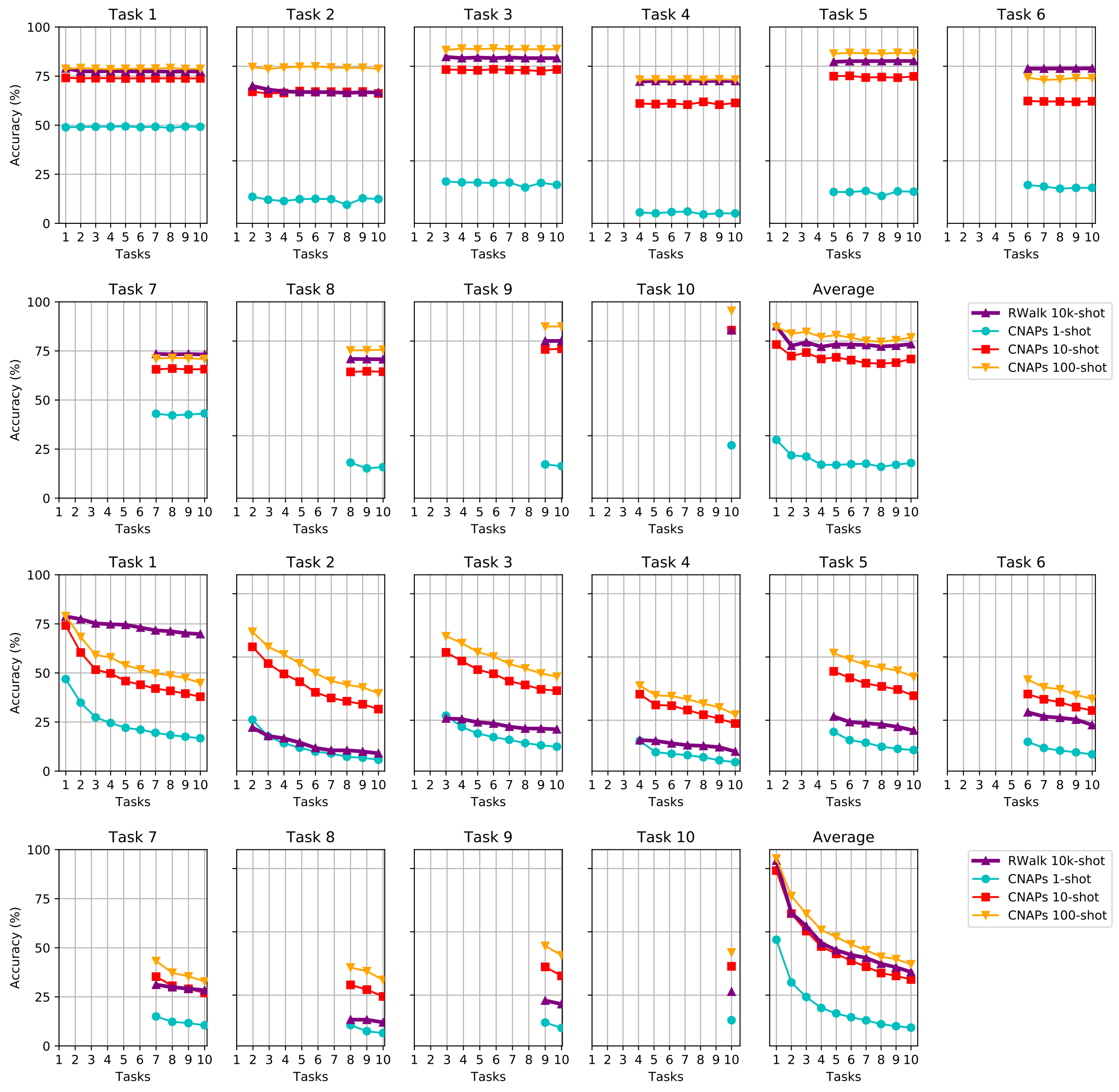


Fig. 4.9 Continual learning results on Split CIFAR100. Top two rows are multi-head, bottom two rows are single-head.

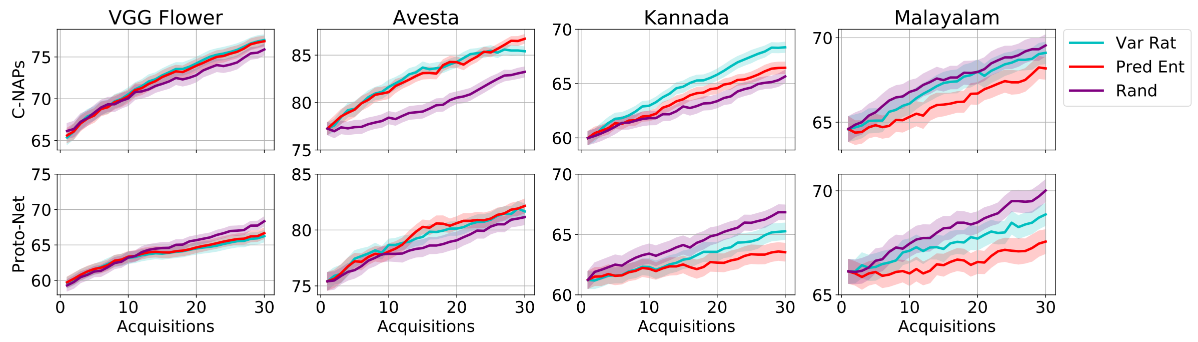


Fig. 4.10 Accuracy vs active learning iterations for held-out classes / languages. (Top) CNAPs and (bottom) prototypical networks. Error shading is one standard error. CNAPs achieves better accuracy than prototypical networks and improvements over random acquisition, whereas prototypical networks do not.

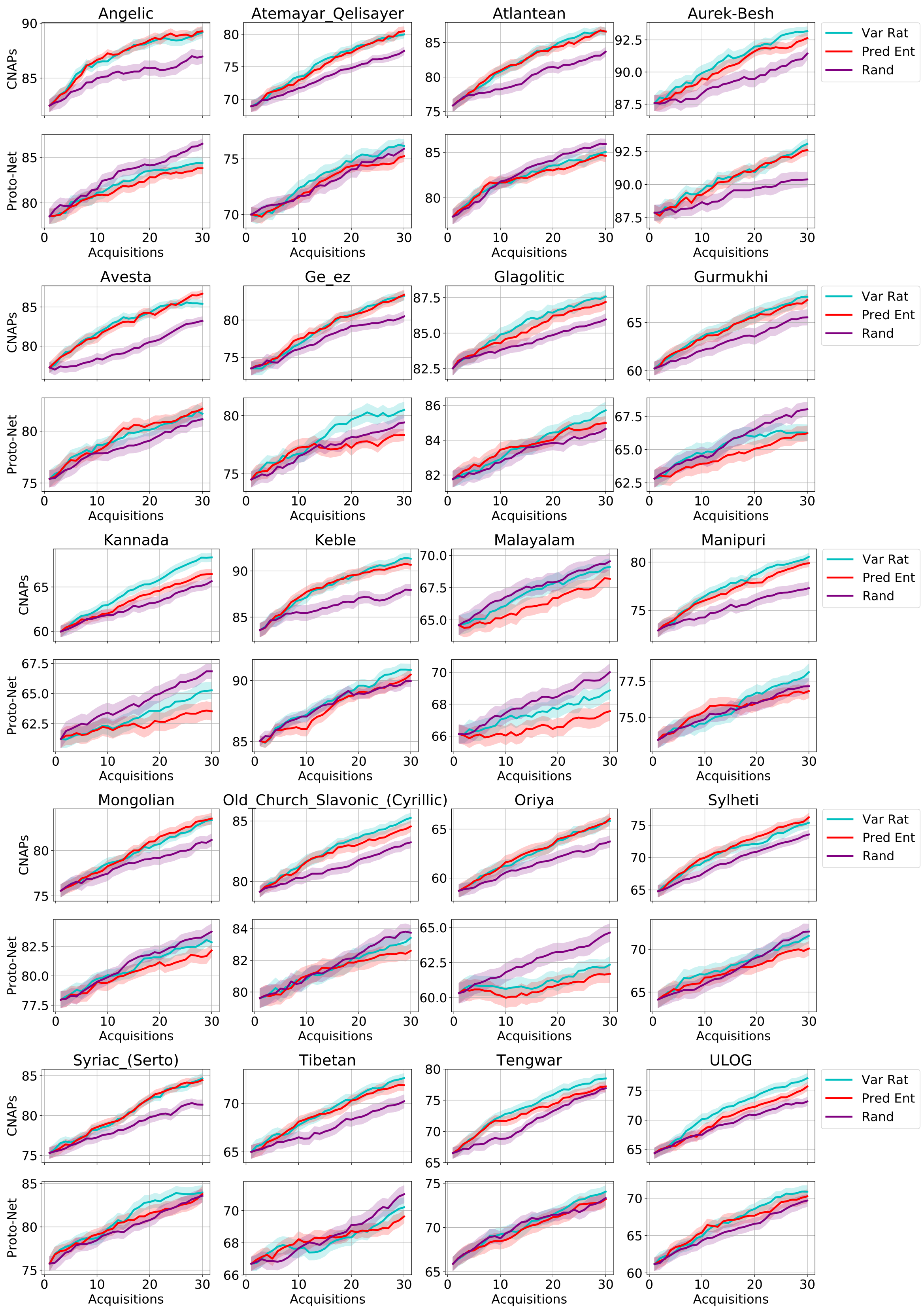


Fig. 4.11 Active learning results on all twenty held-out Omniglot languages. (Top) CNAPs and (bottom) prototypical networks. Error shading is one standard error.

Chapter 5

TaskNorm: Rethinking Batch Normalization for Meta-Learning

5.1 Introduction

Batch normalization has become an essential component of deep learning systems as it significantly accelerates the training of neural networks by allowing the use of higher learning rates and decreasing the sensitivity to network initialization. In this chapter, we investigate the use of batch normalization in meta-learning methods. Recent approaches to meta-learning rely on increasingly deep neural network based architectures to achieve state-of-the-art performance in a range of benchmark tasks (Finn et al., 2017; Mishra et al., 2018; Triantafillou et al., 2020; Requeima et al., 2019b). When constructing very deep networks, a standard component is the use of normalization layers (NL). In particular, in the image-classification domain, batch normalization (BN; Ioffe, 2017) is crucial to the successful training of very deep convolutional networks.

However, as we discuss in Section 5.3, standard assumptions of the meta-learning scenario violate assumptions of BN and vice-versa, complicating the deployment of BN for meta-learning models. Many papers proposing novel meta-learning approaches employ different forms of BN for the proposed models, and some forms make implicit assumptions that, while improving benchmark performance, may result in potentially undesirable behaviours. Moreover, as we demonstrate in Section 5.5, performance of the models can vary significantly based on the form of BN employed, confounding comparisons across methods. Further, naive adoption of BN for meta-learning does not reflect the statistical structure of the data-distribution in this scenario. In contrast, we propose a novel variant of BN – TaskNorm – that explicitly accounts for the statistical structure of the data distribution. We demonstrate that by doing so, TaskNorm further accelerates training of models using meta-learning while achieving improved test-time performance. Our main contributions are as follows:

- We identify and highlight several issues with BN schemes used in the recent meta-learning literature.
- We propose TASKNORM, a novel variant of BN which is tailored for the meta-learning setting.
- In experiments with fourteen datasets, we demonstrate that TASKNORM consistently outperforms competing methods, while making less restrictive assumptions than its strongest competitor.

This chapter is based on the ICML 2020 conference publication entitled 'TASKNORM: Rethinking Batch Normalization for Meta-Learning' (Bronskill et al., 2020). My contributions were to develop the TASKNORM idea as an extension to Jonathan Gordon's METABN concept, jointly devise the set of experiments, independently write all the code and perform the experiments, jointly analyze the results, and jointly write the paper.

5.2 Background and Related Work

In this section we lay the necessary groundwork for our investigation of batch normalization in the meta-learning scenario.

5.2.1 Normalization Layers in Deep Learning

Normalization layers (NL) for deep neural networks were introduced by Ioffe and Szegedy (2015) to accelerate the training of neural networks by allowing the use of higher learning rates and decreasing the sensitivity to network initialization. Since their introduction, they have proven to be crucial components in the successful training of ever-deeper neural architectures. Our emphasis is the few-shot image classification setting, and as such we focus on NLs for 2D convolutional networks. We denote input images $\boldsymbol{x} \in \mathbb{R}^{C \times W \times H}$ where W is the image width, H the image height, C the number of image channels and image labels $\boldsymbol{y}$. The input to a NL is $A = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_B)$, a batch of B image-shaped activations or pre-activations, to which the NL is applied as

$$\boldsymbol{a}'_n = \gamma \left(\frac{\boldsymbol{a}_n - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2 + \epsilon}} \right) + \boldsymbol{\beta}, \tag{5.1}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the *normalization moments*, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learned parameters, ϵ is a small scalar to prevent division by 0, and operations between vectors are element-wise. NLs differ primarily by how the normalization moments are computed. The first such layer – batch normalization (BN) – was introduced by Ioffe and Szegedy (2015). A BN layer distinguishes between training and test modes. At training time, BN computes the moments as

$$\boldsymbol{\mu}_{BN_c} = \frac{1}{BHW} \sum_{b=1}^{B} \sum_{w=1}^{W} \sum_{h=1}^{H} \boldsymbol{a}_{bwhc}, \tag{5.2}$$

$$\boldsymbol{\sigma}^2_{BN_c} = \frac{1}{BHW} \sum_{b=1}^{B} \sum_{w=1}^{W} \sum_{h=1}^{H} (\boldsymbol{a}_{bwhc} - \boldsymbol{\mu}_{BN_c})^2. \tag{5.3}$$

Here, $\boldsymbol{\mu}_{BN}, \boldsymbol{\sigma}^2_{BN}, \boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^C$. As $\boldsymbol{\mu}_{BN}$ and $\boldsymbol{\sigma}^2_{BN}$ depend on the batch of observations, BN can be susceptible to failures if the batches at test time differ significantly from training batches, e.g., for streaming predictions. To counteract this, at training time, a running mean and variance, $\boldsymbol{\mu}_r, \boldsymbol{\sigma}_r \in \mathbb{R}^C$, are also computed for each BN layer over all training tasks and stored. At test time, test activations $\boldsymbol{a}$ are normalized using Equation (5.1) with the statistics $\boldsymbol{\mu}_r$ and $\boldsymbol{\sigma}_r$ in place of the batch statistics. Importantly, BN relies on the implicit assumption that D comprises i.i.d. samples from some underlying distribution.

More recently, additional NLs have been introduced. Many of these methods differ from standard BN in that they normalize each instance independently of the remaining instances in the batch, making them more resilient to batch distribution shifts at test time. These include instance normalization (Ulyanov et al., 2016), layer normalization (Ba et al., 2016), and group normalization (Wu and He, 2018). These are discussed further in Section 5.3.4.

5.2.2 Desiderata for Meta-Learning Normalization Layers

As modern approaches to meta-learning systems routinely employ deep networks, NLs become essential for efficient training and optimal classification performance. For BN in the standard supervised settings, i.i.d. assumptions about the data distribution imply that estimating moments from the training set will provide appropriate normalization statistics for test data. However, this does not hold in the meta-learning scenario, for which data points are only assumed to be i.i.d. within a specific task. Therefore, the choice of what moments to use when applying a NL to the context and target set data points, during both meta-training and meta-test time, is key.

As a result, recent meta-learning approaches employ several normalization procedures that differ according to these design choices. A range of choices are summarized in Figure 5.1. As we discuss in Section 5.3 and demonstrate with experimental results, some of these have implicit, undesirable assumptions which have significant impact on both predictive performance and training efficiency. We argue that an appropriate NL for the meta-learning scenario requires consideration of the data-generating assumptions associated with the setting. In particular, we propose the following desiderata for a NL when used for meta-learning:

1. Improves speed and stability of training without harming test performance (accuracy or log-likelihood);

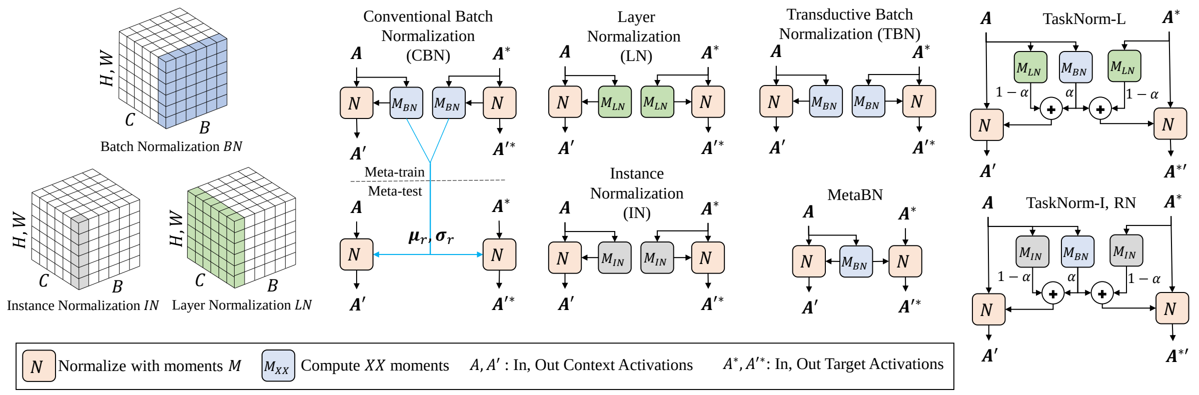

H, W
C
B
Batch Normalization BN
Instance Normalization IN
Layer Normalization LN
Conventional Batch Normalization (CBN)
Meta-train
Meta-test
μ_r, σ_r
Layer Normalization (LN)
Instance Normalization (IN)
Transductive Batch Normalization (TBN)
MetaBN
TaskNorm-L
TaskNorm-I, RN
A
A*
A'
A'*
A*'
N
M_BN
M_LN
M_IN
1 − α
α
N Normalize with moments M
M_XX Compute XX moments
A, A' : In, Out Context Activations
A*, A'*: In, Out Target Activations

Fig. 5.1 A range of options for batch normalization for meta-learning. The cubes on the left depict the dimensions over which different moments are calculated for normalization of 2D convolutional layers. The computational diagrams on the right show how context and target activations are processed for various normalization methods. For all methods except conventional BN (CBN), the processing is identical at meta-train and meta-test time. Cube diagrams are derived from Wu and He (2018).

2. Works well across a range of context set sizes;
3. Is non-transductive, thus supporting inference at meta-test time in a variety of circumstances.

A non-transductive meta-learning system makes predictions for a single test set label conditioned only on a single input and the context set, while a transductive meta-learning system conditions on additional samples from the test set:

$$\underbrace{p(\boldsymbol{y}_i^{\tau *}|\boldsymbol{x}_i^{\tau *}, D^\tau)}_{\text{non-transductive}} \quad ; \quad \underbrace{p(\boldsymbol{y}_i^{\tau *}|\boldsymbol{x}_{i=1:m}^{\tau *}, D^\tau)}_{\text{transductive}} . \tag{5.4}$$

We argue that there are two key issues with transductive meta-learners. The first is that transductive learning is sensitive to the distribution over the target set used during meta-training, and as such is less generally applicable than non-transductive learning. For example, transductive learners may fail to make good predictions if target sets contain a different class balance than what was observed during meta-training, or if are required to make predictions for one example at a time. Transductive learners can also violate privacy constraints. In Table 5.1 and Section 5.5.3, we provide empirical demonstrations of these failure cases.

The second issue is that transductive learners have more information available than non-transductive learners at prediction time, which may lead to unfair comparisons. It is worth noting that some meta-learning algorithms are specifically designed to leverage transductive inference (e.g., Ren et al., 2018; Liu et al., 2019), though we do not discuss them in this work. In Section 5.5.3 we demonstrate that there are significant performance differences for a model when trained transductively versus non-transductively.

5.3 Normalization Layers for Meta-learning

In this section, we discuss several normalization schemes that can and have been applied in the recent meta-learning literature, highlighting the modelling assumptions and effects of different design choices. Throughout, we assume that the meta-learning algorithm is constructed such that the context-set inputs are passed through every neural-network module that the target set inputs are passed through at prediction time. This implies that moments are readily available from both the context and target set observations for any normalization layer, and is the case for many widely-used meta-learning models (e.g., Finn et al., 2017; Snell et al., 2017; Gordon et al., 2019).

To illustrate our arguments, we provide experiments with MAML running simple, but widely used few-shot learning tasks from the Omniglot (Lake et al., 2011) and miniImagenet (Ravi and Larochelle, 2017) datasets. The results of these experiments are provided in Table 5.1, and full experimental details in Section B.3.

5.3.1 Conventional Usage of Batch Normalization (CBN)

We refer to *conventional* batch normalization (CBN) as that defined by Ioffe and Szegedy (2015) and as outlined in Section 5.2.1. In the context of meta-learning, this involves normalizing tasks with computed moments at meta-train time, and using the accumulated running moments to normalize the tasks at meta-test time (see CBN in Figure 5.1).

We highlight two important issues with the use of CBN for meta-learning. The first is that, from the graphical model perspective (refer to Figure 2.3), this is equivalent to lumping $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ with the global parameters $\boldsymbol{\theta}$, i.e., they are learned from the meta-training set and shared across all tasks at meta-test time. We might expect CBN to perform poorly in meta-learning applications since the running moments are *global* across all tasks while the task data is only i.i.d. *locally* within a task, i.e., CBN does not satisfy desiderata 1. This is corroborated by our results (Table 5.1), where we demonstrate that using CBN with MAML results in very poor predictive performance - no better than chance. The second issue is that, as demonstrated by Wu and He (2018), using small batch sizes leads to inaccurate moments, resulting in significant increases in model error. Importantly, the small batch setting may occur often in meta-learning, for example in the 1-shot scenario. Thus, CBN does not satisfy desiderata 2.

Despite these issues, CBN is sometimes used, e.g., by Snell et al. (2017), though testing was performed only on Omniglot and miniImagenet where the distribution of tasks is homogeneous (Triantafillou et al., 2020).

5.3.2 Batch Renormalization (BRN)

Batch renormalization (BRN; Ioffe, 2017) is intended to mitigate the issue of non-identically distributed and/or small batches while retaining the training efficiency and stability of CBN.

In BRN, the CBN algorithm is augmented with an affine transform with batch-derived parameters which correct for the batch statistics being different from the overall population. The normalized activations of a BRN layer are computed as follows:

$$\boldsymbol{a}'_n = \boldsymbol{\gamma}\left(r\left(\frac{\boldsymbol{a}_n - \boldsymbol{\mu}_{BN}}{\boldsymbol{\sigma}_{BN} + \epsilon}\right) + d\right) + \boldsymbol{\beta},$$

where

$$\begin{aligned} r =& \texttt{stop\_grad}\left(\texttt{clip}_{[1/r_{max}, r_{max}]}\left(\frac{\boldsymbol{\sigma}_{BN}}{\boldsymbol{\sigma}_r}\right)\right), \\ d =& \texttt{stop\_grad}\left(\texttt{clip}_{[-d_{max}, d_{max}]}\left(\frac{\boldsymbol{\mu}_{BN} - \boldsymbol{\mu}_r}{\boldsymbol{\sigma}_r}\right)\right). \end{aligned}$$

Here $\texttt{stop\_grad}(\cdot)$ denotes a gradient blocking operation, and $\texttt{clip}_{[a,b]}$ denotes an operation returning a value in the range $[a, b]$. Like CBN, BRN is not well suited to the meta-learning scenario as it does not map directly to the hierarchical form of meta-learning models. In Section 5.5, we show that using BRN can improve predictive performance compared to CBN, but still performs significantly worse than competitive approaches. Table 5.1 shows that batch renormalization performs poorly when using MAML.

5.3.3 Transductive Batch Normalization (TBN)

Another approach is to do away with the running moments used for normalization at meta-test time, and replace these with context / target set statistics. Here, context / target set statistics are used for normalization, both at meta-train *and* meta-test time. This is the approach taken by the authors of MAML (Finn et al., 2017),[1] and, as demonstrated in our experiments, seems to be crucial to achieve the reported performance. From the graphical model perspective, this implies associating the normalization statistics with neither $\boldsymbol{\theta}$ nor $\boldsymbol{\psi}$, but rather with a special set of parameters that is local for each set (i.e., normalization statistics for T^τ are independent of D^τ). We refer to this approach as *transductive* batch normalization (TBN; see Figure 5.1).

Unsurprisingly, Nichol et al. (2018) found that using TBN provides a significant performance boost in all cases they tested, which is corroborated by our results in Table 5.1. In other words, TBN achieves desiderata 2, and, as we demonstrate in Section 5.5, desiderata 1 as well. However, it is transductive. Due to the ubiquity of MAML, many competetive meta-learning methods (e.g. Gordon et al., 2019) have adopted TBN. However, in the case of TBN, transductivity is rarely stated as an explicit assumption, and may often confound the comparison among methods (Nichol et al., 2018). Importantly, we argue that to ensure

[1]See for example (Finn, 2017) for a reference implementation.

comparisons in experimental papers are rigorous, meta-learning methods that are transductive must be labeled as such.

5.3.4 Instance-Based Normalization Schemes

An additional class of non-transductive NLs are *instance-based* NLs. Here, both at meta-train and meta-test time, moments are computed separately for each instance, and do not depend on other observations. From a modelling perspective, this corresponds to treating $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ as local at the *observation* level. As instance-based NLs do not depend on the context set size, they perform equally well across context-set sizes (desiderata 2). However, as we demonstrate in Section 5.5, the improvements in predictive performance are modest compared to more suitable NLs and they are worse than CBN in terms of training efficiency (thus not meeting desiderata 1). Below, we discuss three examples.

Layer Normalization (LN; Ba et al., 2016) LN (see Figure 5.1) has been shown to improve performance compared to CBN in recurrent neural networks, but does not offer the same gains for convolutional neural networks (Ba et al., 2016). The LN moments are computed as:

$$\boldsymbol{\mu}_{LN_b} = \frac{1}{HWC}\sum_{w=1}^{W}\sum_{h=1}^{H}\sum_{c=1}^{C}\boldsymbol{a}_{bwhc}, \tag{5.5}$$

$$\boldsymbol{\sigma}^2_{LN_b} = \frac{1}{HWC}\sum_{w=1}^{W}\sum_{h=1}^{H}\sum_{c=1}^{C}(\boldsymbol{a}_{bwhc} - \boldsymbol{\mu}_{LN_b})^2 \tag{5.6}$$

where $\boldsymbol{\mu}_{LN}, \boldsymbol{\sigma}^2_{LN} \in \mathbb{R}^B$. While non-transductive, Table 5.1 demonstrates that LN falls far short of TBN in terms of accuracy. Further, in Section 5.5 we demonstrate that LN lacks in training efficiency when compared to other NLs.

Instance Normalization (IN; Ulyanov et al., 2016) IN (see Figure 5.1) has been used in a wide variety of image generation applications. The IN moments are computed as:

$$\boldsymbol{\mu}_{IN_{bc}} = \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H}\boldsymbol{a}_{bwhc}, \tag{5.7}$$

$$\boldsymbol{\sigma}^2_{IN_{bc}} = \frac{1}{HW}\sum_{w=1}^{W}\sum_{h=1}^{H}(\boldsymbol{a}_{bwhc} - \boldsymbol{\mu}_{IN_{bc}})^2 \tag{5.8}$$

where $\boldsymbol{\mu}_{IN}, \boldsymbol{\sigma}^2_{IN} \in \mathbb{R}^{B\times C}$. Table 5.1 demonstrates that IN has superior predictive performance to that of LN, but falls considerably short of TBN. In Section 5.5 we show that IN lacks in training efficiency when compared to other NLs.

Group Normalization (GN; Wu and He, 2018) A key insight of Wu and He (2018) is that CBN performance suffers with small batch sizes. The goal of Group Normalization is thus to address the problem of normalization of small batch sizes, which, among other matters, is crucial for training large models in a data-parallel fashion. This is achieved by dividing the image channels into a number of groups G and subsequently computing the moments for each group. GN is equivalent to LN when there is only a single group ($G = 1$) and equivalent to IN when the number of groups is equal to the number of channels in the layer ($G = C$).

5.3.5 Other NLs

There exist additional NLs including Weight Normalization (Salimans and Kingma, 2016), Cosine Normalization (Luo et al., 2018), Filter Response Normalization (Singh and Krishnan, 2019), among many others.

Weight normalization reparameterizes weight vectors in a neural network to improve the conditioning for optimization. Weight normalization is non-transductive, but we don't consider this approach further in this work as we focus on NLs that modify activations as opposed to weights.

Filter Response Normalization (FRN) is another non-transductive NL that performs well for all batch sizes. However we did not include it in our evaluation as FRN also encompasses the activation function as an essential part of normalization making it difficult to be a drop in replacement for CBN in pre-trained networks as is the case for some of our experiments.

Cosine normalization replaces the dot-product calculation in neural networks with cosine similarity for improved performance. We did not consider this method further in our work as it is not a simple drop-in replacement for CBN in pre-existing networks such as the ResNet-18 we use in our experiments.

5.4 Task Normalization

In the previous section, we demonstrated that it is not immediately obvious how NLs should be designed for meta-learning applications. We now develop TaskNorm, the first NL that is specifically tailored towards this scenario. TaskNorm is motivated by the view of meta-learning as hierarchical probabilistic modelling, discussed in Section 2.2.3. Given this hierarchical view of the model parameters, the question that arises is, how should we treat the normalization statistics $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$? Figure 2.3 implies that the data associated with a task τ are i.i.d. *only when conditioning on both* $\boldsymbol{\theta}$ *and* ψ^{τ}. Thus, the normalization statistics $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ should be local at the task level, i.e., absorbed into ψ^{τ}. Further, the view that ψ^{τ} should be inferred conditioned on D^{τ} implies that the normalization statistics for the target set should be computed directly from the context set. Finally, our desire for a non-transductive scheme implies that any contribution from points in the target should not affect the normalization for other points in the target set, i.e., when computing $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ for a particular observation $\boldsymbol{x}^{\tau*} \in T^{\tau}$, the NL should only have access to D^{τ} and $\boldsymbol{x}^{\tau*}$.

5.4.1 Meta-Batch Normalization (MetaBN)

This perspective leads to our definition of MetaBN, which is a simple adaptation of CBN for the meta-learning setting. In MetaBN, the context set alone is used to compute the normalization statistics for *both* the context and target sets, both at meta-train and meta-test time (see Figure 5.1). To our knowledge, MetaBN has not been described in any publication, but concurrent to this work, it is used in the implementation of Meta-Dataset (Triantafillou et al., 2019).

MetaBN meets almost all of our desiderata, it (i) is non-transductive since the normalization of a test input does not depend on other test inputs in the target set, and (ii) as we demonstrate in Section 5.5, it improves training speed while maintaining accuracy levels of meta-learning models. However, as we demonstrate in Section 5.5, MetaBN performs less well for small context sets. This is because moment estimates will have high-variance when there is little data, and is similar to the difficulty of using BN with small-batch training (Wu and He, 2018). To address this issue, we introduce the following extension to MetaBN, which yields our proposed normalization scheme – TaskNorm.

5.4.2 TaskNorm

The key intuition behind TaskNorm is to normalize a task with the context set moments in combination with a set of non-transductive, secondary moments computed from the input being normalized. A blending factor α between the two sets of moments is learned during meta-training. The motivation for TaskNorm is as follows: when the D^{τ} is small (e.g. 1-shot or few-shot learning) the context set alone will lead to noisy and inaccurate estimates of the "true" task statistics. In such cases, a secondary set of moments may improve the estimate of the moments, leading to better training efficiency and predictive performance in the low data regime. Further, this provides information regarding $\boldsymbol{x}^{\tau*}$ at prediction time while maintaining non-transductivity. The pooled moments for TaskNorm are computed as:

$$\begin{aligned} \boldsymbol{\mu}_{TN} &= \alpha \boldsymbol{\mu}_{BN} + (1-\alpha)\boldsymbol{\mu}_{+}, && (5.9) \\ \boldsymbol{\sigma}^2_{TN} &= \alpha\left(\boldsymbol{\sigma}^2_{BN} + (\boldsymbol{\mu}_{BN} - \boldsymbol{\mu}_{TN})^2\right) + (1-\alpha)\left(\boldsymbol{\sigma}^2_{+} + (\boldsymbol{\mu}_{+} - \boldsymbol{\mu}_{TN})^2\right), && (5.10) \end{aligned}$$

where $\boldsymbol{\mu}_{TN}, \boldsymbol{\sigma}_{TN} \in \mathbb{R}^{B\times C}$, $\boldsymbol{\mu}_{+}$, $\boldsymbol{\sigma}^2_{+}$ are additional moments from a non-transductive NL such as LN or IN computed using activations from the example being normalized (see Figure 5.1), and $\boldsymbol{\mu}_{BN}$ and $\boldsymbol{\sigma}_{BN}$ are computed from D^{τ}. Equation (5.10) is the standard *pooled variance* when combining the variance of two Gaussian estimators. Importantly, we parameterize $\alpha = \textsc{sigmoid}(\textsc{scale}|D^{\tau}| + \textsc{offset})$, where the sigmoid function ensures that $0 \leq \alpha \leq 1$. α

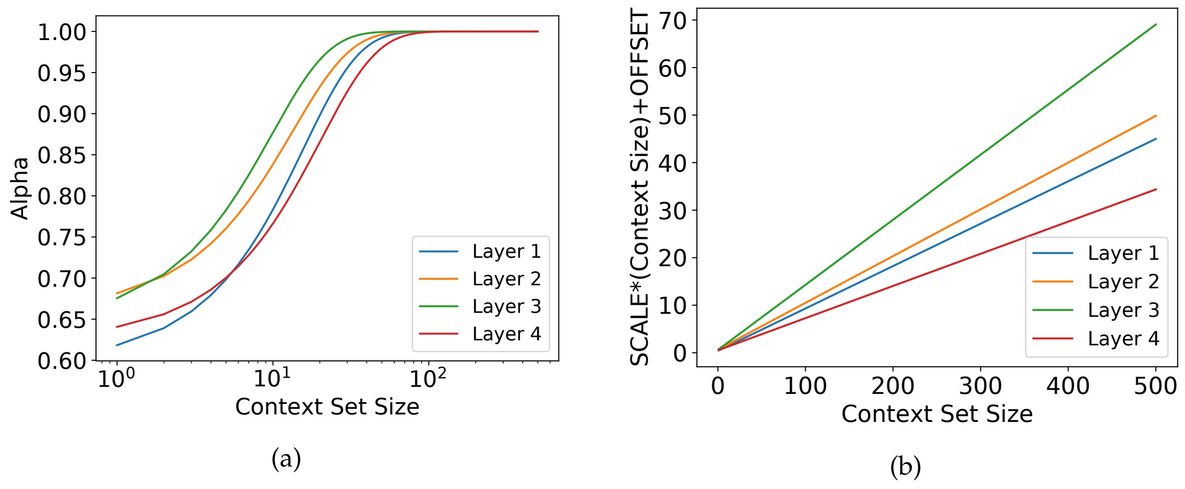


Fig. 5.2 Plots of: (a) α versus context set size, and (b) SCALE$|D^\tau|$ + OFFSET for the first NL in each of the four layers in the feature extractor for the TASKNORM-I model associated with Table 5.3.

and the scalars SCALE and OFFSET are learned during meta-training in a manner identical to every other trainable parameter in the system. This enables us to learn how much each set should contribute to the estimate of task statistics as a function of the context-set size $|D^\tau|$. Figure 5.2a depicts the value of α as a function of context set size $|D^\tau|$ for a representative set of trained TASKNORM layers. In general, when the context size is suitably large ($N_\tau > 25$), α is close to unity, i.e., normalization is carried out entirely with the context set in those layers. When the context size is smaller, there is a mix of the two sets of moments. Allowing each TASKNORM layer to separately adapt to the size of the context set (as opposed to learning a fixed α per layer) is crucial in the meta-learning setting, where we expect the size of D^τ to vary, and are often particularly interested in the "few-shot" regime. Figure 5.2b plots the line SCALE$|D^\tau|$ + OFFSET for same set of NLs as Figure 5.2a. The algorithm has learned that the SCALE parameter is non-zero and the OFFSET is almost zero in all cases indicating the importance of having α being a function of context size. In Section 5.5.4, we provide an ablation study demonstrating the importance of our proposed parameterization of α. If the context size is fixed, we do not use the full parameterization, but learn a single value for alpha directly. The computational cost of TASKNORM is marginally greater than CBN's. As a result, per-iteration time increases only slightly. However, as we demonstrate in Section 5.5, TASKNORM converges faster than CBN.

In related work, Nam and Kim (2018) define Batch-Instance Normalization (BIN) that combines the results of CBN and IN with a learned blending factor in order to attenuate unnecessary styles from images. However, BIN blends the output of the individual CBN and IN normalization operations as opposed to blending the moments and then using the blended moments to normalize as in TASKNORM.

Configuration	TBN	example	class	CBN	BRN	LN	IN	RN	MetaBN	TaskNorm-L	TaskNorm-I
Omniglot-5-1	**98.4±0.7**	21.6±1.3	21.6±1.3	20.1±0.0	20.0±0.0	83.0±1.3	87.4±1.2	92.6±0.9	91.8±0.9	94.0±0.8	94.4±0.8
Omniglot-5-5	**99.2±0.2**	22.0±0.5	23.2±0.5	20.0±0.0	20.0±0.0	91.0±0.8	93.9±0.5	98.2±0.2	98.1±0.3	98.0±0.3	98.6±0.2
Omniglot-20-1	**90.9±0.5**	3.7±0.2	3.7±0.2	5.0±0.0	5.0±0.0	78.1±0.7	80.4±0.7	89.0±0.6	89.6±0.5	89.6±0.5	**90.0±0.5**
Omniglot-20-5	**96.6±0.2**	5.5±0.2	14.5±0.3	5.0±0.0	5.0±0.0	92.3±0.2	92.9±0.2	**96.8±0.2**	96.4±0.2	96.4±0.2	96.3±0.2
miniImageNet-5-1	**45.5±1.8**	26.9±1.5	26.9±1.5	20.1±0.0	20.4±0.4	41.2±1.6	40.7±1.7	40.7±1.7	41.6±1.6	42.0±1.7	42.4±1.7
miniImageNet-5-5	**59.7±0.9**	30.3±0.7	27.2±0.6	20.2±0.2	20.7±0.5	52.8±0.9	54.3±0.9	57.6±0.9	**58.6±0.9**	58.1±0.9	**58.7±0.9**
Average Rank	1.25	-	-	8.42	8.58	6.58	5.75	4.00	3.67	3.75	3.00

Table 5.1 Accuracy results for different few-shot settings on Omniglot and *mini*ImageNet using the MAML algorithm. All figures are percentages and the ± sign indicates the 95% confidence interval. Bold indicates the highest scores. The numbers after the configuration name indicate the way and shots, respectively. The vertical lines enclose the transductive results. The *TBN*, *examples*, and *class* columns indicate accuracy when tested with all target examples at once, one example at a time, and one class at a time, respectively. All other NLs are non-transductive and yield the same result when tested by example or class.

Finally, we note that Reptile (Nichol et al., 2018) uses a non-transductive form of task normalization that involves normalizing examples from the target set one example at a time with the moments of the context set augmented with the single example. We refer to this approach as *reptile* normalization or RN. It is easy to show that RN is a special case of TASKNORM augmented with IN when $\alpha = |D^\tau|/(1 + |D^\tau|)$. In Section 5.5, we show that reptile normalization falls short of TASKNORM, supporting the intuition that learning the value of α is preferable to fixing a value.

5.5 Experiments

In this section, we evaluate TASKNORM along with a range of competitive normalization approaches.[2] The goal of the experiments is to evaluate the following hypotheses: (i) Meta-learning algorithms are sensitive to the choice of NL; (ii) TBN will, in general, outperform non–transductive NLs; and (iii) NLs that consider the meta-learning data assumptions (TASKNORM, METABN, RN) will outperform ones that do not (CBN, BRN, IN, LN, etc.). (iv) Transductive learners will perform poorly if the target set contains a different class balance that what was observed during meta-training or if they are required to make predictions one example at a time.

5.5.1 Small Scale Few-Shot Classification Experiments

We evaluate TASKNORM and a set of NLs using the first order MAML and Prototypical Networks algorithms on the Omniglot and miniImageNet datasets under various way (the number of classes used in each task) and shot (the number of context set examples used per

[2]Source code available at https://github.com/cambridge-mlg/cnaps.

Configuration	TBN	CBN	BRN	LN	IN	RN	MetaBN	TaskNorm-L	TaskNorm-I
Omniglot-5-1	98.4±0.2	**98.5±0.2**	**98.5±0.2**	**98.7±0.2**	93.7±0.4	98.0±0.2	98.4±0.2	**98.6±0.2**	98.4±0.2
Omniglot-5-5	**99.6±0.1**	**99.6±0.1**	**99.6±0.1**	**99.7±0.1**	98.8±0.1	**99.6±0.1**	**99.6±0.1**	**99.6±0.1**	**99.6±0.1**
Omniglot-20-1	94.5±0.2	94.5±0.2	94.6±0.2	**94.9±0.2**	83.5±0.3	94.1±0.2	94.5±0.2	**95.0±0.2**	93.4±0.2
Omniglot-20-5	**98.6±0.1**	**98.6±0.1**	**98.6±0.1**	**98.7±0.1**	96.3±0.1	**98.6±0.1**	**98.6±0.1**	**98.7±0.1**	**98.6±0.1**
miniImageNet-5-1	45.9±0.6	**47.8±0.6**	46.3±0.6	**47.5±0.6**	30.4±0.5	39.7±0.5	42.6±0.6	**47.5±0.6**	43.2±0.6
miniImageNet-5-5	65.5±0.5	**66.7±0.5**	64.7±0.5	**66.3±0.5**	48.8±0.5	63.1±0.5	64.6±0.5	65.3±0.5	63.9±0.5
Average Rank	4.58	3.25	4.33	2.75	9.00	6.67	5.25	3.08	6.08

Table 5.2 Accuracy results for different few-shot settings on Omniglot and *mini*ImageNet using the Prototypical Networks algorithm. All figures are percentages and the ± sign indicates the 95% confidence interval. Bold indicates the highest scores. The numbers after the configuration name indicate the way and shots, respectively. The vertical lines in the TBN column are to emphasize that this method is transductive.

class) configurations. This setting is smaller scale, and considers only fixed-sized context and target sets. Configuration and training details can be found in Section B.3.

Accuracy Table 5.1 and Table 5.2 show accuracy results for various normalization methods on the Omniglot and miniImageNet datasets using the first order MAML and the Prototypical Networks algorithms, respectively. We compute the average rank in an identical manner to (Triantafillou et al., 2020).

For MAML, TBN is clearly the best method in terms of classification accuracy. The best non-transductive approach is TaskNorm that uses IN augmentation (TaskNorm-I). The two methods using instance-based normalization (LN, IN) do significantly less well than methods designed with meta-learning desiderata in mind (i.e. TaskNorm, MetaBN, and RN). The methods using running averages at meta-test time (CBN, BRN) fare the worst. Figure 5.3 compares the performance of MAML on unseen tasks from miniImageNet when trained with TBN, IN, MetaBN, and TaskNorm, as a function of the number of shots per class in D^{τ}, and demonstrates that these trends are consistent across the low-shot range.

Note that when meta-testing occurs one example at a time (e.g. streaming data scenario) or one class at a time (unbalanced class distribution scenario), accuracy for TBN drops dramatically compared to the case where all the examples are tested at once. This is a drawback of the transductive approach. All of the other NLs in the table are non-transductive and do not suffer a decrease in accuracy when tested an example at a time or a class at a time.

Compared to MAML, the Prototypical Networks algorithm is much less sensitive to the NL used. Table 5.2 indicates that with the exception of IN, all of the normalization methods yield good performance. We suspect that this is due to the fact that ProtoNets employs a parameter-less nearest neighbor classifier and no gradient steps are taken at meta-test time, lessening the importance of normalization. In addition, this is likely due to the fact that the tasks in the small scale experiments are homogeneous (i.e. they have fixed shot and way with

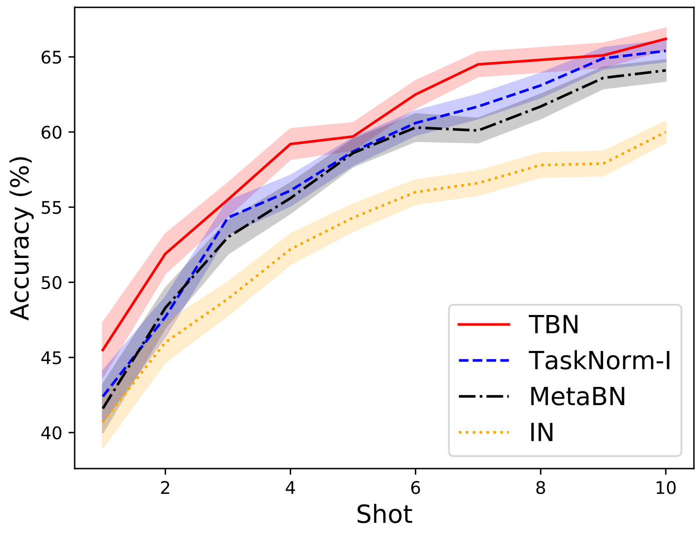


Fig. 5.3 Accuracy vs shot for MAML on 5-way *mini*ImageNet classification.

little variation in the image content). In the large scale experiments with heterogeneous tasks that are described in Section 5.5.2, we see that ProtoNets benefits to a much greater degree from TaskNorm. The top performer for ProtoNets in the small scale experiments is LN which narrowly edges out TaskNorm-L and CBN. Interestingly, TBN is not on top and TaskNorm-I lags as IN is the least effective method.

Training Speed Figure 5.4a plots validation accuracy versus training iteration for the first order MAML algorithm training on Omniglot 5-way-5-shot. TBN is the most efficient in terms of training convergence. The best non-transductive method is again TaskNorm-I, which is only marginally worse than TBN and just slightly better than TaskNorm-L. Importantly, TaskNorm-I is superior to either of MetaBN and IN alone in terms of training efficiency. Figure 5.4b depicts the training curves for the ProtoNets algorithm. With the exception of IN which converges to a lower validation accuracy, all NLs converge at the the same speed.

For the MAML algorithm, the experimental results support our hypotheses. Performance varies significantly across NLs. TBN outperformed all methods in terms of classification accuracy and training efficiency, and TaskNorm is the best non-transductive approach. Finally, The meta-learning specific methods outperformed the more general ones. The picture for Prototypical Networks is rather different. There is little variability across NLs, TBN lagging the most consistent method LN, and the NLs that considered meta-learning needs were not necessarily superior to those that did not.

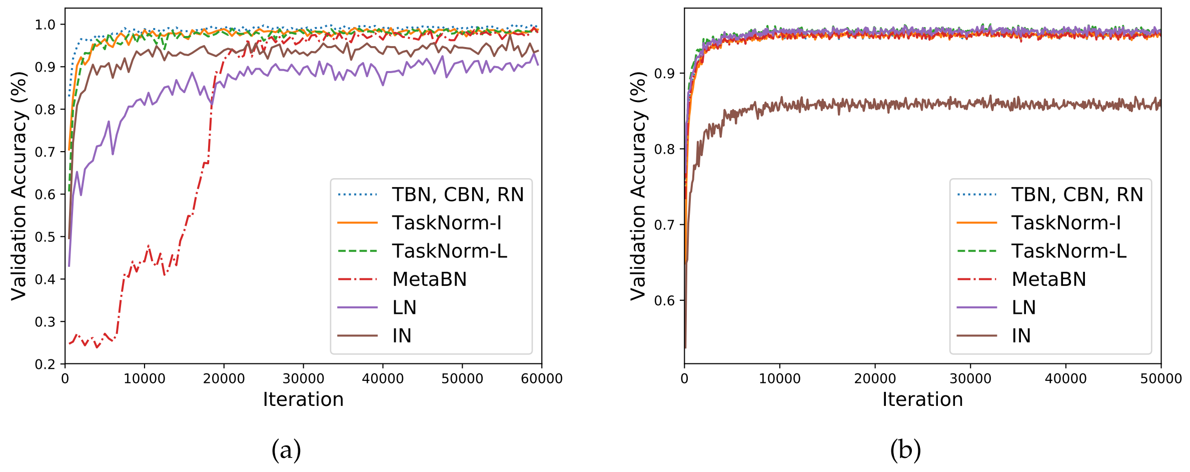


Fig. 5.4 Plots of validation accuracy versus training iteration using: (a) the MAML algorithm for Omniglot 5-way, 5-shot corresponding to the results in Table 5.1, and (b) the ProtoNets algorithm for Omniglot 20-way, 1-shot corresponding to the results in Table 5.2.

5.5.2 Large Scale Few-Shot Classification Experiments

Next, we evaluate NLs on a demanding few-shot classification challenge called Meta-Dataset, composed of thirteen (eight train, five test) image classification datasets (Triantafillou et al., 2020). Experiments are carried out with CNAPS, which achieves state-of-the-art performance on Meta-Dataset (Requeima et al., 2019b) and Prototypical Networks. The challenge constructs few-shot learning tasks by drawing from the following distribution. First, one of the datasets is sampled uniformly; second, the "way" and "shot" are sampled randomly according to a fixed procedure; third, the classes and context / target instances are sampled. As a result, the context size D^τ will vary in the range between 5 and 500 for each task. In the meta-test phase, the identity of the original dataset is not revealed and tasks must be treated independently (i.e. no information can be transferred between them). The meta-training set comprises a disjoint and dissimilar set of classes from those used for meta-test. Details provided in Section B.3 and Triantafillou et al. (2020).

Accuracy The classification accuracy results for CNAPS and Prototypical Networks on Meta-Dataset are shown in Table 5.3 and Table 5.4, respectively. In the case of Prototypical Networks, all the the NLs specifically designed for meta-learning scenarios outperform TBN in terms of classification accuracy based on their average rank over all the datasets. For CNAPS, both RN and TASKNORM-I meet or exceed the rank of TBN. This may be as $|D^\tau|$ (i) is quite large in Meta-Dataset, and (ii) may be imbalanced with respect to classes, making prediction harder with transductive NLs. TASKNORM-I comes out as the clear winner ranking first in 11 and 10 of the 13 datasets using CNAPS and Prototypical Networks, respectively. This supports the hypothesis that augmenting the BN moments with a second, instance based set of moments

Dataset	TBN	Baseline	CBN	BRN	LN	IN	RN	MetaBN	TaskNorm-r	TaskNorm-L	TaskNorm-I
ILSVRC	**50.2±1.0**	**51.3±1.0**	24.8±0.7	19.2±0.7	45.5±1.1	46.7±1.0	49.7±1.1	**51.3±1.1**	49.3±1.0	**51.2±1.1**	**50.6±1.1**
Omniglot	**91.4±0.5**	88.0±0.7	47.9±1.4	60.0±1.6	87.4±0.8	79.7±1.0	**91.0±0.6**	**90.9±0.6**	87.8±0.7	90.6±0.6	**90.7±0.6**
Aircraft	81.6±0.6	76.8±0.8	29.5±0.9	56.3±0.8	76.5±0.8	74.7±0.7	82.4±0.6	**83.9±0.6**	81.1±0.7	81.9±0.6	**83.8±0.6**
Birds	**74.5±0.8**	71.4±0.9	42.1±1.0	32.6±0.8	67.3±0.9	64.9±1.0	72.4±0.8	73.2±0.9	72.8±0.9	72.4±0.8	**74.6±0.8**
Textures	59.7±0.7	**62.5±0.7**	37.5±0.7	50.5±0.6	60.1±0.6	59.7±0.7	58.6±0.7	58.9±0.8	**63.2±0.8**	57.2±0.7	62.1±0.7
Quick Draw	70.8±0.8	71.9±0.8	44.5±1.0	56.7±1.0	71.6±0.8	68.2±0.9	**74.3±0.8**	**74.1±0.7**	71.6±0.8	**74.3±0.8**	**74.8±0.7**
Fungi	46.0±1.0	46.0±1.1	21.1±0.8	26.1±0.9	39.6±1.0	37.8±1.0	**49.0±1.0**	**47.9±1.0**	42.0±1.1	47.1±1.1	**48.7±1.0**
VGG Flower	86.6±0.5	**89.2±0.5**	79.0±0.7	75.7±0.7	84.4±0.6	82.6±0.6	86.9±0.6	85.9±0.6	87.7±0.6	87.3±0.5	**89.6±0.6**
Traffic Signs	**66.6±0.9**	60.1±0.9	38.3±0.9	38.8±1.2	57.3±0.8	62.5±0.8	**66.6±0.8**	58.9±0.9	62.7±0.8	62.0±0.8	**67.0±0.7**
MSCOCO	41.3±1.0	**42.0±1.0**	14.2±0.7	19.1±0.8	32.9±1.0	40.8±1.0	**42.1±1.0**	41.6±1.1	40.1±1.0	41.6±1.0	**43.4±1.0**
MNIST	92.1±0.4	88.6±0.5	65.9±0.8	82.5±0.6	86.8±0.5	89.8±0.5	91.3±0.4	92.1±0.4	**93.2±0.3**	90.5±0.4	**92.3±0.4**
CIFAR10	**70.1±0.8**	60.0±0.8	26.1±0.7	29.1±0.6	55.8±0.8	65.9±0.8	**69.7±0.7**	**69.6±0.8**	66.9±0.8	**70.3±0.8**	**69.3±0.8**
CIFAR100	55.6±1.0	48.1±1.0	16.7±0.8	16.7±0.7	37.9±1.0	52.9±1.0	55.0±1.0	54.2±1.1	53.0±1.1	**59.5±1.0**	54.6±1.1
Average Rank	3.92	5.58	10.69	10.31	7.96	7.54	3.77	4.04	5.38	4.42	2.38

Table 5.3 Few-shot classification results on META-DATASET using the CNAPS algorithm. Meta-training performed on datasets above the dashed line. Datasets below the dashed line are entirely held out. All figures are percentages and the ± sign indicates the 95% confidence interval over tasks. Bold indicates the highest scores. Vertical lines in TBN column emphasize that this method is transductive. Numbers in the BASELINE column are from (Requeima et al., 2019b).

and learning the blending factor α as a function of context set size is superior to fixing α to a constant value (as is the case with RN). With both methods, the instance based NLs fall short of the meta-learning specific ones. However, in the case of CNAPS, they outperform the running average based methods (CBN, BRN), which perform poorly. In the case of Prototypical Networks, BRN outperforms the instance based methods, and IN fairs the worst of all. In general, Prototypical Networks is less sensitive to the NL used when compared to CNAPS. The BASELINE column in Table 5.3 is taken from (Requeima et al., 2019b), where the method reported state-of-the-art results on Meta-Dataset. The BASELINE algorithm uses the running moments learned during pre-training of its feature extractor for normalization. Using meta-learning specific NLs (in particular TASKNORM) achieves significantly improved accuracy compared to BASELINE.

As an ablation, we have also added an additional variant of TASKNORM that blends the batch moments from the context set with the running moments accumulated during meta-training that we call TASKNORM-r. TASKNORM-r makes use of the global running moments to augment the local context statistic and did not perform as well as the TASKNORM variants that employed local moments (i.e. TASKNORM-I and TASKNORM-L).

Training Speed Figure 5.5a plots training loss versus training iteration for the models in Table 5.3 that use the CNAPS algorithm. The fastest training convergence is achieved by TASKNORM-I. The instance based methods (IN, LN) are the slowest to converge. Note that TASKNORM converges within 60k iterations while BASELINE takes 110k iterations and IN takes

Dataset	TBN	CBN	BRN	LN	IN	RN	MetaBN	TaskNorm-r	TaskNorm-L	TaskNorm-I
ILSVRC	**44.7±1.0**	43.6±1.0	43.0±1.0	33.9±0.9	32.5±0.9	**45.1±1.0**	**44.2±1.0**	42.7±1.0	**45.1±1.1**	**44.9±1.0**
Omniglot	**90.7±0.6**	77.5±1.1	89.1±0.7	**90.8±0.6**	83.4±0.8	**90.8±0.6**	**90.4±0.6**	88.6±0.7	**90.2±0.6**	**90.6±0.6**
Aircraft	83.3±0.6	77.0±0.7	**84.4±0.5**	73.9±0.7	75.0±0.6	80.9±0.6	82.3±0.6	79.6±0.6	81.2±0.6	**84.7±0.5**
Birds	69.6±0.9	67.5±0.9	69.0±0.9	54.1±1.0	50.2±1.0	68.6±0.9	68.6±0.8	64.2±0.9	68.8±0.9	**71.0±0.9**
Textures	61.2±0.7	57.7±0.7	58.0±0.7	55.8±0.7	45.3±0.7	64.1±0.7	60.5±0.7	60.8±0.7	63.4±0.8	**65.9±0.7**
Quick Draw	75.0±0.8	62.1±1.0	74.3±0.8	72.5±0.8	70.8±0.8	75.4±0.7	74.2±0.7	73.2±0.8	75.4±0.7	**77.5±0.7**
Fungi	46.4±1.0	43.6±1.0	46.5±1.0	33.2±1.1	29.8±1.0	46.7±1.0	46.5±1.0	42.3±1.1	46.5±1.0	**49.6±1.1**
VGG Flower	83.1±0.6	82.3±0.6	84.5±0.6	78.3±0.8	69.4±0.8	84.4±0.7	**86.0±0.6**	81.1±0.7	82.9±0.7	83.2±0.6
Traffic Signs	64.0±0.8	59.5±0.8	65.7±0.8	**69.1±0.7**	60.7±0.8	66.0±0.8	63.2±0.8	64.9±0.8	67.0±0.7	65.8±0.7
MSCOCO	38.2±1.0	36.6±1.0	**38.4±1.0**	30.1±0.9	27.7±0.9	37.3±1.0	**38.6±1.1**	35.4±1.0	**39.2±1.0**	**38.5±1.0**
MNIST	93.4±0.4	86.5±0.6	91.9±0.4	**94.0±0.4**	87.4±0.5	**93.9±0.4**	**93.9±0.4**	92.5±0.4	91.9±0.4	93.3±0.4
CIFAR10	64.7±0.8	57.3±0.8	60.1±0.8	51.5±0.8	50.5±0.8	62.3±0.8	63.0±0.8	61.4±0.8	**66.9±0.8**	**67.6±0.8**
CIFAR100	48.0±1.1	43.1±1.0	43.9±1.0	34.0±0.9	32.1±1.0	47.2±1.1	47.0±1.0	45.2±1.0	**51.3±1.1**	**50.0±1.0**
Average Rank	4.04	8.19	5.31	7.46	9.58	3.65	3.96	6.73	3.58	2.50

Table 5.4 Few-shot classification results on Meta-Dataset using the Prototypical Networks algorithm. Meta-training performed on datasets above the dashed line. Datasets below the dashed line are entirely held out. All figures are percentages and the ± sign indicates the 95% confidence interval over tasks. Bold indicates the highest scores. Vertical lines in TBN column emphasize that this method is transductive.

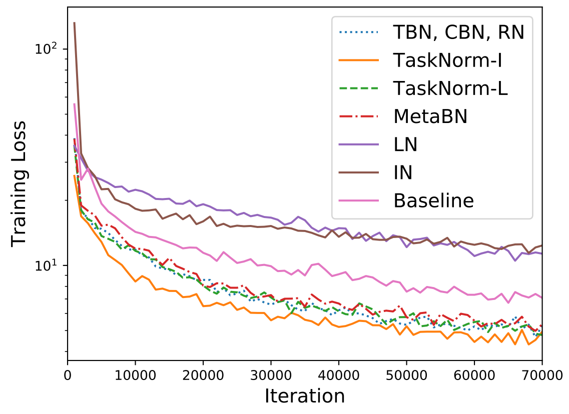
200k. Figure 5.5b show the training curves for the ProtoNets algorithm. The convergence speed trends are very similar to the CNAPs case, with TaskNorm-I being the fastest.

Our results demonstrate that TaskNorm is the best approach for normalizing tasks on the large scale Meta-Dataset benchmark in terms of classification accuracy and training efficiency. Here, we see high sensitivity of performance across NLs. Interestingly, in this setting TaskNorm-I outperformed TBN in classification accuracy, as did both RN and MetaBN. This refutes the hypothesis that TBN will always outperform other methods due to its transductive property, and implies that designing NL methods specifically for meta-learning has significant value. In general, the meta-learning specific methods outperformed more general NLs, supporting our third hypothesis. We suspect the reason that TaskNorm outperforms other methods is due to its ability to adaptively leverage information from both D^τ and $\boldsymbol{x}^{\tau *}$ when computing moments, based on the size of D^τ.

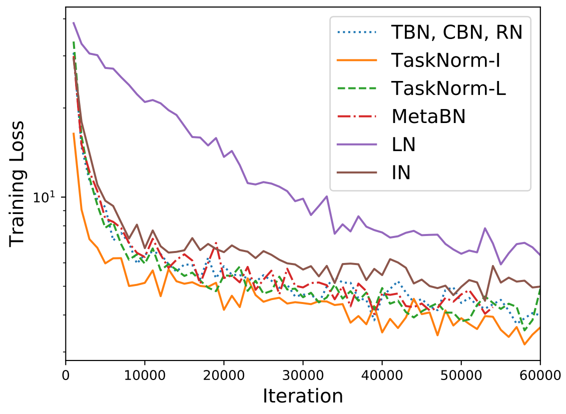
5.5.3 Transduction Tests

A non-transductive meta-learning system makes predictions for a single test set label conditioned only on a single input and the context set. A transductive meta-learning system also conditions on additional samples from the test set.

Table 5.5 demonstrates failure modes for transductive learning. In addition to reporting the classification accuracy results when the target set is evaluated all at once (first column of results for each NL), we report the classification accuracy when meta-testing is performed one target-set *example* at a time (second column of results for each NL), and one target-set *class* at a



(a)



(b)

Fig. 5.5 Training Loss versus iteration on META-DATASET corresponding to the results using: (a) the CNAPS algorithm in Table 5.3, and (b) the ProtoNets algorithm in Table 5.4. Note that TBN, CBN, and RN all share the same meta-training step.

time (third column of results for each NL). Table 5.5 demonstrates that classification accuracy drops dramatically for TBN when testing is performed one example or one class at a time.

Importantly, in the case of TASKNORM-I (or any non-transductive NL – i.e. all of NLs evaluated in this work apart from TBN), the evaluation results are identical whether they are meta-tested on the entire target set at once, one example at a time, or one class at a time. This shows that transductive learning is sensitive to the distribution over the target set used during meta-training, demonstrating that transductive learning is less generally applicable than non-transductive learning. In particular, transductive learners may fail to make good predictions if target sets contains a different class balance than what was observed during meta-training, or if they are required to make predictions for one example at a time (e.g. in streaming applications).

5.5.4 Ablation Study: Choosing the best parameterization for α

There are a number of possibilities for the parameterization of the TASKNORM blending parameter α. We consider four different configurations for each NL:

1. α is learned separately for each channel (i.e. channel specific) as an independent parameter.
2. α is learned shared across all channels as an independent parameter.
3. α is learned separately for each channel (i.e. channel specific) as a function of context set size (i.e. $\alpha = \text{SIGMOID}(\text{SCALE}|D^\tau| + \text{OFFSET})$).

	TBN			TaskNorm-I		
Dataset	All	Example	Class	All	Example	Class
ILSVRC	50.2±1.0	9.5±0.3	11.8±0.4	50.4±1.1	50.4±1.1	50.4±1.1
Omniglot	91.4±0.5	7.5±0.4	9.6±0.4	91.3±0.6	91.3±0.6	91.3±0.6
Aircraft	81.6±0.6	11.8±0.4	14.4±0.4	83.8±0.6	83.8±0.6	83.8±0.6
Birds	74.5±0.8	7.6±0.4	8.4±0.4	74.4±0.9	74.4±0.9	74.4±0.9
Textures	59.7±0.7	17.0±0.2	18.1±0.4	61.1±0.7	61.1±0.7	61.1±0.7
Quick Draw	70.8±0.8	5.6±0.4	8.8±0.4	74.7±0.7	74.7±0.7	74.7±0.7
Fungi	46.0±1.0	5.0±0.3	6.5±0.4	50.6±1.1	50.6±1.1	50.6±1.1
VGG Flower	86.6±0.5	11.2±0.4	12.6±0.4	87.8±0.5	87.8±0.5	87.8±0.5
Traffic Signs	66.6±0.9	6.0±0.3	8.1±0.4	64.8±0.8	64.8±0.8	64.8±0.8
MSCOCO	41.3±1.0	6.1±0.3	7.9±0.4	42.2±1.0	42.2±1.0	42.2±1.0
MNIST	92.1±0.4	14.4±0.3	19.3±0.4	91.3±0.4	91.3±0.4	91.3±0.4
CIFAR10	70.1±0.8	14.4±0.3	16.4±0.4	70.0±0.8	70.0±0.8	70.0±0.8
CIFAR100	55.6±1.0	5.6±0.3	7.7±0.4	54.6±1.0	54.6±1.0	54.6±1.0

Table 5.5 Few-shot classification results for TBN and TaskNorm-I on Meta-Dataset using the CNAPs algorithm. For each NL, the first column of results "All" reports accuracy when meta-testing is performed on the entire target set at once. The second column of results "Example" reports accuracy when meta-testing is performed one example at a time. The third column of results "Class" reports accuracy when meta-testing is performed one class at a time. All figures are percentages and the ± sign indicates the 95% confidence interval over tasks. Meta-training is performed on datasets above the dashed line, while datasets below the dashed line are entirely held out.

4. α is learned shared across all channels as a function of context set size (i.e. $\alpha = \text{SIGMOID}(\text{SCALE}|D^\tau| + \text{OFFSET})$).

Accuracy Table 5.6 and Table 5.7 show classification accuracy for the various parameterizations for MAML and the CNAPs algorithms, respectively using the TaskNorm-I NL.

When using the MAML algorithm, there are only two options to evaluate as the context size is fixed for each configuration of dataset, shot, and way and thus we need only evaluate the independent options (1 and 2 above). Table 5.6 indicates that the classification accuracy for the channel specific and shared parameterizations are nearly identical, but the shared parameterization is better in the Omniglot-5-1 benchmark and hence has the best ranking overall.

When using the CNAPs algorithm on the Meta-Dataset benchmark, the best parameterization option in terms of classification accuracy is α shared across channels as a function of context size. One justification for having α be a function of context size can be seen in Figure 5.2b. Here we plot the line $\text{SCALE}|D^\tau| + \text{OFFSET}$ on a linear scale for a representative set of NLs in the ResNet-18 used in the CNAPs algorithm. The algorithm has learned that the SCALE parameter is non-zero and the OFFSET is almost zero in all cases. If a constant α

	Independent	
Configuration	Channel Specific	Shared
Omniglot-5-1	90.7±1.0	**94.4±0.8**
Omniglot-5-5	98.3±0.2	**98.6±0.2**
Omniglot-20-1	**90.6±0.5**	**90.0±0.5**
Omniglot-20-5	**96.4±0.2**	**96.3±0.2**
miniImageNet-5-1	**42.6±1.8**	**42.4±1.7**
miniImageNet-5-5	**58.8±0.9**	**58.7±0.9**
Average Rank	1.67	1.33

Table 5.6 Few-shot classification results for two α parameterizations on Omniglot and *mini*ImageNet using the MAML algorithm. All figures are percentages and the ± sign indicates the 95% confidence interval over tasks. Bold text indicates the highest scores.

would lead to better accuracy, we would see the opposite (i.e the SCALE parameter would be at or near zero and the OFFSET parameter being some non-zero value). From Table 5.7 we can also see that accuracy is better when the parameterization is a shared α opposed to having a channel-specific α.

Training Speed Figure 5.6a and Figure 5.6b show the learning curves for the various parameterization options using the MAML and the CNAPs algorithms, respectively with a TaskNorm-I NL.

For the MAML algorithm the training efficiency of the shared and channel specific parameterizations are almost identical. For the CNAPs algorithm, Figure 5.6b indicates the training efficiency of the independent parameterization is considerably worse than the functional one. The two functional representations for the CNAPs algorithm have almost identical training curves. Based on Figure 5.6a and Figure 5.6b, we conclude that the training speed of the functional parameterization is superior to that of the independent parameterization and that there is little or no difference in the training speeds between the functional, shared parameterization and the functional, channel specific parameterization.

In summary, the best parameterization for α when it is learned shared across channels as a function of context set size (option 4, above). We use this parameterization in all of the CNAPs experiments in the main paper. For the MAML experiments, the functional parameterization is meaningless given that all the test configurations have a fixed context size. In that case, we used the independent, shared across channels parameterization for α for the experiments in the main paper.

	Independent		Functional	
Dataset	Channel Specific	Shared	Channel Specific	Shared
ILSVRC	45.3±1.0	**49.6±1.1**	**49.8±1.1**	**50.6±1.1**
Omniglot	**90.8±0.6**	**90.9±0.6**	**90.1±0.6**	**90.7±0.6**
Aircraft	82.3±0.7	**84.6±0.6**	**84.4±0.6**	**83.8±0.6**
Birds	70.1±0.9	73.2±0.9	73.1±0.9	**74.6±0.8**
Textures	54.8±0.7	58.5±0.7	61.0±0.8	**62.1±0.7**
Quick Draw	73.0±0.8	**73.9±0.7**	**74.2±0.7**	**74.8±0.7**
Fungi	43.8±1.0	**47.6±1.0**	**48.0±1.0**	**48.7±1.0**
VGG Flower	85.9±0.6	86.3±0.5	86.5±0.7	**89.6±0.6**
Traffic Signs	62.6±0.8	62.6±0.8	60.1±0.8	**67.0±0.7**
MSCOCO	38.3±1.1	40.9±1.0	40.2±1.0	**43.4±1.0**
MNIST	**92.6±0.4**	91.7±0.4	91.1±0.4	**92.3±0.4**
CIFAR10	65.7±0.9	67.7±0.8	67.3±0.9	**69.3±0.8**
CIFAR100	48.1±1.2	52.1±1.1	**53.3±1.0**	**54.6±1.1**
Average Rank	3.5	2.5	2.5	1.5

Table 5.7 Few-shot classification results for various α parameterizations on META-DATASET using the CNAPS algorithm. All figures are percentages and the ± sign indicates the 95% confidence interval over tasks. Bold text indicates the highest scores. Meta-training performed on datasets above the dashed line, while datasets below the dashed line are entirely held out.

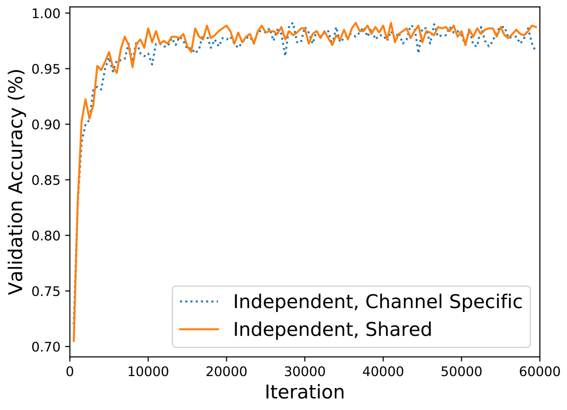
5.5.5 Evolution of α as Training Progresses

Figure 5.7a and Figure 5.7b show the how the SCALE and OFFSET parameters, respectively, vary as training progresses. Both SCALE and OFFSET increase monotonically with training iteration. As SCALE and OFFSET increase, α will also increase, which informs us that as training progresses, there is less reliance on the the auxiliary moment (in this case IN) and more weight is put on the standard batch moment (BN). It also tells us that the auxiliary moment plays a more significant role early on in training when SCALE and OFFSET are smaller.

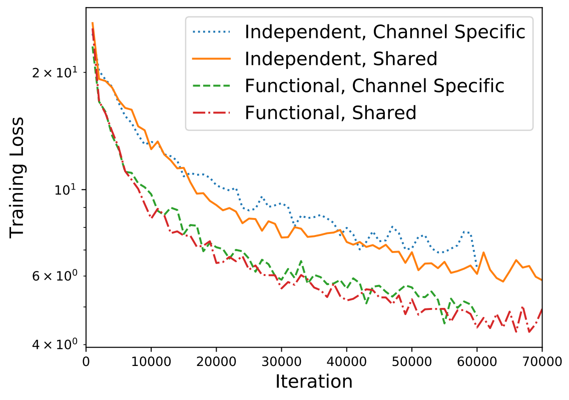
Similarly, Figure 5.8a and Figure 5.8b depict plots of α as a function of training iteration and context set size for two of the NLs in the feature extractor. It is clear from these plots that the weight placed on the standard batch moment (BN) increases and the weight placed on the auxiliary moment (IN) decreases as training progresses.

5.6 Summary

In this chapter, we have identified and specified several issues and challenges with NLs for the meta-learning setting. We have introduced a novel variant of batch normalization – that we call TASKNORM – which is geared towards the meta-learning setting. Our experiments demonstrate that TASKNORM achieves performance gains in terms of both classification accuracy and training speed, sometimes exceeding transductive batch normalization. We

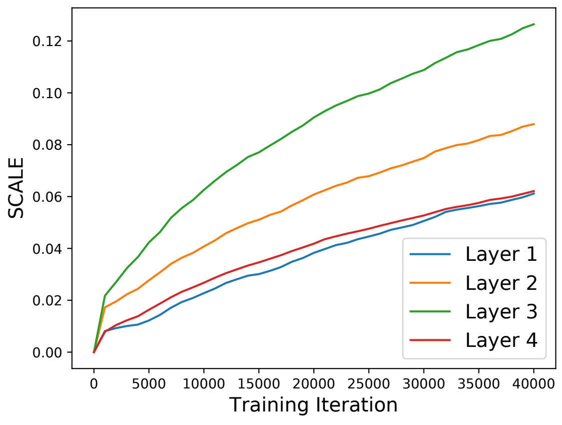


(a)

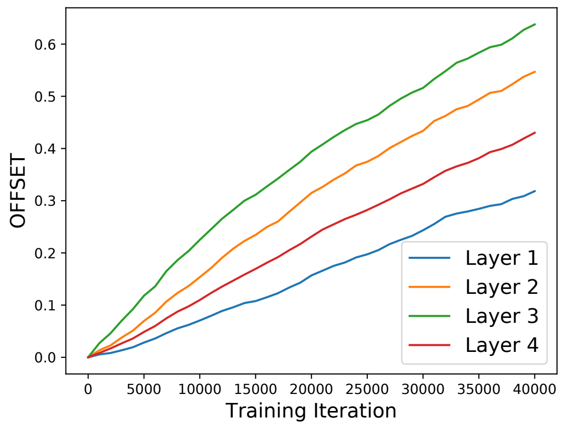


(b)

Fig. 5.6 (a) Plots of validation accuracy versus training iteration corresponding to the parameterization experiments using the MAML algorithm in Table 5.6. (b) Plot of training loss versus iteration corresponding to the parameterization experiments using the CNAPS algorithm in Table 5.7.



(a)



(b)

Fig. 5.7 (a) Plot of the learned SCALE parameter versus training iteration for the first NL in each of the four layers in the feature extractor for the TASKNORM-I model. (b) Plot of the learned OFFSET parameter versus training iteration for the first NL in each of the four layers in the feature extractor for the TASKNORM-I model.

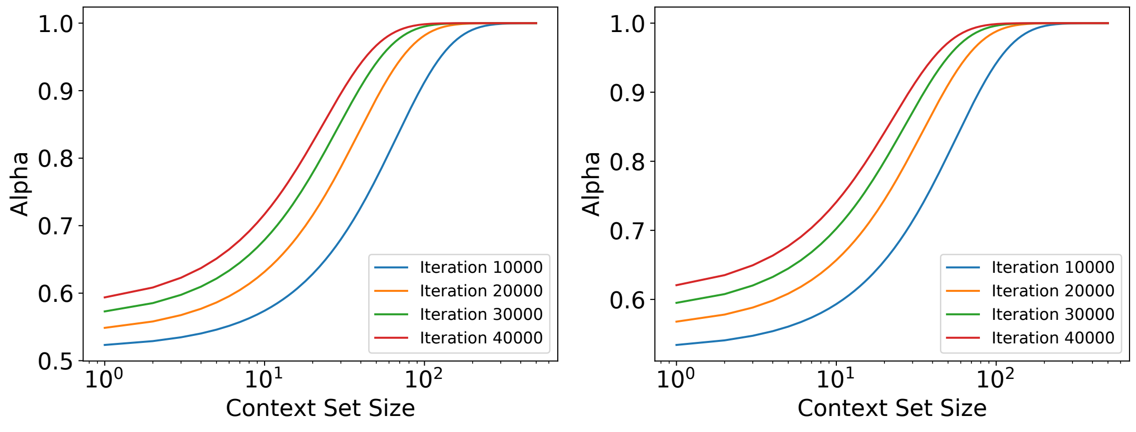


Fig. 5.8 Plots of α as a function of training iteration and context set size for the TASKNORM-I model. (a) Is for the first NL in the first layer of the feature extractor. (b) Is for the first NL in the last layer of the feature extractor.

recommend that future work in the few-shot / meta-learning community adopt TASKNORM, and if not, declare the form of normalization used and implications thereof, especially where transductive methods are applied.

Chapter 6

Conclusions and Discussion

6.1 Summary

The goal of this work has been to further the theory, design, development, and analysis of data and computation efficient machine learning systems. In particular, the work focused on few-shot image classification systems. Our resulting system CNAPS:

- during meta-training is exposed to multiple tasks such that it meta-learns to solve new tasks from only a few examples;
- at meta-test time can effectively and efficiently solve new, unseen tasks from a broad range of data distributions, in both the low and high data regimes, without the need for retraining;
- is fast at meta-test-time as learning a new task learning requires only a forward pass of the task context set through the meta-trained network and as such it is amenable to execution on a mobile device;
- has demonstrated state of the art results on META-DATASET (Triantafillou et al., 2020), arguably the most challenging few-shot learning benchmark to date.

6.1.1 Primary Contributions

The following is a list of the primary contributions of this work:

- We introduced ML-PIP, a probabilistic framework for meta-learning that unifies a broad class of recently proposed meta-learning methods, and suggests alternative approaches.
- We developed VERSA, an instance of ML-PIP employing a fast, flexible and versatile amortization network that takes few-shot learning datasets as inputs, with arbitrary numbers of training examples, and outputs a distribution over task-specific parameters

in a single forward pass of the network, avoiding the need to compute gradients at meta-test time.

- We demonstrated high quality one-shot reconstruction of unseen views of a 3D model by extending VERSA to regression tasks.
- We introduced CNAPS that extends CNPs to the multi-task classification setting. CNAPS adds a second amortized network to VERSA in order to adapt key parameters in the feature extractor to the current task.
- We devised a rich adaptation neural network with a novel auto-regressive structure that provides information to deeper layers in the feature extractor concerning how shallower layers have adapted to the task.
- We demonstrated that CNAPS achieved state-of-the-art results (at the time) on the challenging META-DATASET benchmark indicating high-quality transfer-learning.
- We showed that trained CNAPS models are immediately deployable to continual learning and active learning where they can outperform existing approaches that do not leverage transfer learning.
- We evaluated a range of approaches to batch normalization for meta-learning scenarios, and developed a novel approach called TASKNORM.
- We demonstrated that TASKNORM achieves performance gains in terms of both classification accuracy and training speed, sometimes exceeding transductive batch normalization in large scale experiments.

6.2 Discussion

In this section, we first discuss building a state-of-the-art, few-shot, multi-task classification system given the insights that I accumulated during my research. Following that, we offer some thoughts on future extensions to this work.

6.2.1 Building a State-of-the-Art Few-shot, Multi-task Image Classifier

During the course of this thesis, we have endeavoured to design and implement better and better few-shot, multi-task image classification systems. In this section, I will distill the knowledge and insights that I have gained during this process and express how I believe a state-of-the-art system should be built today.

Feature Extraction It is well known that a good feature embedding is key to accurate image classification (Tian et al., 2020). Much of the recent progress on few-shot learning benchmarks such as *mini*ImageNet has been due to more powerful feature extractors. Image feature extraction techniques have evolved rapidly in recent years from using feature descriptors such as histogram of oriented gradients (HOG) (Dalal and Triggs, 2005) and scale-invariant feature transform (SIFT) (Lowe, 2004), to basic convolutional neural networks (LeCun et al., 1999), residual networks (He et al., 2016), and more recently, inverted residual and linear bottleneck layers (Sandler et al., 2018) used in MobileNetV2 and EfficientNet (Tan and Le, 2019). EfficientNets are scalable in terms of network depth, network width, and input image resolution and can trade off classification accuracy with computational resources by changing the depth, width, or resolution. Many of the few-shot learning benchmarks use small image sizes (e.g. *mini*ImageNet images are 84×84 pixels). Using larger image sizes can yield more than a 10 percentage point increase in accuracy (Sandler et al., 2018) on the ImageNet (Krizhevsky et al., 2012) benchmark. Almost all of these high performance networks are available in a pre-trained, ready to use form.

A key requirement for meta-learning is the ability to adapt the systems to new unseen tasks. While feature extractors pre-trained on ImageNet are powerful, the features extracted on images that differ significantly from those in the ImageNet dataset (e.g. MNIST handwritten digits (LeCun et al., 2010)) will be non-optimal. As demonstrated in Chapter 4, FiLM (Perez et al., 2018) layers are a parameter efficient and expressive method to modulate the feature activations in a pre-trained feature extractor that can effectively adapt the pre-trained network adaptations to novel task data.

My recommendation for feature extraction at this time would be to use at least 224×224 pixel images with the largest size pre-trained EfficientNet variant supplemented with FiLM layer modulation that fits the computational resources available.

Final Layer Classifier In any few-shot, multi-task classification, the final classifier layer must adapt in some manner to the task at hand. If nothing else, the final classifier layer must have an output of size equal to the way (or number of classes) of the current task. There are many options for the final layer classifier, however the most commonly used are a simple linear classifier or a metric type classifier such as prototypical networks (Snell et al., 2017). Almost all of the work in this thesis focused on the use of VERSA-style final layer linear classifier with the weights generated class-by-class using a network that was trained across many tasks in an amortized fashion. However, Bateni et al. (2019) show that you can improve the classification accuracy of CNAPS by replacing the final VERSA classification layer with a prototypical networks classifier that uses the Euclidean distance metric. The authors also show that if you use a prototypical networks classifier with the Mahalanobis distance metric you can improve on the CNAPS accuracy performance by an even larger margin. I have independently verified these results. These results are interesting in that VERSA and prototypical networks are both special cases of ML-PIP (as was shown in Section 3.5) and both VERSA and prototypical networks when using the Euclidean distance metric can be expressed in the form of a linear classifier. However, VERSA requires a non-trivial multi-layer perceptron to generate the linear classifier weights while prototypical networks does not require a network at all (see Section 2.3.3). We postulate that prototypical networks using the Euclidean distance metric edges out the VERSA classifier in terms of classification accuracy because the VERSA amortized weight generation network is over-fitting, while there is no such network to overfit with prototypical networks. Computing the Mahalanobis distance is considerably more expensive compared to the Euclidean distance due to the requirement to invert the potentially large task-specific covariance matrix, but in my view that extra computation is worth the considerable gain in classification accuracy. Thus, my recommendation is to use prototypical networks with the Mahalanobis distance metric as the final layer classifier in a multi-task, few-shot classification system.

Learning Algorithm This thesis has focused on using the episodic training regime introduced by Vinyals et al. (2016) to train networks to generate parameters that adapt a few-shot, multi-task classifier to a new task in an amortized fashion. The key advantages of this approach are:

- Speed of adaptation. Once meta-trained, only one forward pass of the context set through the parameter generating networks is required to adapt the classifier to a new task.
- Automatic adaptation. There are no parameters to tune to adapt the system to a new task.

These attributes make the the system computationally efficient such that it is suitable for real-time scenarios.

Tian et al. (2020) argue that you don't need a sophisticated meta-learning algorithm at all to be successful on common few-shot image classification benchmarks. Their work shows that given a pre-trained feature extractor, you can use fine-tuning on the context set at meta-test time to learn a final-layer linear classifier that is adapted to the current task. They forego the episodic meta-training step entirely. This approach was evaluated only on fixed shot and way tasks and only on in-distribution data. An extension to their approach would be to add FiLM layers to allow adaptation to out of distribution data where the FiLM layer parameters are learned via fine-tuning. Preliminary experiments that I have performed indicate that fine-tuning both the final layer linear classifier and the FiLM layer parameters significantly outperform tuning only the final layer classifier. In contrast to the amortized approach, fine tuning is slow and is not fully automatic requiring the learning rate and the number of optimization steps to use to be specified. Fine tuning is also more susceptible to over-fitting on few-shot data.

As the number of shots increases, the performance of amortization based meta-learning approaches will suffer compared to fine-tuning as: (i) it is expensive to train the amortization based approaches with high shot; (ii) the capacity of the amortization networks is limited and this can be an issue in the case of a large context set that requires extensive adaptation; (iii) fine-tuning becomes less-likely to over-fit as the context size increases.

Another downside to the meta-learning approach that employs episodic training is that during meta-training, it is not possible to break a task into smaller pieces for processing which puts a limit on the size of a task that can be processed on a single GPU. This forces a trade-off between the number of images in the context and target sets and the size of the images images in the task. The fine-tuning approach does not face such limits as in this case the learning can be easily split in smaller pieces (i.e. a mini-batch) of arbitrary size.

The fundamental trade-off between the fast, but potentially less accurate amortized approach and the slow, but potentially more accurate fine-tuning approach was depicted in Figure 4.6.

Thus, my recommendation for the few-shot learning mechanism is as follows. If the time to adapt a few-shot, multi-task system is critical (i.e. in a real-time or near real-time system), use an amortized approach to generate the FiLM layer parameters in the feature extractor and for the final layer classifier use prototypical networks with Mahalanobis distance metric which has no parameters and has shown superiority over a linear classifier. If you have the time (and potentially more data), use fine-tuning to learn the parameters for the FiLM layers and the final layer linear classifier. Note that I am not recommending any of the few-step gradient methods, such as MAML (Finn et al., 2017), as the amortized approach is both faster and achieves better classification accuracy than any of the few-step gradient methods.

Batch Normalization When meta-training a few-shot image classification system in an episodic manner, I unreservedly recommend the use of TASKNORM-I. In large scale experiments, it resulted in higher classification accuracy and faster training convergence than any other method.

6.2.2 Future Work

In this section we offer a glimpse of what could come next as an outgrowth of the work in this thesis and some thoughts on directions for future work.

Personalization We see the biggest impact of meta-learning in personalized machine learning i.e. rapidly adapting machine learning solutions to an individual by learning only from that user's data. Examples of this include: (i) a personal object recognition system to aid a visually impaired user to locate various personal items with the use of a camera on a smartphone; (ii) generating a talking head model of a user from only a few views (Zakharov et al., 2019). In both cases above, a lengthy meta-training procedure on many few-shot learning tasks is carried out, but once trained, the model is quick to personalize from only a small amount of data. If personalization is data and computation efficient, it will be amenable to execution on an end user's device (such as a smartphone or laptop computer) potentially protecting end-user privacy and eliminating networking delays. Otherwise, if the personalization step requires gradient-based or other compute intensive optimization, the computation may not be feasible on an end user's device requiring that the computation be done in the cloud along with the associated issues of networking, privacy, security, and the cost of running a cloud-based service. Using a CNAPs style architecture with amortized inference networks to generate the personalization parameters would allow the personalization step to be performed on a smart phone.

Designing Few-Shot Learning Systems for Computational Efficiency Neural network models are becoming ever larger and more powerful (e.g. the BERT language representation model has 340 million parameters (Devlin et al., 2018)). Designing meta-learning architectures needs to take into consideration parameter count a well as the time and energy usage to make a prediction on a new task if meta-learning technology is to be incorporated into everyday learning products and services. Using amortized networks that are trained on many few-shot learning tasks as used in the CNAPs system that was detailed in Chapter 4 have a significant advantage in terms of speed of execution and power usage at meta-test time compared to gradient based approaches such as MAML or fine tuning as only a forward pass through the amortization network is required to generate task specific parameters.

Given that a good feature embedding is key to good few-shot image classification performance, work on improving the efficacy of convolutional neural networks becomes important. Recent work on separable depth-wise convolutions (Sifre and Mallat, 2014) and inverted residuals (Sandler et al., 2018) has allowed parameter counts to be drastically reduced with little impact on classification accuracy. For example, the MobileNetv2(1.4) network that uses depth-wise separable convolutions within inverted residual blocks consists of 6.9 million parameters and achieves 74.7% top-1 accuracy on the ImageNet benchmark (Sandler et al., 2018) while the VGG-19 network (Simonyan and Zisserman, 2014) that uses conventional convolutional layers consists of 144 million parameters and achieves a similar 74.7% accuracy. Neural architecture search (NAS) (Zoph and Le, 2016) has also led to smaller, but high performance models, for example MNASNet (Tan et al., 2019). This is an area that is ripe for future investigation and the creation of new efficiency oriented benchmarks.

Continual Learning As outlined in Section 2.5, continual learning is the ability of a machine learning model to continuously learn from new data, building on past learning, and without forgetting what it has learned in the past. Few-shot classification methods that employ amortization networks (including CNAPs) can be directly applied to continual learning scenarios since: (i) like any other few-shot learning system they are meta-trained to adapt to new task data (i.e. they have "learned how to learn"); and (ii) a deep sets based amortization network can generate a compact summary of a context set categorized by class, that when persisted can serve as a simple memory of the class data seen to date which can be easily updated as new data is seen.

Section 4.6 detailed the use of CNAPS in the continual learning setting. The promising CNAPS results imply that there is a strong connection between meta-learning and continual learning. This raises the question as to whether meta-learning would benefit from explicitly using continual learning tasks at training time to improve overall performance as the current approach does not. Other researchers have also been aware of this connection. Vuorio et al. (2018) consider a meta continual learning approach where a neural network is used to predict parameter update steps which constrain parameters that are important to past tasks, but allow large updates for parameters used to learn the current task, which helps mitigate catastrophic forgetting. Antoniou et al. (2020) have defined a new few-shot continual learning benchmark which should spur additional progress in the field.

Meta-Imitation Learning Imitation learning is when a robot or other artificial agent learns a new task by observing many demonstrations of that task. The demonstration may be a video clip or tele-operation in the case of a robot. However, a complex task may require an impractical number of demonstrations to learn from. Meta-Imitation Learning is when the robot or agent is given only a few demonstrations of a new task to learn from at meta-test time, but is meta-trained on many tasks via demonstrations and has "learned how to learn" a new task from only a few demonstrations. Learning from a a small number of demonstrations is an appealing concept for many applications, especially robotics, since they are often easy and intuitive to provide compared to explicitly programming each specific movement of a robotic arm. Learning from a demonstration is particularly challenging as a time dimension is usually involved (e.g. a video demonstration) increasing the amount of data and variability in the tasks. For example, Zhou et al. (2019) introduce a concept called "Watch, Try, Learn" where the goal is for an agent to observe a single demonstration (Watch), then attempt to solve a similar task from what it has learned from the demonstration (Try), receive success or failure indication on the attempt, and from the indication improve its strategy (Learn), such that the agent can successfully complete any similar task. A Conditional Neural Process (CNP - refer back to Section 2.4) approach has been used to learn robot movement primitives (i.e. motion paths) from demonstrations (Seker et al., 2019) in order to solve picking and placing tasks. It would be interesting to extend the CNP approach with CNAPS style adaptation to see if that would improve performance, especially in the case where the movements are learned from video. Meta-imitation learning for robotics is a challenging area of research, but there are many practical applications where the concepts from few-shot learning can be readily applied.

Image Retrieval A potentially interesting avenue of exploration is the connection between information retrieval (Manning et al., 2008), meta-learning, and few-shot learning. In particular, given the work in this thesis, the focus would be on a few-shot learning approach to image retrieval. This idea is not entirely new. Triantafillou et al. (2017) take an information retrieval approach to few-shot learning using mean average precision as the ranking loss function. The authors apply this approach to few-shot classification on Omniglot and *mini*ImageNet with competitive results. They also devise a new 1-shot image retrieval task where they select N classes at random and then 10 examples at random from each class. Each of the $10N$ images acts as a query in turn and ranks the remaining $10N - 1$ images. The goal is that the 9 relevant images are ranked higher that the remaining $10(N-1)$ images, with the performance measured in terms of mean average precision. Wang et al. (2017) present a method for few-shot hash learning for image retrieval where hash functions learned from unlabeled images are used for generating codes of a new image category from only a few examples.

Noh et al. (2017) introduce the Google-Landmarks Dataset as well as a high performance image retrieval system. The Landmarks dataset contains over 1.2 million images of more than 14 thousand landmarks throughout the world. Some landmarks have over 50 thousand images while some have only one. The database has been used for both image recognition as well as image retrieval. Given the large number of landmark classes and the wide range of examples per class, this dataset would be ideal to evaluate few-shot image classification systems with tasks of varying way and shot. The image retrieval system has a multi-step training process which begins with a ResNet50 model pre-trained on ImageNet. The system is then fine-tuned in a supervised fashion on the Landmarks dataset. Finally, attention layers on the feature maps are trained. At this point the system could be used for image classification or image retrieval. In the image retrieval scenario, attention-weighted features from the query images are matched to images in the database using a highly optimized k-nearest neighbors algorithm, making the approach not entirely different from prototypical networks. It would be interesting to try and combine the few-shot image retrieval approach of Triantafillou et al. (2017) with the approach of Noh et al. (2017) to see if a high-scale, meta-learned, few-shot image retrieval system could be realized.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Antoniou, A., Patacchiola, M., Ochal, M., and Storkey, A. (2020). Defining benchmarks for continual few-shot learning. *arXiv preprint arXiv:2004.11967*.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bakker, B. and Heskes, T. (2003). Task clustering and gating for Bayesian multitask learning. *Journal of Machine Learning Research*, 4(May):83–99.

Bateni, P., Goyal, R., Masrani, V., Wood, F., and Sigal, L. (2019). Improved few-shot visual classification. *arXiv preprint arXiv:1912.03432*.

Bauer, M., Rojas-Carulla, M., Świątkowski, J. B., Schölkopf, B., and Turner, R. E. (2017). Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*.

Berger, J. O. (2013). *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media.

Bertinetto, L., Henriques, J. F., Valmadre, J., Torr, P., and Vedaldi, A. (2016). Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*, pages 523–531.

Bird, J. J., Kobylarz, J., Faria, D. R., Ekárt, A., and Ribeiro, E. P. (2020). Cross-domain mlp and cnn transfer learning for biological signal processing: Eeg and emg. *IEEE Access*, 8:54789–54801.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622.

Bouma, S. (2017). One shot learning and siamese networks in keras. https://sorenbouma.github.io/blog/oneshot/. Accessed: 2018-07-25.

Bronskill, J., Gordon, J., Requeima, J., Nowozin, S., and Turner, R. (2020). Tasknorm: Rethinking batch normalization for meta-learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Casella, G. and Berger, R. L. (2002). *Statistical inference*, volume 2. Duxbury Pacific Grove, CA.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. (2018). Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547.

Chen, Y. (2018). A re-implementation of "prototypical networks for few-shot learning". https://github.com/cyvius96/prototypical-network-pytorch.

Chen, Y., Wang, X., Liu, Z., Xu, H., and Darrell, T. (2020). A new meta-baseline for few-shot learning. *arXiv preprint arXiv:2003.04390*.

Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., and Vedaldi, A. (2014). Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613.

Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1996). Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145.

Cremer, C., Li, X., and Duvenaud, D. (2018). Inference suboptimality in variational autoencoders. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1078–1086, Stockholmsmässan, Stockholm Sweden. PMLR.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE.

Dawid, A. P. (2007). The geometry of proper scoring rules. *Annals of the Institute of Statistical Mathematics*, 59(1):77–93.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Duda, R. O., Hart, P. E., and Stork, D. G. (2012). *Pattern classification*. John Wiley & Sons.

Edelen, J., Li, J., Bronskill, J. F., Guiver, J. P., Dastgir, K., Rajmohan, S., and Sadovsky, A. (2019). Personalized predictive models. US Patent 10,504,029.

Edwards, H. and Storkey, A. (2017). Towards a neural statistician. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Efron, B. (1975). The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, 70(352):892–898.

Eslami, S. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., et al. (2018). Neural scene representation and rendering. *Science*, 360(6394):1204–1210.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, International Convention Centre, Sydney, Australia. PMLR.

Finn, C., Xu, K., and Levine, S. (2018). Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*.

Finn, C. B. (2017). Code for "Model-agnostic meta-learning for fast adaptation of deep networks". https://github.com/cbfinn/maml.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

Foong, A. Y., Bruinsma, W. P., Gordon, J., Dubois, Y., Requeima, J., and Turner, R. E. (2020). Meta-learning stationary stochastic process prediction with convolutional neural processes. *arXiv preprint arXiv:2007.01332*.

French, R. M. (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Furukawa, Y., Hernández, C., et al. (2015). Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org.

Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y. W., Rezende, D., and Eslami, S. M. A. (2018a). Conditional neural processes. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713, Stockholmsmässan, Stockholm Sweden. PMLR.

Garnelo, M., Schwarz, J., Rosenbaum, D., Viola, F., Rezende, D. J., Eslami, S., and Teh, Y. W. (2018b). Neural processes. *arXiv preprint arXiv:1807.01622*.

Geisser, S. (1983). On the prediction of observables: a selective update. Technical report, University of Minnesota.

Geisser, S. (2017). *Predictive inference*. Routledge.

Gidaris, S. and Komodakis, N. (2018). Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. (2019). Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. In *International Conference on Learning Representations*.

Ha, D. and Eck, D. (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*.

Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge university press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Heskes, T. (2000). Empirical bayes for learning to learn. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 367–374, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

Hoi, S. C., Sahoo, D., Lu, J., and Zhao, P. (2018). Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*.

Hospedales, T., Antoniou, A., Micaelli, P., and Storkey, A. (2020). Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*.

Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M., and Igel, C. (2013). Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE.

Huszar, F. (2013). *Scoring rules, divergences and information in Bayesian machine learning*. PhD thesis, University of Cambridge.

Iakovleva, E., Verbeek, J., and Alahari, K. (2020). Meta-learning with shared amortized variational inference. *arXiv preprint arXiv:2008.12037*.

Ioffe, S. (2017). Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 1945–1953. Curran Associates, Inc.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. and Blei, D., editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France. PMLR.

Jaynes, E. T. (2003). *Probability theory: the logic of science*. Cambridge university press.

Kaiser, Ł., Nachum, O., Aurko, R., and Bengio, S. (2017). Learning to remember rare events. In *International Conference on Learning Representations (ICLR)*.

Kim, H., Mnih, A., Schwarz, J., Garnelo, M., Eslami, A., Rosenbaum, D., Vinyals, O., and Teh, Y. W. (2019). Attentive neural processes. In *International Conference on Learning Representations*.

Kim, Y., Wiseman, S., Miller, A. C., Sontag, D., and Rush, A. M. (2018). Semi-amortized variational autoencoders. In *Proceedings of the 35th International Conference on Machine Learning*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. (2014). Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pages 3581–3589.

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lacoste, A., Oreshkin, B., Chung, W., Boquet, T., Rostamzadeh, N., and Krueger, D. (2018). Uncertainty in multitask transfer learning. *arXiv preprint arXiv:1806.07528*.

Lacoste-Julien, S., Huszár, F., and Ghahramani, Z. (2011). Approximate inference for the loss-calibrated Bayesian. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 416–424.

Lake, B., Salakhutdinov, R., Gross, J., and Tenenbaum, J. (2011). One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

LeCun, Y., Cortes, C., and Burges, C. (2010). MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18.

LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-sgd: Learning to learn quickly for few shot learning. *arXiv preprint arXiv:1707.09835*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Liu, L., Hamilton, W., Long, G., Jiang, J., and Larochelle, H. (2020). A universal representation transformer layer for few-shot image classification. *arXiv preprint arXiv:2006.11702*.

Liu, Q. and Wang, D. (2016). Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pages 2378–2386.

Liu, Y., Lee, J., Park, M., Kim, S., Yang, E., Hwang, S., and Yang, Y. (2019). Learning to propagate labels: transductive propagation network for few-shot learning. In *International Conference on Learning Representations*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Luo, C., Zhan, J., Xue, X., Wang, L., Ren, R., and Yang, Q. (2018). Cosine normalization: Using cosine similarity instead of dot product in neural networks. In *International Conference on Artificial Neural Networks*, pages 382–391. Springer.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605.

Maji, S., Rahtu, E., Kannala, J., Blaschko, M., and Vedaldi, A. (2013). Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*.

Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge university press.

Maudsley, D. B. (1980). A theory of meta-learning and principles of facilitation: An organismic perspective.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2018). A simple neural attentive meta-learner. In *International Conference on Learning Representations*.

Mohamed, S. (2018). Density ratio trick. http://blog.shakirm.com/2018/01/machine-learning-trick-of-the-day-7-density-ratio-trick/.

Naik, D. K. and Mammone, R. (1992). Meta-neural networks that learn by learning. In *Neural Networks, 1992. IJCNN., International Joint Conference on*, volume 1, pages 437–442. IEEE.

Nam, H. and Kim, H.-E. (2018). Batch-instance normalization for adaptively style-invariant neural networks. In *Advances in Neural Information Processing Systems*, pages 2558–2567.

Narayanaswamy, S., Paige, T. B., van de Meent, J.-W., Desmaison, A., Goodman, N., Kohli, P., Wood, F., and Torr, P. (2017). Learning disentangled representations with semi-supervised deep generative models. In *Advances in Neural Information Processing Systems*, pages 5927–5937.

Ng, A. Y. and Jordan, M. I. (2002). On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems*, pages 841–848.

Nguyen, C. V., Hassner, T., Archambeau, C., and Seeger, M. (2020). Leep: A new measure to evaluate transferability of learned representations. *arXiv preprint arXiv:2002.12462*.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning. *arXiv preprint arXiv:1710.10628*.

Nichol, A., Achiam, J., and Schulman, J. (2018). On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*.

Nilsback, M.-E. and Zisserman, A. (2008). Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE.

Noh, H., Araujo, A., Sim, J., Weyand, T., and Han, B. (2017). Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465.

Oreshkin, B. N., Lacoste, A., and Rodriguez, P. (2018). TADAM: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). FiLM: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Pohar, M., Blas, M., and Turk, S. (2004). Comparison of logistic regression and linear discriminant analysis: a simulation study. *Metodoloski zvezki*, 1(1):143.

Pratt, L. Y., Mostow, J., Kamm, C. A., and Kamm, A. A. (1991). Direct transfer of learned information among neural networks. In *AAAI*, volume 91, pages 584–589.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

Qiao, S., Liu, C., Shen, W., and Yuille, A. L. (2018). Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7229–7238.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. (2019). Meta-learning with implicit gradients. In Wallach, H., Larochelle, H., Beygelzimer, A., d' Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 113–124. Curran Associates, Inc.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *Proceedings of the International Conference on Learning Representations*.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2017). Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516.

Rebuffi, S.-A., Bilen, H., and Vedaldi, A. (2018). Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127.

Ren, M., Ravi, S., Triantafillou, E., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. (2018). Meta-learning for semi-supervised few-shot classification. In *International Conference on Learning Representations*.

Rendell, L. A., Sheshu, R., and Tcheng, D. K. (1987). Layered concept-learning and dynamically variable bias management. In *IJCAI*, pages 308–314.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. (2019a). Code for "Fast and flexible multi-task classification using conditional neural adaptive processes". https://github.com/cambridge-mlg/cnaps.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. (2019b). Fast and flexible multi-task classification using conditional neural adaptive processes. In Wallach, H., Larochelle, H., Beygelzimer, A., dÁlché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 7957–7968. Curran Associates, Inc.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286.

Ring, M. B. (1997). Child: A first step towards continual learning. *Machine Learning*, 28(1):77–104.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2018). Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*.

Saad, D. (2009). *On-line learning in neural networks*, volume 17. Cambridge University Press.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520.

Satorras, V. G. and Estrach, J. B. (2018). Few-shot learning with graph neural networks. In *International Conference on Learning Representations*.

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

Schmidhuber, J. (1993). A 'self-referential'weight matrix. In *International Conference on Artificial Neural Networks*, pages 446–450. Springer.

Schroeder, B. and Cui, Y. (2018). Fgvcx fungi classification challenge at fgvc5. https://www.kaggle.com/c/fungi-challenge-fgvc-2018.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress compress: A scalable framework for continual learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537, Stockholmsmässan, Stockholm Sweden. PMLR.

Seker, M. Y., Imre, M., Piater, J., and Ugur, E. (2019). Conditional neural movement primitives. In *Robotics: Science and Systems (RSS)*.

Settles, B. (2012). Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–114.

Sifre, L. and Mallat, S. (2014). Rigid-motion scattering for image classification. *Ph. D. thesis*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Singh, S. and Krishnan, S. (2019). Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. *arXiv preprint arXiv:1911.09737*.

Snell, J. (2017). Code for the nips 2017 paper "prototypical networks for few-shot learning". https://github.com/jakesnell/prototypical-networks.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Sugiyama, M., Suzuki, T., and Kanamori, T. (2012). *Density ratio estimation in machine learning*. Cambridge University Press.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. (2018). Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Swaroop, S., Nguyen, C. V., Bui, T. D., and Turner, R. E. (2019). Improving and understanding variational continual learning. *arXiv preprint arXiv:1905.02099*.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828.

Tan, M. and Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

Thrun, S. and Pratt, L. (2012). *Learning to learn*. Springer Science & Business Media.

Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J. B., and Isola, P. (2020). Rethinking few-shot image classification: a good embedding is all you need? *arXiv preprint arXiv:2003.11539*.

Triantafillou, E., Zemel, R., and Urtasun, R. (2017). Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems*, pages 2255–2265.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., and Larochelle, H. (2020). Meta-dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations*.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P.-A., and Larochelle, H. (2019). Code for "Meta-dataset: A dataset of datasets for learning to learn from few examples". https://github.com/google-research/meta-dataset.

Trippe, B. and Turner, R. (2018). Overpruning in variational bayesian neural networks. *arXiv preprint arXiv:1801.06230*.

Tseng, H.-Y., Lee, H.-Y., Huang, J.-B., and Yang, M.-H. (2020). Cross-domain few-shot classification via learned feature-wise transformation. *arXiv preprint arXiv:2001.08735*.

Turner, R. E. (2018). Notes from review of neural scene representation and rendering.

Turner, R. E. and Sahani, M. (2011). Two problems with variational expectation maximisation for time-series models. *Bayesian Time series models*, 1(3.1):3–1.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016). Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. *Machine learning: An artificial intelligence approach*, 2:107–148.

Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., and Larochelle, H. (2017). A meta-learning perspective on cold-start recommendations for items. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6904–6914. Curran Associates, Inc.

Vinyals, O., Blundell, C., Lillicrap, T., kavukcuoglu, k., and Wierstra, D. (2016). Matching networks for one shot learning. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc.

Vuorio, R., Cho, D.-Y., Kim, D., and Kim, J. (2018). Meta continual learning. *arXiv preprint arXiv:1806.06928*.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. (2011). The caltech-ucsd birds-200-2011 dataset.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1-2):1–305.

Wang, Y., Yao, Q., Kwok, J., and Ni, L. M. (2019). Generalizing from a few examples: A survey on few-shot learning. *arXiv preprint arXiv:1904.05046*.

Wang, Y.-X., Gui, L., and Hebert, M. (2017). Few-shot hash learning for image retrieval. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1228–1237.

Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19.

Xiong, W., Wu, L., Alleva, F., Droppo, J., Huang, X., and Stolcke, A. (2018). The microsoft 2017 conversational speech recognition system. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5934–5938. IEEE.

Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. (2018). Bayesian model-agnostic meta-learning. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 7332–7342. Curran Associates, Inc.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404.

Zakharov, E., Shysheya, A., Burkov, E., and Lempitsky, V. (2019). Few-shot adversarial learning of realistic neural talking head models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9459–9468.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org.

Zhang, C., Butepage, J., Kjellstrom, H., and Mandt, S. (2017). Advances in variational inference. *arXiv preprint arXiv:1711.05597*.

Zhou, A., Jang, E., Kappler, D., Herzog, A., Khansari, M., Wohlhart, P., Bai, Y., Kalakrishnan, M., Levine, S., and Finn, C. (2019). Watch, try, learn: Meta-learning from demonstrations and reward. *arXiv preprint arXiv:1906.03352*.

Zhuo, J., Liu, C., Shi, J., Zhu, J., Chen, N., and Zhang, B. (2017). Message passing stein variational gradient descent. *arXiv preprint arXiv:1711.04425*.

Zintgraf, L. M., Shiarlis, K., Kurin, V., Hofmann, K., and Whiteson, S. (2018). CAML: Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Appendix A

Additional Modeling Details and Experiments

A.1 Additional ML-PIP Modeling Details and Experiments

A.1.1 Bayesian Decision Theoretic Generalization of ML-PIP

A generalization of the new inference framework presented in Section 3.2 is based upon Bayesian decision theory (BDT). BDT provides a recipe for making predictions $\hat{\boldsymbol{y}}$ for an unknown test variable $\boldsymbol{y}^*$ by combining information from observed training data D^τ (here from a single task τ) and a loss function $L(y^*, \hat{y})$ that encodes the cost of predicting $\hat{\boldsymbol{y}}$ when the true value is $\boldsymbol{y}^*$ (Berger, 2013; Jaynes, 2003). In BDT an optimal prediction minimizes the expected loss (suppressing dependencies on the inputs and θ to reduce notational clutter):[1]

$$\hat{\boldsymbol{y}}^* = \underset{\hat{\boldsymbol{y}}}{\text{argmin}} \int p(\boldsymbol{y}^*|D^\tau) L(\boldsymbol{y}^*, \hat{\boldsymbol{y}}) \mathrm{d}\boldsymbol{y}^*, \quad \text{where} \quad p(\boldsymbol{y}^*|D^\tau) = \int p(\boldsymbol{y}^*|\boldsymbol{\psi}^\tau) p(\boldsymbol{\psi}^\tau|D^\tau) \mathrm{d}\boldsymbol{\psi}^\tau \quad (\text{A.1})$$

is the Bayesian predictive distribution and $p(\boldsymbol{\psi}^\tau|D^\tau)$ the posterior distribution of $\boldsymbol{\psi}^\tau$ given the training data from task τ.

BDT separates test and training data and so is a natural lens through which to view recent episodic approaches to training that utilize many internal training/test splits (Vinyals et al., 2016). Based on this insight, what follows is a fairly dense derivation of an ultimately simple stochastic variational objective for meta-learning probabilistic inference that is rigorously grounded in Bayesian inference and decision theory.

Distributional BDT. We generalize BDT to cases where the goal is to return a full predictive *distribution* $q(\cdot)$ over the unknown test variable $\boldsymbol{y}^*$ rather than a point prediction. The quality of q is quantified through a distributional loss function $L(\boldsymbol{y}^*, q(\cdot))$. Typically, if $\boldsymbol{y}^*$ (the true

[1]For discrete outputs the integral may be replaced with a summation.

value of the underlying variable) falls in a low probability region of $q(\cdot)$ the loss will be high, and vice versa. The optimal predictive q^* is found by optimizing the expected distributional loss with q constrained to a distributional family $\mathcal{Q}$:

$$q^* = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \int p(\boldsymbol{y}^*|D^\tau) L(\boldsymbol{y}^*, q(\cdot)) \mathrm{d}\boldsymbol{y}^*. \tag{A.2}$$

Amortized variational training. Here, we amortize q to form quick predictions at test time and learn parameters by minimizing average expected loss *over tasks*. Let $\boldsymbol{\phi}$ be a set of shared variational parameters such that $q(\boldsymbol{y}^*) = q_{\boldsymbol{\phi}}(\boldsymbol{y}^*|D)$ (or $q_{\boldsymbol{\phi}}$ for shorthand). Now the approximate predictive distribution can take any training dataset D^τ as an argument and directly perform prediction of $\boldsymbol{y}^{\tau *}$. The optimal variational parameters are found by minimizing the expected distributional loss across tasks

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \mathcal{L}\left[q_{\boldsymbol{\phi}}\right], \; \mathcal{L}\left[q_{\boldsymbol{\phi}}\right] = \int p(D) p(\boldsymbol{y}^*|D) L(\boldsymbol{y}^*, q_{\boldsymbol{\phi}}(\cdot|D)) \mathrm{d}\boldsymbol{y}^* \, \mathrm{d}D = \mathbb{E}_{p(D,\boldsymbol{y}^*)}[L(\boldsymbol{y}^*, q_{\boldsymbol{\phi}}(\cdot|D))] \,. \tag{A.3}$$

Here the variables $D, \boldsymbol{x}^*$ and $\boldsymbol{y}^*$ are placeholders for integration over all possible datasets, test inputs and outputs. Note that Equation (A.3) can be stochastically approximated by sampling a task t and randomly partitioning into training data D and test data $\{\boldsymbol{x}^*_m, \boldsymbol{y}^*_m\}_{m=1}^M$, which naturally recovers episodic mini-batch training over tasks and data (Vinyals et al., 2016; Ravi and Larochelle, 2017). Critically, this does not require computation of the true predictive distribution. It also emphasizes the meta-learning aspect of the procedure, as the model is *learning how to infer* predictive distributions from training tasks.

Loss functions. We employ the log-loss: the negative log density of $q_{\boldsymbol{\phi}}$ at $\boldsymbol{y}^*$. In this case,

$$\mathcal{L}\left[q_{\boldsymbol{\phi}}\right] = \mathbb{E}_{p(D,\boldsymbol{y}^*)}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{y}^*|D)\right] = \mathbb{E}_{p(D)}\left[\mathrm{KL}\left[p(\boldsymbol{y}^*|D) \| q_{\boldsymbol{\phi}}(\boldsymbol{y}^*|D)\right] + \mathrm{H}\left[p(\boldsymbol{y}^*|D)\right]\right], \tag{A.4}$$

where $\mathrm{KL}[p(\boldsymbol{y})\|q(\boldsymbol{y})]$ is the KL-divergence, and $\mathrm{H}\left[p(\boldsymbol{y})\right]$ is the entropy of p. Equation (A.4) has the elegant property that the optimal $q_{\boldsymbol{\phi}}$ is the closest member of $\mathcal{Q}$ (in a KL sense) to the true predictive $p(\boldsymbol{y}^*|D)$, which is unsurprising as the log-loss is a proper scoring rule (Huszar, 2013). This is reminiscent of the sleep phase in the wake-sleep algorithm (Hinton et al., 1995). Exploration of alternative proper scoring rules (Dawid, 2007) and more task-specific losses (Lacoste-Julien et al., 2011) is left for future work.

Specification of the approximate predictive distribution. Next, we consider the form of $q_{\boldsymbol{\phi}}$. Motivated by the optimal predictive distribution, we replace the true posterior by an approximation:

$$q_{\boldsymbol{\phi}}(\boldsymbol{y}^*|D) = \int p(\boldsymbol{y}^*|\boldsymbol{\psi}) q_{\boldsymbol{\phi}}(\boldsymbol{\psi}|D) \mathrm{d}\boldsymbol{\psi}. \tag{A.5}$$

A.1.2 Justification for Context-Independent Approximation

In this section we lay out both theoretical and empirical justifications for the context-independent approximation detailed in Section 3.3.

Theoretical Argument – Density Ratio Estimation

A principled justification for the approximation is best understood through the lens of density ratio estimation (Mohamed, 2018; Sugiyama et al., 2012). We denote the conditional density of each class as $p(x|y=c)$ and assume equal a priori class probability $p(y=c)=1/C$. Density ratio theory then uses Bayes' theorem to show that the optimal softmax classifier can be expressed in terms of the conditional densities (Mohamed, 2018; Sugiyama et al., 2012):

$$\text{Softmax}(y=c|x) = \frac{\exp(h(x)^\top \boldsymbol{w}_c)}{\sum_{c'} \exp(h(x)^\top \boldsymbol{w}_{c'})} = p(y=c|x) = \frac{p(x|y=c)}{\sum\limits_{c'} p(x|y=c')}, \tag{A.6}$$

This implies that the optimal classifier will construct estimators for the conditional density for each class, that is $\exp(h(x)^\top \boldsymbol{w}_c) \propto p(x|y=c)$. Importantly for our approximation, notice that these estimates are constructed *independently* for each class, similarly to training a naive Bayes classifier. VERSA mirrors this optimal form using:

$$\log p(x|y=c) \propto h_\theta(\tilde{x})^\top \boldsymbol{w}_c, \tag{A.7}$$

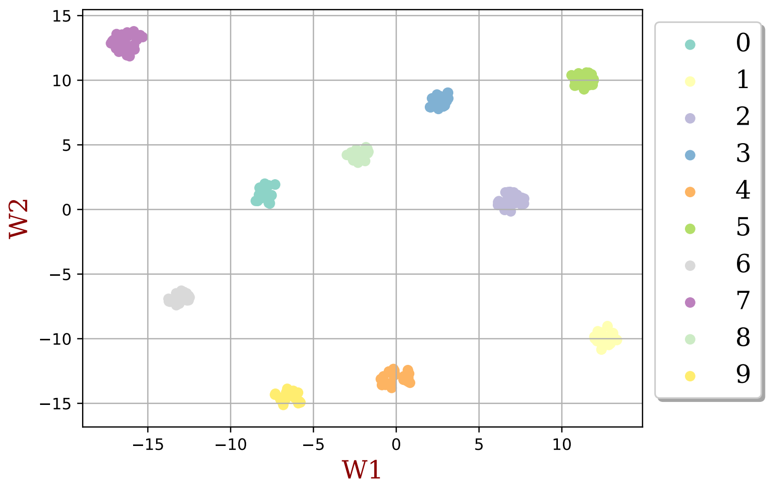
where $w_c \sim q_\phi\left(\boldsymbol{w}|\{x_n|y_n=c\}\right)$ for each class in a given task. Under ideal conditions (i.e., if one could perfectly estimate $p(x^*|y=c)$), the context-independent assumption holds, further motivating our design.

Empirical Justification

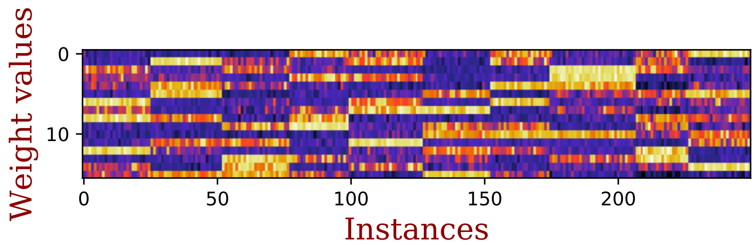
Here we detail a simple experiment to evaluate the validity of the context-independent inference assumption. The goal of the experiment is to examine if weights may be context-independent without imposing the assumption on the amortization network. To see this, we randomly generate fifty tasks from a dataset, where classes may appear a number of times in different tasks. We then perform free-form (non-amortized) variational inference on the weights for each of the tasks, with a Gaussian variational distribution:

$$q_\phi\left(W^\tau|\mathcal{D}^\tau, \boldsymbol{\theta}\right) = \mathcal{N}\left(W^\tau; \mu_\phi^\tau, \sigma_\phi^{\tau 2}\right). \tag{A.8}$$

If the assumption is reasonable, we may expect the distribution of the weights of a specific class to be similar regardless of the additional classes in the task.



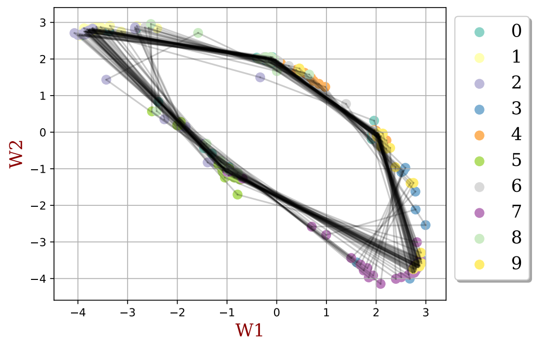
(a)



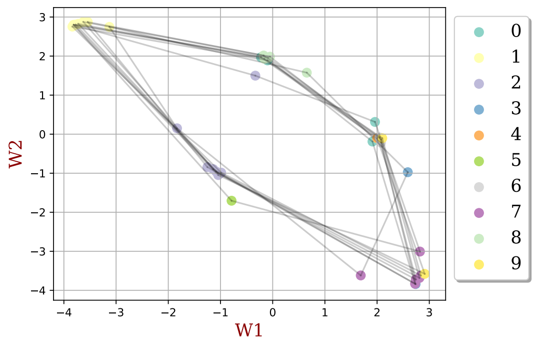
(b)

Fig. A.1 Visualizing the learned weights for $d_{\boldsymbol{\theta}} = 16$. (a) Weight dimensionality is reduced using T-SNE (Maaten and Hinton, 2008). Weights are colored according to class. (b) Each weight represents one column of the image. Weights are grouped by class.

We examine 5-way classification in the MNIST dataset. We randomly sample and fix fifty such tasks. We train the model twice using the same feature extraction network used in the few-shot classification experiments, and fix the $d_{\boldsymbol{\theta}}$ to be 16 and 2. We then train the model in an episodic manner by mini-batching tasks at each iteration. The model is trained to convergence, and achieves 99% accuracy on held out test examples for the tasks. After training is complete we examine the optimized μ_{ϕ}^{τ} for each class in each task. Figure A.1a shows a t-SNE (Maaten and Hinton, 2008) plot for the 16-dimensional weights. We see that when reduced to 2-dimensions, the weights cluster according to class. Figure A.1b visualizes the weights in their original space. In this plot, weights from the same class are grouped together, and clear similarity patterns are evident across the image, showing that weights from the same class have similar means across tasks. Figure A.2 details the task weights in 2-dimensional space. Here, each pentagon represents the weight means learned for one training task, where the nodes of the pentagon are colored according to the class the weights represent. In Figure A.2a we see that overall, the classes cluster in 2-dimensional space as well. However, there is some



(a)



(b)

Fig. A.2 Visualizing the task weights for $d_{\boldsymbol{\theta}} = 2$. (a) All training tasks. (b) Only training tasks containing both '1's and '2's.

overlap (e.g., classes '1' and '2'), and that for some tasks a class-weight may appear away from the cluster. Figure A.2b shows the same plot, but only for tasks that contain both class '1' and '2'. Here we can see that for these tasks, class '2' weights are all located away from their cluster.

This implies that each class-weights are typically well-approximated as being independent of the task. However, if the model lacks capacity to properly assign each set of class weights to different regions of space, for tasks where classes from similar regions of space appear, the inference procedure will 'move' one of the class weights to an 'empty' region of the space.

A.2 Additional CNAPs Few-Shot Classification Results

A.2.1 Joint Training of θ and ϕ

Our experiments in jointly training $\boldsymbol{\theta}$ and ϕ show that the two-stage training procedure proposed in Section 4.3 is crucially important. In particular, we found that joint training diverged in almost all cases we attempted. We were only able to train jointly in two circumstances: (i) Using batch normalization in "train" mode for both context *and* target sets. We stress that this implies computing the batch statistics at test time, and using those to normalize the batches. This is in contrast to the methodology we propose in the main text: only using batch normalization in "eval" mode, which enforces that no information is transferred across tasks or datasets. (ii) "Warm-start" the training procedure with batch normalization in "train" mode, and after a number of epochs (we use 50 for the results shown below), switch to proper usage of batch normalization. All other training procedures we attempted diverged.

Table A.1 details the results of our study on training procedures. The results demonstrate that the two-stage greatly improves performance of the model, even compared to using batch normalization in "train mode", which gives the model an unfair advantage over our standard model.

Dataset	Joint Training (warmstart BN)	Joint Training (BN train mode)	Two-Stage Training (BN test mode)
ILSVRC	17.3±0.7	41.6±1.0	49.5±1.0
Omniglot	74.9±1.0	80.8±0.9	89.7±0.5
Aircraft	51.4±0.8	70.5±0.7	87.2±0.5
Birds	44.1±1.0	48.3±1.0	76.7±0.9
Textures	49.1±0.7	73.5±0.6	83.0±0.6
Quick Draw	46.6±1.0	71.5±0.8	72.3±0.8
Fungi	20.4±0.9	43.1±1.1	50.5±1.1
VGG Flower	66.6±0.8	71.0±0.7	92.5±0.4
Traffic Signs	21.2±0.8	40.4±1.1	48.4±1.1
MSCOCO	18.8±0.7	37.1±1.0	39.7±0.9

Table A.1 Few-shot classification results on META-DATASET (Triantafillou et al., 2020) comparing joint training for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ (columns 2 and 3) to two-stage training (column 4). All figures are percentages and the $\pm$ sign indicates the 95% confidence interval.

A.2.2 Comparison Between CNAPs and Parallel Residual Adapters

CNAPs adds FiLM layers (Perez et al., 2018) in series with each convolutional layer to adapt the feature extractor to a particular task while parallel residual adapters from Rebuffi et al. (2018) adds 1×1 convolutions in parallel with each convolution layer to do the same. However, if the number of feature channels is C, then the number of parameters required for each convolutional layer in the feature extractor is $2C$ for CNAPs and C^2 for parallel residual adapters. Hence, parallel residual adapters have $C/2$ times the capacity compared to FiLM layers. Despite this advantage, CNAPs achieves superior results as can be seen in Table A.2.

Dataset	Parallel Residual Adapter	CNAPs
ILSVRC	**51.2** $\pm$ **1.0**	**52.3** $\pm$ **1.0**
Omniglot	**87.3** $\pm$ **0.7**	**88.4** $\pm$ **0.7**
Aircraft	78.3 $\pm$ 0.7	**80.5** $\pm$ **0.6**
Birds	67.8 $\pm$ 0.9	**72.2** $\pm$ **0.9**
Textures	55.5 $\pm$ 0.7	**58.3** $\pm$ **0.7**
Quick Draw	70.9 $\pm$ 0.7	**72.5** $\pm$ **0.8**
Fungi	44.6 $\pm$ 1.1	**47.4** $\pm$ **1.0**
VGG Flower	81.7 $\pm$ 0.7	**86.0** $\pm$ **0.5**
Traffic Signs	57.2 $\pm$ 0.9	**60.2** $\pm$ **0.9**
MSCOCO	**43.7** $\pm$ **1.0**	**42.6** $\pm$ **1.1**
MNIST	91.1 $\pm$ 0.4	**92.7** $\pm$ **0.4**
CIFAR10	**64.5** $\pm$ **0.8**	61.5 $\pm$ 0.7
CIFAR100	**50.4** $\pm$ **0.9**	**50.1** $\pm$ **1.0**

Table A.2 Few-shot classification results on META-DATASET (Triantafillou et al., 2020) using models trained on all training datasets for Parallel Residual Adapters (Rebuffi et al., 2018) and CNAPs. All figures are percentages and the $\pm$ sign indicates the 95% confidence interval over tasks. Bold text indicates the scores within the confidence interval of the highest score. Tasks from datasets below the dashed line were not used for training.

Appendix B

Experiment Details

B.1 VERSA Experimentation Details

In this section we provide comprehensive details on the VERSA few-shot classification experiments.

B.1.1 Omniglot Few-shot Classification Training Procedure

Omniglot (Lake et al., 2011) is a few-shot learning dataset consisting of 1623 handwritten characters (each with 20 instances) derived from 50 alphabets. We follow a pre-processing and training procedure akin to that defined in (Vinyals et al., 2016). First the images are resized to 28×28 pixels and then character classes are augmented with rotations of 90 degrees. The training, validation and test sets consist of a random split of 1100, 100, and 423 characters, respectively. When augmented this results in 4400 training, 400 validation, and 1292 test classes, each having 20 character instances. For C-way, k_c-shot classification, training proceeds in an episodic manner. Each training iteration consists of a batch of one or more tasks. For each task C classes are selected at random from the training set. During training, k_c character instances are used as training inputs and 15 character instances are used as test inputs. The validation set is used to monitor the progress of learning and to select the best model to test, but does not affect the training process. Final evaluation of the trained model is done on 600 randomly selected tasks from the test set. During evaluation, k_c character instances are used as training inputs and k_c character instances are used as test inputs. We use the Adam (Kingma and Ba, 2015) optimizer with a constant learning rate of 0.0001 with 16 tasks per batch to train all models. The 5-way - 5-shot and 5-way - 1-shot models are trained for 80,000 iterations while the 20-way - 5-shot model is trained for 60,000 iterations, and the 20-way - 1-shot model is trained for 100,000 iterations. In addition, we use a Gaussian form for q and set the number of ψ samples to $L = 10$.

B.1.2 *mini*ImageNET Few-shot Classification Training Procedure

*mini*ImageNet (Vinyals et al., 2016) is a dataset of 60,000 color images that is sub-divided into 100 classes, each with 600 instances. The images have dimensions of 84×84 pixels. For our experiments, we use the 64 training, 16 validation, and 20 test class splits defined by (Ravi and Larochelle, 2017). Training proceeds in the same episodic manner as with Omniglot. We use the Adam (Kingma and Ba, 2015) optimizer and a Gaussian form for q and set the number of ψ samples to $L = 10$. For the 5-way - 5-shot model, we train using 4 tasks per batch for 100,000 iterations and use a constant learning rate of 0.0001. For the 5-way - 1-shot model, we train with 8 tasks per batch for 50,000 iterations and use a constant learning rate of 0.00025.

B.2 CNAPS Experimentation Details

All experiments were implemented in PyTorch (Paszke et al., 2019) and executed either on NVIDIA Tesla P100-PCIE-16GB or Tesla V100-SXM2-16GB GPUs. The full CNAPS model runs in a distributed fashion across 2 GPUs and takes approximately one and a half days to complete episodic training and testing.

B.2.1 META-DATASET Training and Evaluation Procedure

Feature Extractor Weights θ Pretraining

We first reduce the size of the images in the ImageNet ILSVRC-2012 dataset (Russakovsky et al., 2015) to 84 × 84 pixels. Some images in the ImageNet ILSVRC-2012 dataset are duplicates of images in other datasets included in META-DATASET, so these were removed. We then split the 1000 training classes of the ImageNet ILSVRC-2012 dataset into training, validation, and test sets according to the criteria detailed in (Triantafillou et al., 2020). The test set consists of the 130 leaf-node subclasses of the “device" synset node, the validation set consists of the the 158 leaf-node subclasses of the “carnivore" synset node, and the training set consists of the remaining 712 leaf-node classes. We then pretrain a feature extractor with parameters θ based on a modified ResNet-18 (He et al., 2016) architecture on the above 712 training classes. The ResNet-18 architecture is detailed in Table C.10. Compared to a standard ResNet-18, we reduced the initial convolution kernel size from 7 to 5 and eliminated the initial max-pool step. These changes were made to accommodate the reduced size of the imagenet training images. We train for 125 epochs using stochastic gradient descent with momentum of 0.9, weight decay equal to 0.0001, a batch size of 256, and an initial learning rate of 0.1 that decreases by a factor of 10 every 25 epochs. During pretraining, the training dataset was augmented with random crops, random horizontal flips, and random color jitter. The top-1 accuracy after pretraining was 63.9%. For all subsequent training and evaluation steps, the ResNet-18 weights were frozen.The dimensionality of the feature extractor output is $d_f = 512$. The hyper-parameters used were derived from the PyTorch (Paszke et al., 2019) ResNet training tutorial. The only tuning that was performed was on the number of epochs used for training and the interval at which the learning rate was decreased. For the number of epochs, we tried both 90 and 125 epochs and selected 125, which resulted in slightly higher accuracy. We also found that dropping the learning rate at an interval of 25 versus 30 epochs resulted in slightly higher accuracy.

Episodic Training of ϕ

Next we train the functions that generate the parameters ψ_f^τ, ψ_w^τ for the feature extractor adapters and the linear classifier, respectively. We train two variants of CNAPS (on ImageNet ILSVRC-2012 only and all datasets - see Table B.1). We generate training and validation episodes using the reader from (Triantafillou et al., 2019). We train in an end-to-end fashion for 110,000 episodes with the Adam (Kingma and Ba, 2015) optimizer, using a batch size of 16 episodes, and a fixed learning rate of 0.0005. We validate using 200 episodes per validation dataset. Note that when training on ILSVRC only, we validate on ILSVRC only, however, when training on all datasets, we validate on all datasets that have validation data (see Table B.1) and consider a model to be better if more than half of the datasets have a higher classification accuracy than the current best model. No data augmentation was employed during the training of ϕ. Note that while training ϕ the feature extractor $f_{\boldsymbol{\theta}}(\cdot)$ is in 'eval' mode (i.e. it will use the fixed batch normalization statistics learned during pretraining the feature extractor weights $\boldsymbol{\theta}$ with a moving average). No batch normalization is used in any of the functions generating the ψ^τ parameters, with the exception of the set encoder g (that generates the global task representation z_G^τ). Note that the target points are never passed through the set encoder g. Again, very little hyper-parameter tuning was performed. No grid search or other hyper-parameter search was used. For learning rate we tried both 0.0001 and 0.0005, and selected the latter. We experimented with the number of training episodes in the range of 80,000 to 140,000, with 110,000 episodes generally yielding the best results. We also tried lowering the batch size to 8, but that led to decreased accuracy.

Evaluation

We generate test episodes using the reader from (Triantafillou et al., 2019). We test all models with 600 episodes each on all test datasets. The classification accuracy is averaged over the episodes and a 95% confidence interval is computed. We compare the best validation and fully trained models in terms of accuracy and use the best of the two. Note that during evaluation, the feature extractor $f_{\boldsymbol{\theta}}(\cdot)$ is also in 'eval' mode.

ImageNet ILSVRC-2012			All Datasets		
Train	**Validation**	**Test**	**Train**	**Validation**	**Test**
ILSVRC	ILSVRC	ILSVRC	ILSVRC	ILSVRC	ILSVRC
		Omniglot	Omniglot	Omniglot	Omniglot
		Aircraft	Aircraft	Aircraft	Aircraft
		Birds	Birds	Birds	Birds
		Textures	Textures	Textures	Textur
		Quick Draw	Quick Draw	Quick Draw	Quick Draw
		Fungi	Fungi	Fungi	Fungi
		VGG Flower	VGG Flower	VGG Flower	VGG Flower
		MSCOCO		MSCOCO	MSCOCO
		Traffic Signs			Traffic Signs
		MNIST			MNIST
		CIFAR10			CIFAR10
		CIFAR100			CIFAR100

Table B.1 Datasets used to train, validate, and test models.

B.2.2 Continual Learning Experimentation Details

Split MNIST Benchmark The MNIST dataset (LeCun et al., 2010) consists of ten classes of handwritten digits (zero through nine). In the split MNIST benchmark, the MNIST dataset is split into five disjoint subsets of two consecutive digits i.e. $\{\{0,1\},\{2,3\},\{4,5\},\{6,7\},\{8,9\}\}$ (Chaudhry et al., 2018). These five subsets serve as a sequence of tasks to learn consecutively.

Split CIFAR-100 Benchmark The CIFAR 100 dataset (Krizhevsky and Hinton, 2009) consists of 100 classes of photographs of various objects. In the split CIFAR 100 benchmark, the CIFAR 100 is split into ten disjoint datasets of consecutive classes i.e. $\{\{0-9\},\{10-19\},\{20-29\},\{30-39\},\{40-49\},\{50-59\},\{60-69\},\{70-79\},\{80-89\},\{90-99\}\}$. These ten subsets serve as a sequence of tasks to learn consecutively.

Single versus Multi-Head Evaluation The primary difference between single-head and multi-head evaluations is that at test time, the task identifier in multi-head it is known, while in single-head it is unknown (Chaudhry et al., 2018). For example, consider the split MNIST benchmark at the 5th and final task in the sequence. During training on this task, both evaluation methodologies only get to see data from the classes in this task (i.e. $\{8,9\}$). At test time, in multi-head evaluation, the goal is to predict a class out of only these two labels (i.e. $\{8,9\}$). However at test time in single-head evaluation, the goal is to predict a label out of all ten classes (i.e. $\{0,...,9\}$) that the model has seen thus far, even though the training data for this task included only classes labeled $\{8,9\}$ and there is no access to training data from earlier classes. As a result, single-head evaluation is considerably more challenging that multi-head evaluation.

Continual Learning Evaluation Procedure For all of the continual learning experiments, we use a CNAPS model with FiLM adaptation that has been meta-trained on the eight training datasets from META-DATASET. For each benchmark of 30 epochs, the context set consists of 1, 10, and 100 labeled shots per class drawn from the training split of the dataset, and the target set consists of 1000 images per class drawn from the test split of the dataset for which predictions of the label are made. The context and target sets are then passed into the meta-trained model and the average classification accuracy of all the test examples over all the epochs is computed.

B.2.3 Active Learning Experimentation Details

Active Learning Datasets We use the test split of the Omniglot (Lake et al., 2011) and VGG Flower (Nilsback and Zisserman, 2008) datasets for the active learning experiments. Note that these datasets are members of the META-DATASET benchmark and the learners evaluated in the active learning experiments have been meta-trained on the training splits of these datasets.

Each active learning dataset is split into three parts: the first *labeled* set contains 5 labeled examples per class, the second *test* set contains 5 labeled images per class when evaluating with Omniglot and 20 labeled images per class when evaluating with VGG Flower, and the third *pool* set contains the remaining unlabeled images.

Active Learning Evaluation Procedure For all of the active learning experiments, we use a CNAPS model with FiLM adaptation and a prototypical networks model that have been meta-trained on the eight training datasets from META-DATASET. We perform 30 active learning iterations to acquire 30 new images to be labeled by the *oracle* that will be added to the labeled set of images (refer to Section 2.6). During each iteration, we: (i) compute the classification accuracy on the test set using the meta-trained model with the labeled set as the context set (this is the iteration accuracy reported in all of the plots); (ii) use the meta-trained model to compute the predictive distribution using the currently labeled set of images as the context set and all of the unlabeled images in the pool set as the target inputs; (iii) choose the best candidate to label by computing the acquisition function using each of the predictions. (iv) move the best candidate from the pool set to the labeled set of images for use in the next iteration. The above is repeated for both CNAPS and prototypical networks for each of the acquisition functions evaluated (Predictive Entropy, Variation Ratios, and Random (Gal et al., 2017)) on both Omniglot and VGG Flower datasets.

B.3 TaskNorm Experimentation Details

In this section, we provide the experimental details required to reproduce our experiments. The experiments using MAML (Finn et al., 2017) were implemented in TensorFlow (Abadi et al., 2015), the Prototypical Networks experiments were implemented in Pytorch (Paszke et al., 2019), and the experiments using CNAPs (Requeima et al., 2019b) were implemented using a combination of TensorFlow (Abadi et al., 2015) and Pytorch. All experiments were executed on NVIDIA Tesla P100-16GB GPUs.

B.3.1 MAML Experiments

We evaluate MAML using a range of normalization layers on:

1. Omniglot (Lake et al., 2011): a few-shot learning dataset consisting of 1623 handwritten characters (each with 20 instances) derived from 50 alphabets.
2. *mini*ImageNet (Vinyals et al., 2016): a dataset of 60,000 color images that is sub-divided into 100 classes, each with 600 instances.

For all the MAML experiments, we used the codebase provided by the MAML authors (Finn, 2017) with only small modifications to enable additional normalization techniques. Note that we used the first-order approximation version of MAML for all experiments. MAML was invoked with the command lines as specified in the `main.py` file in the MAML codebase. No hyper-parameter tuning was performed and we took the results from a single run. All models were trained for 60,000 iterations and then tested. No early stopping was used. We did not select the model based on validation accuracy or other criteria. The MAML code employs ten gradient steps at test time and computes classification accuracy after each step. We report the maximum accuracy across those ten steps. To generate the plot in Figure 5.3, we use the same command line as Omniglot-5-1, but vary the update batch size from one to ten.

B.3.2 CNAPs Experiments

We evaluate CNAPs using a range of normalization layers on a demanding few-shot classification challenge called Meta-Dataset (Triantafillou et al., 2020). Meta-Dataset is composed of ten (eight train, two test) image classification datasets. We augment Meta-Dataset with three additional held-out datasets: MNIST (LeCun et al., 2010), CIFAR10 (Krizhevsky and Hinton, 2009), and CIFAR100 (Krizhevsky and Hinton, 2009). The challenge constructs few-shot learning tasks by drawing from the following distribution. First, one of the datasets is sampled uniformly; second, the "way" and "shot" are sampled randomly according to a fixed procedure; third, the classes and context / target instances are sampled. Where a hierarchical structure exists in the data (ILSVRC or Omniglot), task-sampling respects the hierarchy. In the meta-test phase, the identity of the original dataset is not revealed and the tasks must be treated independently (i.e. no information can be transferred between them). Notably, the meta-training set comprises a disjoint and dissimilar set of classes from those used for meta-test. Full details are available in Triantafillou et al. (2020).

For all the CNAPS experiments, we use the code provided by the the CNAPS authors (Requeima et al., 2019a) with only small modifications to enable additional normalization techniques. We follow an identical dataset configuration and training process as prescribed in Requeima et al. (2019a). To generate results in Table 5.3, we used the following CNAPS options: FiLM feature adaptation, a learning rate of 0.001, and TBN, CBN, BRN, and RN used 70,000 training iterations, IN used 200,000 iterations, LN used 110,000 iterations, and TASKNORM used 60,000 iterations. The CNAPS code generates two models: fully trained and best validation. We report the better of the two. We performed no hyper-parameter tuning and report the test results from the first run. Note that CBN, TBN, and RN share the same trained model. They differ only in how meta-testing is done.

B.3.3 Prototypical Networks Experiments

We evaluate the Prototypical Networks (Snell et al., 2017) algorithm with a range of NLs using the same Omniglot, *mini*ImageNet, and META-DATASET benchmarks.

For Omniglot, we used the codebase created by the Prototypical Networks authors (Snell, 2017). For *mini*ImageNet, we used the a different codebase ((Chen, 2018)) as the first codebase did not support *mini*ImageNet. Only small modifications were made to the two codebases to enable additional NLs. For Omniglot and *mini*ImageNet, we set hyper-parameters as prescribed in (Snell et al., 2017). Early stopping was employed and the model that produced the best validation was used for testing.

For META-DATASET, we use the code provided by the the CNAPS authors (Requeima et al., 2019a) with only small modifications to enable additional normalization techniques and a new classifier adaptation layer to generate the linear classifier weights per equation (8) in (Snell et al., 2017). We follow an identical dataset configuration and training process as prescribed in Requeima et al. (2019a). To generate results in Table 5.4, we used the following CNAPS options: FiLM feature adaptation, a learning rate of 0.001, 60,000 training iterations for all NLs, and the pretrained feature extractor weights were not frozen and allowed to update during meta-training.

Appendix C

Network Architectures

C.1 VERSA Few-shot Classification Network Architectures

Tables C.1 to C.3 and Section C.1 detail the neural network architectures for the feature extractor θ, amortization network ϕ, and linear classifier ψ, respectively. The feature extraction network is very similar to that used in (Vinyals et al., 2016). The output of the amortization network yields mean-field Gaussian parameters for the weight distributions of the linear classifier ψ. When sampling from the weight distributions, we employ the local-reparameterization trick (Kingma et al., 2015), that is we sample from the implied distribution over the logits rather than directly from the variational distribution. To reduce the number of learned parameters, we share the feature extraction network θ with the pre-processing phase of the amortizaion network ψ.

Omniglot Shared Feature Extraction Network ($\boldsymbol{\theta}$): $\boldsymbol{x}^* \rightarrow h_{\boldsymbol{\theta}}(\boldsymbol{x}^*)$

Output size	**Layers**
$28 \times 28 \times 1$	Input image
$14 \times 14 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$7 \times 7 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$4 \times 4 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, SAME)
256	flatten

Table C.1 Feature extraction network used for Omniglot few-shot learning. Batch Normalization and dropout with a keep probability of 0.9 used throughout.

***mini*ImageNet Shared Feature Extraction Network ($\boldsymbol{\theta}$): $\boldsymbol{x}^* \rightarrow h_{\boldsymbol{\theta}}(\boldsymbol{x}^*)$**

Output size	Layers
$84 \times 84 \times 1$	Input image
$42 \times 42 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$21 \times 21 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$10 \times 10 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$5 \times 5 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), dropout, pool (2×2, stride 2, VALID)
256	flatten

Table C.2 Feature extraction network used for *mini*ImageNet few-shot learning. Batch Normalization and dropout with a keep probability of 0.5 used throughout.

Amortization Network (ϕ): $\boldsymbol{x}_1^c, ..., \boldsymbol{x}_{k_c}^c \rightarrow \mu_{\boldsymbol{w}^{(c)}}, \sigma^2_{\boldsymbol{w}^{(c)}}$

Phase	Output size	Layers
feature extraction	$k \times 256$	shared feature network ($\boldsymbol{\theta}$)
instance pooling	256	mean
ψ weight distribution	256	$2 \times$ fully connected, ELU + linear fully connected to $\mu_{\boldsymbol{w}^{(c)}}, \sigma^2_{\boldsymbol{w}^{(c)}}$

Table C.3 Amortization network used for Omniglot and *mini*ImageNet few-shot learning.

C.2 VERSA View Reconstruction Network Architectures

C.3 CNAPs Network Architectures

C.3.1 ResNet18 Architecture details

Throughout our experiments in Section 4.5, we use a ResNet18 (He et al., 2016) as our feature extractor, the parameters of which we denote $\boldsymbol{\theta}$. Table C.8 and Table C.9 detail the architectures of the basic block (left) and basic scaling block (right) that are the fundamental components of the ResNet that we employ. Table C.10 details how these blocks are composed to generate the overall feature extractor network. We use the implementation that is provided by the PyTorch (Paszke et al., 2019)[1], though we adapt the code to enable the use of FiLM layers.

C.3.2 Adaptation Network Architecture Details

In this section, we provide the details of the architectures used for our adaptation networks. Table C.11 details the architecture of the set encoder $g : D^\tau \mapsto \boldsymbol{z}_G$ that maps context sets to global representations.

[1]https://pytorch.org/docs/stable/torchvision/models.html

Linear Classifier (ψ): $h_{\boldsymbol{\theta}}(\boldsymbol{x}^*) \rightarrow p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{\psi}_\tau)$

Output size	Layers
256	Input features
C	fully connected, softmax

Table C.4 Linear classifier used for Omniglot and *mini*ImageNet few-shot learning.

ShapeNet Encoder Network (ϕ): $\boldsymbol{y} \rightarrow h$

Output size	Layers
$32 \times 32 \times 1$	Input image
$16 \times 16 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$8 \times 8 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$4 \times 4 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
$2 \times 2 \times 64$	conv2d (3×3, stride 1, SAME, RELU), pool (2×2, stride 2, VALID)
d_ϕ	fully connected, RELU

Table C.5 Encoder network used for ShapeNet few-shot learning. No dropout or batch normalization is used.

ShapeNet Amortization Network (ϕ): $\boldsymbol{x}_1^\tau, ..., \boldsymbol{x}_k^\tau, y_1^\tau, ..., y_k^\tau \rightarrow \mu_\psi, \sigma_\psi^2$

Phase	Output size	Layers
ϕ_{pre}	$k \times d_\phi$	encoder network (ϕ)
concatenate h and $\boldsymbol{x}$	$k \times (d_\psi + d_x)$	concat(h, $\boldsymbol{x}$)
ϕ_{mid}	$k \times d_\phi$	2×2 fully connected, ELU
instance pooling	$1 \times d_\phi$	average
ϕ_{post}	$1 \times d_\phi$	$2\times$ fully connected, ELU
ψ distribution	d_ψ	fully connected linear layers to μ_ψ, σ_ψ^2

Table C.6 Amortization network used for ShapeNet few-shot learning.

Table C.12 details the architecture used in the auto-regressive parameterization of $\boldsymbol{z}_{\text{AR}}$. In our experiments, there is one such network for every block in the ResNet18 (detailed in Table C.10). These networks accept as input the set of activations from the previous block, and map them (through the permutation invariant structure) to a vector representation of the output of the layer. The representation $\boldsymbol{z}_i = (\boldsymbol{z}_{\text{G}}, \boldsymbol{z}_{\text{AR}})$ is then generated by concatenating the global and auto-regressive representations, and fed into the adaptation network that provides the FiLM layer parameters for the next layer. This network is detailed in Table C.13, and illustrated in Figure 4.4. Note that, as depicted in Figure 4.4, each layer has four networks with architectures as detailed in Table C.13, one for each $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, for each convolutional layer in the block.

ShapeNet Generator Network (θ): $\tilde{x} \to p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau)$	
Output size	**Layers**
$d_\psi + d_{\boldsymbol{x}}$	concat($\boldsymbol{\psi}$, $\boldsymbol{x}$)
512	fully connected, RELU
1024	fully connected, RELU
$2 \times 2 \times 256$	reshape
$4 \times 4 \times 128$	deconv2d (3×3, stride 2, SAME, RELU)
$8 \times 8 \times 64$	deconv2d (3×3, stride 2, SAME, RELU)
$16 \times 16 \times 32$	deconv2d (3×3, stride 2, SAME, RELU)
$32 \times 32 \times 1$	deconv2d (3×3, stride 2, SAME, sigmoid)

Table C.7 Generator network used for ShapeNet few-shot learning. No dropout or batch normalization is used.

Layers
Input
Conv2d (3×3, stride 1, pad 1)
BatchNorm
FiLM ($\boldsymbol{\gamma}_{b,1}, \boldsymbol{\beta}_{b,1}$)
ReLU
Conv2d (3×3, stride 1, pad 1)
BatchNorm
FiLM ($\boldsymbol{\gamma}_{b,2}, \boldsymbol{\beta}_{b,2}$)
Sum with Input
ReLU

Table C.8 ResNet-18 basic block b.

Layers
Input
Conv2d (3×3, stride 2, pad 1)
BatchNorm
FiLM ($\boldsymbol{\gamma}_{b,1}, \boldsymbol{\beta}_{b,1}$)
ReLU
Conv2d (3×3, stride 1, pad 1)
BatchNorm
FiLM ($\boldsymbol{\gamma}_{b,2}, \boldsymbol{\beta}_{b,2}$)
Downsample Input by factor of 2
Sum with Downsampled Input
ReLU

Table C.9 ResNet-18 basic scaling block b.

C.3.3 Linear Classifier Adaptation Network

Finally, in this section we give the details for the linear classifer $\boldsymbol{\psi}_w^\tau$, and the adaptation network that provides these task-specific parameters $\boldsymbol{\psi}_w(\cdot)$. The adaptation network accepts a class-specific representation that is generated by applying a mean-pooling operation to the adapted feature activations of each instance associated with the class in the context set: $\boldsymbol{z}_c^\tau = \frac{1}{N_c^\tau} \sum\limits_{\boldsymbol{x} \in D_c^\tau} f_{\boldsymbol{\theta}}(\boldsymbol{x}; \boldsymbol{\psi}_f^\tau)$, where N_c^τ denotes the number of context instances associated with class c in task τ. $\boldsymbol{\psi}_w$ is comprised of two separate networks (one for the weights $\boldsymbol{\psi}_w$ and one for the biases $\boldsymbol{\psi}_b$) detailed in Table C.14 and Table C.15. The resulting weights and biases (for each class in task τ) can then be used as a linear classification layer, as detailed in Table C.16.

ResNet-18 Feature Extractor (θ) with FiLM Layers: $\boldsymbol{x} \rightarrow f_{\boldsymbol{\theta}}(\boldsymbol{x}; \boldsymbol{\psi}_f^{\tau})$, $\boldsymbol{x}^* \rightarrow f_{\boldsymbol{\theta}}(\boldsymbol{x}^*; \boldsymbol{\psi}_f^{\tau})$

Stage	Output size	Layers
Input	$84 \times 84 \times 3$	Input image
Pre-processing	$41 \times 41 \times 64$	Conv2d (5×5, stride 2, pad 1, BatchNorm, ReLU)
Layer 1	$41 \times 41 \times 64$	Basic Block $\times$ 2
Layer 2	$21 \times 21 \times 128$	Basic Block, Basic Scaling Block
Layer 3	$11 \times 11 \times 256$	Basic Block, Basic Scaling Block
Layer 4	$6 \times 6 \times 512$	Basic Block, Basic Scaling Block
Post-Processing	512	AvgPool, Flatten

Table C.10 ResNet-18 feature extractor network.

Set Encoder (g): $\boldsymbol{x} \rightarrow \boldsymbol{z}_G^{\tau}$

Output size	Layers
$84 \times 84 \times 3$	Input image
$42 \times 42 \times 64$	Conv2d (3×3, stride 1, pad 1, ReLU), MaxPool (2×2, stride 2)
$21 \times 21 \times 64$	Conv2d (3×3, stride 1, pad 1, ReLU), MaxPool (2×2, stride 2)
$10 \times 10 \times 64$	Conv2d (3×3, stride 1, pad 1, ReLU), MaxPool (2×2, stride 2)
$5 \times 5 \times 64$	Conv2d (3×3, stride 1, pad 1, ReLU), MaxPool (2×2, stride 2)
$2 \times 2 \times 64$	Conv2d (3×3, stride 1, pad 1, ReLU), MaxPool (2×2, stride 2)
64	AdaptiveAvgPool2d

Table C.11 Set encoder g.

Set Encoder ($\boldsymbol{\phi}_f$): $\{f_{\boldsymbol{\theta}}^{l_i}(\boldsymbol{x}; \boldsymbol{\psi}_f^{\tau})\} \rightarrow \boldsymbol{z}_{\text{AR}}^i$

Output size	Layers
l_i channels $\times$ l_i channel size	Input $\{f_{\boldsymbol{\theta}}^{l_i}(x; \boldsymbol{\psi}_f^{\tau})\}$
l_i channels $\times$ l_i channel size	AvgPool, Flatten
l_i channels	fully connected, ReLU
l_i channels	2 $\times$ fully connected with residual skip connection, ReLU
l_i channels	fully connected with residual skip connection
l_i channels	mean pooling over instances
l_i channels	Input from mean pooling
l_i channels	fully connected, ReLU

Table C.12 Network of set encoder $\boldsymbol{\phi}_f$.

Network (ϕ_f): $(\boldsymbol{z}_\text{G}, \boldsymbol{z}_\text{AR}) \rightarrow (\boldsymbol{\gamma}, \boldsymbol{\beta})$

Output size	Layers
$64 + l_i$ channels	Input from Concatenate
l_i channels	fully connected, ReLU
l_i channels	2 × fully connected with residual skip connection, ReLU
l_i channels	fully connected with residual skip connection

Table C.13 Network ϕ_f.

Network ($\boldsymbol{\phi}_w$):
$\boldsymbol{z}_c \rightarrow \boldsymbol{\psi}_{w,w}$

Output size	Layers
512	Input from mean pooling
512	2 × fully connected, ELU
512	fully connected
512	Sum with Input

Table C.14 Network $\boldsymbol{\phi}_w$.

Network ($\boldsymbol{\phi}_b$):
$\boldsymbol{z}_c \rightarrow \boldsymbol{\psi}_{w,b}$

Output size	Layers
512	Input from mean pooling
512	2 × fully connected, ELU
1	fully connected

Table C.15 Network $\boldsymbol{\phi}_b$.

Linear Classifier ($\boldsymbol{\psi}_w$): $f_{\boldsymbol{\theta}}(\boldsymbol{x}^*; \boldsymbol{\psi}_f^\tau) \rightarrow p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\psi}^\tau(D^\tau), \boldsymbol{\theta})$

Output size	Layers
512	Input features $f_{\boldsymbol{\theta}}(\boldsymbol{x}^*; \boldsymbol{\psi}_f^\tau)$
$512 \times C^\tau$	Input weights w
512×1	Input biases b
C^τ	fully connected
C^τ	softmax

Table C.16 Linear classifier network.