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

James Requeima[*]
University of Cambridge
Invenia Labs
jrr41@cam.ac.uk

Jonathan Gordon[*]
University of Cambridge
jg801@cam.ac.uk

John Bronskill[*]
University of Cambridge
jfb54@cam.ac.uk

Sebastian Nowozin
Google Research Berlin
nowozin@google.com

Richard E. Turner
University of Cambridge
Microsoft Research
ret26@cam.ac.uk

Abstract

The goal of this paper is to design image classification systems that, after an initial multi-task training phase, can automatically adapt to new tasks encountered at test time. We introduce a conditional neural process based approach to the multi-task classification setting for this purpose, 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 achieves state-of-the-art results 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.

1 Introduction

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 [1, 2] and few-shot learning [3]. 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. (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 [4, 5]. While useful in simple settings, this approach is prone to under-fitting when the task distribution is heterogeneous [6]. Alternatively, we can adapt all parameters in the feature extractor [7, 8] thereby increasing

[*]Authors contributed equally

Preprint. Under review.

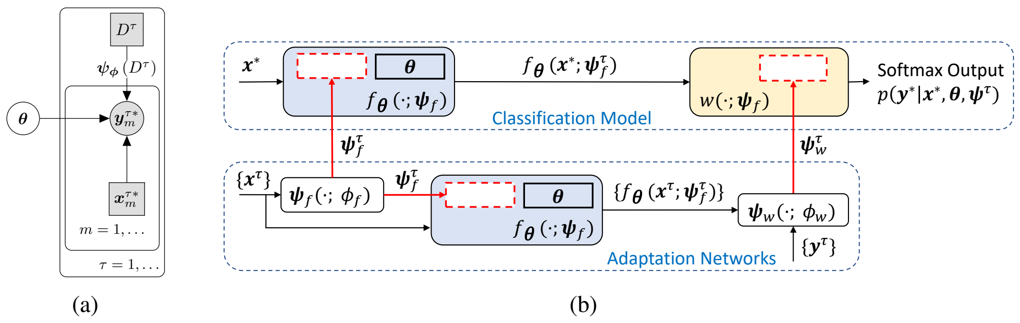


Figure 1: (a) Probabilistic graphical model detailing CNP [13] framework. (b) Computational diagram depicting the CNAPS model class. Red boxes imply parameters in the model architecture supplied by adaptation networks. Blue shaded boxes depict the feature extractor and the gold box depicts the linear classifier.

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 [7, 9]. 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*) [5, 10]. 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).[2] CNAPS directly model the desired predictive distribution [11, 12], thereby introducing a *conditional neural processes* (CNPs) [13] approach to the multi-task classification setting. 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 [6]. In Section 5 we evaluate CNAPS. Recently, Triantafillou et al. [6] 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 achieve 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.

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 [13] construct predictive distributions given $\boldsymbol{x}^*$ as:

$$p\left(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta}, D^\tau\right) = p\left(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau\right) = p\left(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right)\right). \tag{1}$$

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 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

[2]Full source code for CNAPS will be made available upon publication.

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 1b.

2.1 Specification of the classifier: global θ and task-specific parameters ψ^τ

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 [14]. In what follows, we assume that the global parameters $\boldsymbol{\theta}$ are fixed and known. In Section 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 [9]) gives flexibility, but it is also slow and prone to over-fitting [6]. Instead, we employ linear modulation of the convolutional feature maps as proposed by Perez et al. [15], which adapts the feature extractor through a relatively small number of task specific parameters.

A Feature-wise Linear Modulation (FiLM) layer [15] scales and shifts the i^{th} unadapted feature map $\boldsymbol{f}_i$ in the feature extractor 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. Figure E.9a illustrates a FiLM layer operating on a convolutional layer, and Figure E.9b illustrates how a FiLM layer can be added to a standard Residual network block [14]. A key advantage of FiLM layers is that they enable expressive feature adaptation while adding only a small number of parameters [15]. 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, they allow the model to adapt to a broad class of datasets.

2.2 Computing the local parameters via adaptation networks

The previous sections have specified the form of the classifier $p\left(\boldsymbol{y}^* | \boldsymbol{x}^*, \boldsymbol{\theta}, \boldsymbol{\psi}^\tau\right)$ 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., [16, 17]), 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 generalisation, 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.

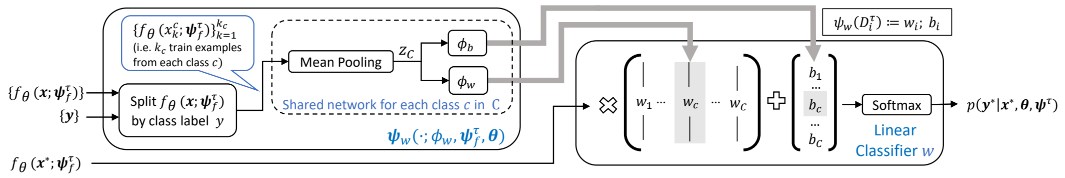

$\{f_\theta(x_k^c; \boldsymbol{\psi}_f^\tau)\}_{k=1}^{k_c}$
(i.e. k_c train examples from each class c)
$\{f_\theta(\boldsymbol{x}; \boldsymbol{\psi}_f^\tau)\}$
$\{\boldsymbol{y}\}$
Split $f_\theta(\boldsymbol{x}; \boldsymbol{\psi}_f^\tau)$ by class label y
Mean Pooling
z_C
ϕ_b
ϕ_w
Shared network for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

Figure 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: 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 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. [5]. First, each column of the weight matrix is generated independently from the context points from that class $\boldsymbol{\psi}_w^\tau = [\boldsymbol{\psi}_w(D_1^\tau), \quad \ldots, \quad \boldsymbol{\psi}_w(D_C^\tau)]$, an approach which scales to arbitrary numbers of classes. Second, we employ a permutation invariant architecture [18, 19] for $\boldsymbol{\psi}_w(\cdot)$ to handle the variable input cardinality (see Appendix E for details). Third, as permutation invariant architectures can be incrementally updated [20], continual learning is supported (as discussed in Section 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.,

$$\boldsymbol{\psi}_w(D_c^\tau; \boldsymbol{\phi}_w, \boldsymbol{\psi}_f, \boldsymbol{\theta}) = \boldsymbol{\psi}_w(\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_m; \boldsymbol{\psi}_f) | \boldsymbol{x}_m \in D^\tau, \boldsymbol{y}_m = c\}; \boldsymbol{\phi}_w). \tag{2}$$

Here $\boldsymbol{\phi}_w$ are learnable parameters of $\boldsymbol{\psi}_w(\cdot)$. See Figure 2 for an illustration.

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 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 $\boldsymbol{\phi}_f$.

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 [18, 19] (see Appendix E for details). The full parameterization is illustrated in Figure 3, and the architecture of $\boldsymbol{\psi}_f^i(\cdot)$ networks is illustrated in Figure E.10.

3 Model Training

The previous section has specified the model (see Figure 1b for a schematic). We now describe how to train the global classifier parameters $\boldsymbol{\theta}$ and the adaptation network parameters $\boldsymbol{\phi} = \{\boldsymbol{\phi}_f, \boldsymbol{\phi}_w\}$.

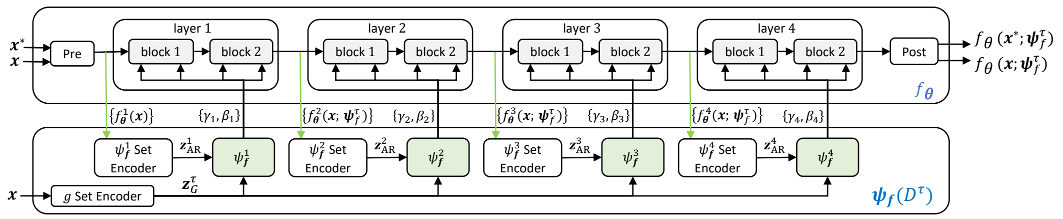


Figure 3: Implementation of the feature-extractor: an independently learned set encoder g provides a fixed context that is concatenated to the (processed) activations of $\boldsymbol{x}$ from the previous ResNet block. The inputs $\boldsymbol{z}_i = (\boldsymbol{z}_{\text{G}}^\tau, \boldsymbol{z}_{\text{AR}}^i)$ are then fed to $\boldsymbol{\psi}_f^i(\cdot)$, which outputs the FiLM parameters for layer i.

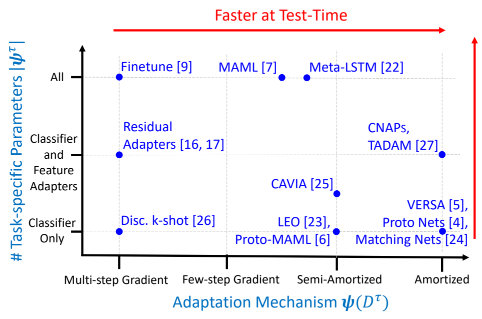


Figure 4: Model design space. The y-axis represents the number of task-specific parameters $|\boldsymbol{\psi}^\tau|$. Increasing $|\boldsymbol{\psi}^\tau|$ increases model flexibility, but also the propensity to over-fit. The x-axis represents the complexity of the mechanism used to adapt the task-specific parameters to training data $\boldsymbol{\psi}(D^\tau)$. 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.

Training the global classifier parameters $\boldsymbol{\theta}$. A natural approach to training the model (originally employed by CNPs [13]) would be to maximize the likelihood of the training data jointly over $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. However, experiments (detailed in Appendix D.3) 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 [21, 6]) 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 generalisation. (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 desireable as it results in a dramatic decrease in training time.

Training the adaptation network parameters $\boldsymbol{\phi}$. Following the work of Garnelo et al. [13], we train $\boldsymbol{\phi}$ with maximum likelihood. An unbiased stochastic estimator of the log-likelihood is:

$$\hat{\mathcal{L}}(\boldsymbol{\phi}) = \frac{1}{MT} \sum_{m,\tau} \log p\left(\boldsymbol{y}_m^{*\tau} | \boldsymbol{x}_m^{*\tau}, \boldsymbol{\psi}_{\boldsymbol{\phi}}\left(D^\tau\right), \boldsymbol{\theta}\right), \tag{3}$$

where $\{\boldsymbol{y}_m^{*\tau}, \boldsymbol{x}_m^{*\tau}, D^\tau\} \sim \hat{P}$, with $\hat{P}$ representing the data distribution (e.g., sampling tasks and splitting them into disjoint context (D^τ) and target data $\{(\boldsymbol{x}_m^{*\tau}, \boldsymbol{y}_m^{*\tau})\}_{m=1}^{M_t}$). 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. [6] and META-DATASET for this sampling procedure. Algorithm A.1 details computation of the stochastic estimator for a single task.

4 Related Work

Our work frames multi-task classification as directly modelling the predictive distribution $p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\psi}(D^\tau))$. The perspective allows previous work [7, 5, 15, 22, 16, 17, 23, 4, 6, 24, 9, 25, 26] 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. This space is illustrated in Figure 4, and further elaborated upon in Appendix B.

One of the inspirations for our work is conditional neural processes (CNPs) [13]. CNPs directly model the predictive distribution $p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{\psi}(D^\tau))$ 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 [5] which employs a distributional treatment of the local-parameters $\boldsymbol{\psi}$.

A model with design choices closely related to CNAPS is TADAM [27]. 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 5).

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 Appendix E for full details). While CNAPS can utilize any feature extractor, a ResNet18 [14] is used throughout to enable fair comparison with Triantafillou et al. [6]. To ensure that each task is handled independently, batch normalization statistics [28] 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 [6]. 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 details are available in Appendix C.1 and [6].

Triantafillou et al. [6] 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 Appendix D.1. 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. [6], including their proposed method (Proto-MAML) in Table 1. We meta-test CNAPS on three additional held-out datasets: MNIST [29], CIFAR10 [30], and CIFAR100 [30]. 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 1, we distinguish between two types of generalization: (i) unseen tasks (classes) in meta-training datasets, and (ii) unseen datasets.

Unseen tasks: CNAPS achieve significant improvements over existing methods on all 8 datasets (with the exception of fine-tuning on QUICKDRAW, which achieves comparable performance on this dataset, but markedly poorer performance for all others). The ablation study demonstrates that removing $\boldsymbol{z}_{\text{AR}}$ from the feature extractor adaptation consistently degrades accuracy, and that removing all feature extractor adaptation results in drastic reductions in accuracy.

Dataset	Finetune	MatchingNet	ProtoNet	fo-MAML	Proto-MAML	CNAPS (no $\boldsymbol{\psi}_f$)	CNAPS (no $\boldsymbol{z}_{AR}$)	CNAPS
ILSVRC [21]	39.7 ± 1.0	40.8 ± 1.0	41.8 ± 1.1	22.4 ± 0.8	45.5 ± 1.0	42.6 ± 1.4	46.7 ± 1.0	**49.5 ± 1.0**
Omniglot [31]	85.6 ± 0.9	75.6 ± 1.1	78.6 ± 1.1	68.1 ± 1.4	86.3 ± 0.9	55.5 ± 1.3	**89.0 ± 0.6**	**89.7 ± 0.5**
Aircraft [32]	69.8 ± 0.9	60.7 ± 0.9	66.6 ± 0.9	44.5 ± 0.9	79.2 ± 0.7	63.9 ± 0.9	85.1 ± 0.6	**87.2 ± 0.5**
Birds [33]	54.1 ± 1.1	57.1 ± 0.9	63.6 ± 1.0	36.7 ± 1.1	72.7 ± 1.0	50.5 ± 1.1	72.2 ± 1.0	**76.7 ± 0.9**
Textures [34]	62.7 ± 0.8	64.7 ± 0.8	66.6 ± 0.8	45.8 ± 0.7	66.7 ± 0.8	77.7 ± 0.6	74.8 ± 0.6	**83.0 ± 0.6**
Quick Draw [35]	**73.9 ± 0.8**	58.9 ± 1.0	63.6 ± 0.9	41.3 ± 1.5	67.8 ± 0.9	54.7 ± 1.1	**72.5 ± 0.8**	**72.3 ± 0.8**
Fungi [36]	31.9 ± 1.1	34.4 ± 1.0	38.0 ± 1.1	14.2 ± 0.8	44.6 ± 1.2	30.9 ± 1.1	**48.9 ± 1.1**	**50.5 ± 1.1**
VGG Flower [37]	77.6 ± 0.9	82.6 ± 0.7	84.4 ± 0.7	61.1 ± 1.1	88.2 ± 0.7	82.2 ± 0.7	88.0 ± 0.5	**92.5 ± 0.4**
Traffic Signs [38]	53.1 ± 1.1	**57.9 ± 1.2**	50.6 ± 1.0	24.0 ± 1.1	46.4 ± 1.0	46.2 ± 1.1	46.1 ± 1.1	48.4 ± 1.1
MSCOCO [39]	27.7 ± 1.2	30.2 ± 1.1	37.6 ± 1.1	13.6 ± 1.0	35.1 ± 1.2	36.2 ± 1.0	36.1 ± 1.0	**39.7 ± 0.9**
MNIST [29]			74.3 ± 0.8*			75.9 ± 0.8	91.5 ± 0.4	**92.1 ± 0.4**
CIFAR10 [30]			66.4 ± 0.7*			60.0 ± 0.8	**70.2 ± 0.8**	**69.9 ± 0.9**
CIFAR100 [30]			**54.7 ± 1.1***			48.5 ± 1.1	**56.6 ± 1.0**	53.2 ± 0.7

Table 1: Few-shot classification results on META-DATASET [6] using models trained on all training datasets. All figures are percentages and the ± 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 [6], except where $*$ denotes our implementation.

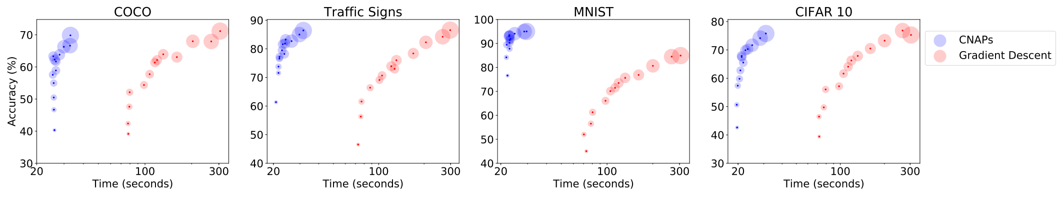


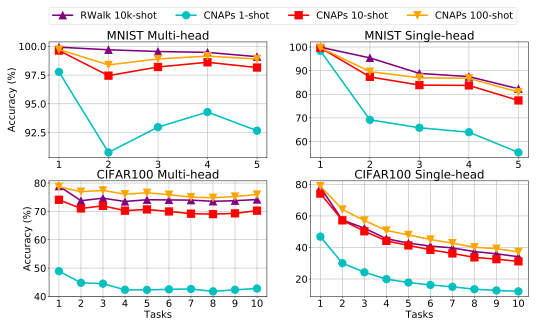
Figure 5: Comparing CNAPS to gradient based feature extractor adaptation: accuracy on 5-way classification tasks from withheld datasets as a function of processing time. Dot size reflects shot number (1 to 25 shots).

Unseen datasets: CNAPS-models achieve the best performance on four of the five datasets. Other than for CIFAR100, removing $\boldsymbol{z}_{\text{AR}}$ from the feature extractor adaptation either decreases accuracy or has no effect. Again, removing 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.

Additional results: Results when meta-training only on the META-DATASET version of ILSVRC are given in Table D.3. In Appendix D.2, we visualize the task encodings and parameters, demonstrating 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. CNAPS generate FiLM layer parameters for each task τ at test time using the adaptation network $\boldsymbol{\psi}_f(D_\tau)$. It is also possible to learn the FiLM parameters via gradient descent (see [16, 17]). Here we compare CNAPS to this approach. Figure 5 shows plots of 5-way classification accuracy versus time for four held out data sets as the number of shots was varied. For gradient 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.

Complex Learning Scenarios: Continual Learning. In continual learning [40] 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. Although CNAPS has not been explicitly trained for continual learning, we 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 [41] and Split CIFAR100 [42]. We modify the model to compute running averages for the representations of both $\boldsymbol{\psi}_w^\tau$ and $\boldsymbol{\psi}_f^\tau$ (see Appendix F for further details), in this way



	MNIST		*CIFAR100*	
Method	Multi	Single	Multi	Single
SI [41]	99.3	57.6	73.2	22.8
EWC [43]	99.3	55.8	72.8	23.1
VCL [44]	98.5 $\pm$ 0.4	-	-	-
RWalk [42]	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

Figure 6: 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 [42, 45].

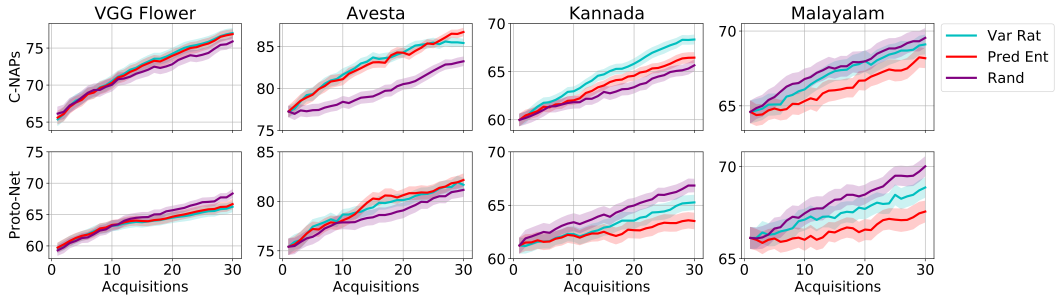


Figure 7: Accuracy vs active learning iterations for held-out classes / languages. (Top) CNAPS and (bottom) prototypical networks. Error shading is one standard error. CNAPS achieves better accuracy than prototypical networks and improvements over random acquisition, whereas prototypical networks do not.

it performs incremental updates using the new data and old model, and does not need to access old data. Figure 6 (left) shows the accumulated multi- and single-head [42] test accuracy averaged over 30 runs (further results and more detailed figures are in Appendix G). Figure 6 (right) shows average results at the final task comparing to SI [41], EWC [43], VCL [44], and Riemannian Walk [42].

Figure 6 demonstrates that CNAPS naturally resists catastrophic forgetting [43] 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, 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.

Complex Learning Scenarios: Active Learning. Active learning [46, 47] requires accurate data-efficient learning that returns well-calibrated uncertainty estimates. Figure 7 compares the performance of CNAPS and prototypical networks using two standard active learning acquisition functions (variation ratios and predictive entropy [46]) against random acquisition on the FLOWERS dataset and three representative held-out languages from OMNIGLOT (performance on all languages is presented in Appendix H). Figure 7 and Appendix H 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.

6 Conclusions

This paper has introduced CNAPS, an automatic, fast and flexible modelling approach for multi-task classification. We have demonstrated that 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 distributional extensions to CNAPS [48, 49].

References

[1] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning*. PhD thesis, Technische Universität München, 1987.

[2] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[3] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[4] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4080–4090, 2017.

[5] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HkxStoC5F7.

[6] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[7] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

[8] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[9] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[10] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan Yuille. Few-shot image recognition by predicting parameters from activations. *arXiv preprint arXiv:1706.03466*, 2017.

[11] Seymour Geisser. On the prediction of observables: a selective update. Technical report, University of Minnesota, 1983.

[12] Seymour Geisser. *Predictive inference*. Routledge, 2017.

[13] Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Eslami. Conditional neural processes. *arXiv preprint arXiv:1807.01613*, 2018.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, pages 506–516, 2017.

[17] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127, 2018.

[18] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404, 2017.

[19] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[20] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6904–6914. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7266-a-meta-learning-perspective-on-cold-start-recommendations-for-items.pdf`.

[21] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[22] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[23] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[24] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[25] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. CAML: Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

[26] Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartłomiej Świątkowski, Bernhard Schölkopf, and Richard E Turner. Discriminative k-shot learning using probabilistic models. *arXiv preprint arXiv:1706.00326*, 2017.

[27] Boris N Oreshkin, Alexandre Lacoste, and Pau Rodriguez. TADAM: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*, 2018.

[28] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[29] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18, 2010.

[30] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[31] Brenden Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua Tenenbaum. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 33, 2011.

[32] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[33] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[34] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014.

[35] David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

[36] Brigit Schroeder and Yin Cui. Fgvcx fungi classification challenge at fgvc5. `https://www.kaggle.com/c/fungi-challenge-fgvc-2018`, 2018.

[37] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[38] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. 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, 2013.

[39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

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

[41] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3987–3995. JMLR. org, 2017.

[42] Arslan Chaudhry, Puneet K Dokania, Thalaiyasingam Ajanthan, and Philip HS Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[43] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[44] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[45] Siddharth Swaroop, Cuong V Nguyen, Thang D Bui, and Richard E Turner. Improving and understanding variational continual learning. *arXiv preprint arXiv:1905.02099*, 2019.

[46] David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of artificial intelligence research*, 4:129–145, 1996.

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

[48] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[49] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SkE6PjC9KX`.

[50] Taesup Kim, Jaesik Yoon, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

[51] Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. *arXiv preprint arXiv:1801.03558*, 2018.

[52] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[53] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[54] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

A 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 A.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 A.1 Stochastic Objective Estimator for Meta-Training.

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

B Additional Related Work Details

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., neighbourhood-based approaches [4], variational inference [26], or inference networks [5]. On the other end of the spectrum are models that adapt *all* the parameters of the classifier, e.g., [7, 8, 50]. The trade-off here is clear: as more parameters are adapted, the resulting model is more flexible, but also slow and prone to over-fitting. For this reason we modulate a small portion of the network parameters, following recent work on multi-task learning [16, 17, 15].

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 [31] and *mini*-imageNet [22]). However, when faced with a diverse set of tasks, such as that introduced recently by Triantafillou et al. [6], it is important to adapt the feature extractor on a per-task basis as well.

The adaptation mechanism $\psi_\phi(D^\tau)$. Adaptation varies in the literature from performing full gradient descent learning with D^τ [9] to relying on simple operations such as taking the mean of class-specific feature representations [4, 24]. Recent work has focused on reducing the number of required gradient steps by learning a global initialization [7, 8] or additional parameters of the optimization procedure [22]. 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 *amortization* [5]. Amortization greatly reduces the cost of adaptation and enables sharing of global parameters, but may suffer from the amortization gap [51] (i.e., underfitting), particularly in the large data regime. Recent work has proposed using semi-amortized inference [6, 23], but have done so while only adapting the classification layer parameters.

C Experimentation Details

All experiments were implemented in PyTorch [52] 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.

C.1 META-DATASET Training and Evaluation Procedure

C.1.1 Feature Extractor Weights θ Pretraining

We first reduce the size of the images in the ImageNet ILSVRC-2012 dataset [21] 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 [6]. 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 [14] architecture on the above 712 training classes. The ResNet-18 architecture is detailed in Table E.7. 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 [52] 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.

C.1.2 Episodic Training of ϕ

Next we train the functions that generate the parameters $\boldsymbol{\psi}_f^\tau$, $\boldsymbol{\psi}_w^\tau$ 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 C.2). Images in the training datasets were cropped using object bounding boxes (if available), resized to be 84×84 pixels, and intensity normalized. Note that for other than ILSVRC and Omniglot, the specific splits for training, validation, and tests sets for each training dataset are not specified, which could affect the comparison of results between different classification algorithms. We train in an end-to-end fashion for 140 epochs (100 epochs when training on ILSVRC only) with the Adam [53] optimizer, using a batch size of 16 episodes, and a fixed learning rate of 0.0005. When training on ILSVRC only, an epoch consists of 1000 episodes. When training on all datasets, an epoch consists of 800 episodes (i.e. 100 episodes per training dataset uniformly sampled). We follow the random dataset, class, and instance sampling procedure as prescribed in [6]. We validate using 100 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 C.2) 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 $\boldsymbol{\phi}$. Note that while training $\boldsymbol{\phi}$ 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 $\boldsymbol{\psi}^\tau$ parameters, with the exception of the set encoder g (that generates the global task representation $\boldsymbol{z}_{\text{G}}^\tau$). 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 tried 100 when training epochs for the all datasets case and found that the loss was still decreasing. As a result, we tried 140 epochs and chose that. We also tried lowering the batch size to 8, but that led to decreased accuracy.

C.1.3 Evaluation

Images in the test datasets were cropped using object bounding boxes (if available), resized to be 84×84 pixels, and intensity normalized. We test all models with 600 episodes each (following the same task-sampling procedure used during training) 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 [21]	ILSVRC [21]	ILSVRC [21]	ILSVRC [21]	ILSVRC [21]	ILSVRC [21]
		Omniglot [31]	Omniglot [31]	Omniglot [31]	Omniglot [31]
		Aircraft [32]	Aircraft [32]	Aircraft [32]	Aircraft [32]
		Birds [33]	Birds [33]	Birds [33]	Birds [33]
		Textures [34]	Textures [34]	Textures [34]	Textures [34]
		Quick Draw [35]	Quick Draw [35]	Quick Draw [35]	Quick Draw [35]
		Fungi [36]	Fungi [36]	Fungi [36]	Fungi [36]
		VGG Flower [37]	VGG Flower [37]	VGG Flower [37]	VGG Flower [37]
		MSCOCO [39]		MSCOCO [39]	MSCOCO [39]
		Traffic Signs [38]			Traffic Signs [38]
		MNIST [29]			MNIST [29]
		CIFAR10 [30]			CIFAR10 [30]
		CIFAR100 [30]			CIFAR100 [30]

Table C.2: Datasets used to train, validate, and test models.

D Additional Few-Shot Classification Results

D.1 Few-Shot Classification Results When Training on ILSVRC-2012 only

Table D.3 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. [6]. In particular, our method relies on training the parameters $\boldsymbol{\phi}$ 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. [6] even in this scenario.

D.2 Feature Extractor Parameter Learning

Figure D.8 shows t-SNE [54] plots that visualize the output of the set encoder $\boldsymbol{z}_{\text{G}}$ 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.

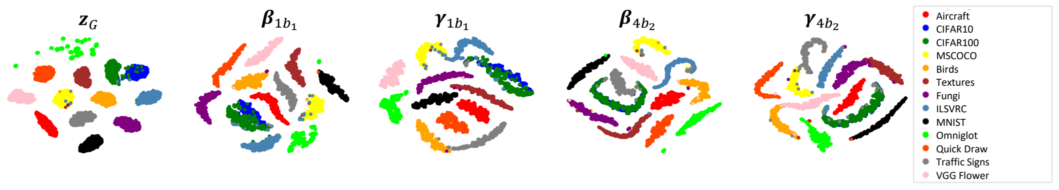


Figure D.8: t-SNE plots of the output of the set encoder $\boldsymbol{z}_{\text{G}}$ and the FiLM layer parameters at the start $(\boldsymbol{\beta}_{1b1}, \boldsymbol{\gamma}_{1b1})$ and end $(\boldsymbol{\beta}_{4b2}, \boldsymbol{\gamma}_{4b2})$ of the feature extraction process at test time.

D.3 Joint Training of θ and ϕ

Our experiments in jointly training $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ show that the two-stage training procedure proposed in Section 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

Dataset	Finetune	MatchingNet	ProtoNet	fo-MAML	Proto-MAML	CNAPs
ILSVRC [21]	47.5±1.1	43.9±1.1	43.4±1.1	29.2±1.0	**50.2±1.1**	**51.0±1.1**
Omniglot [31]	**63.0±1.4**	**62.4±1.3**	60.4±1.4	45.4±1.6	60.7±1.4	51.4±1.3
Aircraft [32]	56.4±1.0	50.6±1.0	48.6±0.9	33.8±0.9	54.5±1.0	**59.6±0.9**
Birds [33]	61.6±1.0	56.4±1.0	63.7±1.0	39.0±1.2	**69.7±1.0**	52.6±1.0
Textures [34]	67.8±0.9	65.6±0.8	62.2±0.8	50.6±0.7	66.7±0.8	**80.2±0.6**
Quick Draw [35]	50.9±1.2	50.2±1.1	50.5±1.0	24.3±1.4	49.0±1.1	**53.9±1.0**
Fungi [36]	33.0±1.1	33.7±1.0	36.0±1.1	16.4±0.9	**39.0±1.0**	35.3±1.2
VGG Flower [37]	82.3±0.9	80.2±0.7	79.5±0.8	56.0±1.2	**85.8±0.8**	**86.1±0.6**
Traffic Signs [38]	55.7±1.2	**59.6±1.2**	46.9±1.1	23.5±1.2	47.8±1.0	52.4±1.2
MSCOCO [39]	33.8±1.4	29.8±1.2	35.2±1.1	13.5±1.0	38.1±1.2	**42.7±1.0**
MNIST [29]						71.4±0.7
CIFAR10 [30]						75.6±0.8
CIFAR100 [30]						45.6±1.1

Table D.3: Few-shot classification results on META-DATASET [6] 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 [6]

Dataset	Joint Training (warmstart BN)	Joint Training (BN train mode)	Two-Stage Training (BN test mode)
ILSVRC [21]	17.3±0.7	41.6±1.0	49.5±1.0
Omniglot [31]	74.9±1.0	80.8±0.9	89.7±0.5
Aircraft [32]	51.4±0.8	70.5±0.7	87.2±0.5
Birds [33]	44.1±1.0	48.3±1.0	76.7±0.9
Textures [34]	49.1±0.7	73.5±0.6	83.0±0.6
Quick Draw [35]	46.6±1.0	71.5±0.8	72.3±0.8
Fungi [36]	20.4±0.9	43.1±1.1	50.5±1.1
VGG Flower [37]	66.6±0.8	71.0±0.7	92.5±0.4
Traffic Signs [38]	21.2±0.8	40.4±1.1	48.4±1.1
MSCOCO [39]	18.8±0.7	37.1±1.0	39.7±0.9

Table D.4: Few-shot classification results on META-DATASET [6] 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 ± sign indicates the 95% confidence interval. Bold text indicates the highest scores that overlap in their confidence intervals.

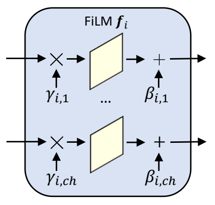
we propose in the main text: only using batch normalization in “eval” mode, which enforces that no information is transferred across tasks or datasets. (ii) “Warm-start" the training procedure with batch normalization in “train” mode, and after a number of epochs (we use 50 for the results shown below), switch to proper usage of batch normalization. All other training procedures we attempted diverged.

Table D.4 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.

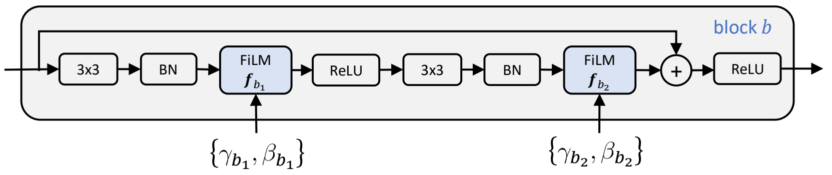
E Network Architecture Details

E.1 FiLM Layer Details

In Section 2 we introduced the usage of FiLM layers [15] to modulate the parameters of a feature extractor and allow it to adapt the resulting feature representation to specific tasks τ. Figure E.9a illustrates how a FiLM layer operates on a convolutional layer: each channel in the layer (i.e. an image-shaped activation) has an associated pair of scalars (γ and β) which are used to apply a shared affine transformation to the entire channel. Figure E.9b illustrates how a standard ResNet block [14] can be modified to include FiLM layers. Note that each block contains two FiLM layers, one associated with each of the convolutional layers in the block.



(a) A FiLM layer.



block b
3x3
BN
FiLM f_{b_1}
ReLU
3x3
BN
FiLM f_{b_2}
ReLU
$\{\gamma_{b_1}, \beta_{b_1}\}$
$\{\gamma_{b_2}, \beta_{b_2}\}$

(b) A ResNet basic block with FiLM layers.

Figure E.9: (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 [14].

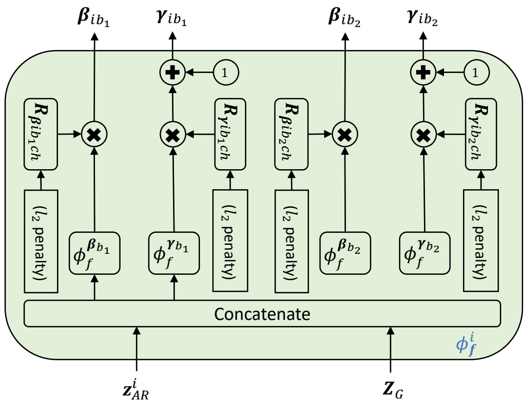


Figure E.10: Adaptation network ϕ_f. $\boldsymbol{R}_{\gamma ib_j ch}$ and $\boldsymbol{R}_{\beta ib_j ch}$ denote a vector of regularization weights that are learned with an l_2 penalty.

Figure E.10 shows the details of the adaptation network ϕ_f that generates the FiLM layer parameters for each ResNet layer.

E.2 ResNet18 Architecture details

Throughout our experiments in Section 5, we use a ResNet18 [14] as our feature extractor, the parameters of which we denote $\boldsymbol{\theta}$. Table E.5 and Table E.6 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 E.7 details how these blocks are composed to generate the overall feature extractor network. We use the implementation that is provided by the PyTorch [52][3], though we adapt the code to enable the use of FiLM layers.

Table E.5: ResNet-18 basic block b.

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 E.6: ResNet-18 basic scaling 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

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

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 E.7: ResNet-18 feature extractor network.

E.3 Adaptation Network Architecture Details

In this section, we provide the details of the architectures used for our adaptation networks. Table E.8 details the architecture of the set encoder $g : D^\tau \mapsto \boldsymbol{z}_{\text{G}}$ that maps context sets to global representations.

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)
256	flatten

Table E.8: Set encoder g.

Table E.9 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 E.7). 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 E.10, and illustrated in Figure E.10. Note that, as depicted in Figure E.10, each layer has four networks with architectures as detailed in Table E.10, one for each $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$, for each convolutional layer in the block.

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

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 E.9: 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
$256 + l_i$ channels	Input from Concatenate
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

Table E.10: Network ϕ_f.

E.4 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: $\boldsymbol{z}_c^\tau = \frac{1}{N_c^\tau} \sum_{\boldsymbol{x} \in D_c^\tau} f_{\boldsymbol{\theta}}(\boldsymbol{x}; \boldsymbol{\psi}_f^\tau)$, 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 E.11 and Table E.12. The resulting weights and biases (for each class in task τ) can then be used as a linear classification layer, as detailed in Table E.13.

Table E.11: Network $\boldsymbol{\phi}_w$.

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

Output size	Layers
512	Input from mean pooling
512	$2 \times$ fully connected, ELU
512	fully connected
512	Sum with Input

Table E.12: Network $\boldsymbol{\phi}_b$.

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

Output size	Layers
512	Input from mean pooling
512	$2 \times$ fully connected, ELU
1	fully connected

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 E.13: Linear classifier network.

F Continual Learning Implementation Details

As noted in Sections 2 and 5, our model can be applied to continual learning with one small modification: we store a compact representation of our training data that can be updated at each step of the continual learning procedure. Notice that Figure 2 indicates that the functional representation of our linear classification layer $\boldsymbol{\psi}_w^\tau(\cdot)$ contains a mean pooling layer that combines the per-class output of our feature extractor $\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_m^\tau; \boldsymbol{\psi}_f) \,|\boldsymbol{x}_m^\tau \in D^\tau, \boldsymbol{y}_m^\tau = c\}$. The result of this pooling,

$$\boldsymbol{z}_c = \frac{1}{M} \sum f_{\boldsymbol{\theta}}(\boldsymbol{x}_m^\tau; \boldsymbol{\psi}_f) \tag{F.4}$$

where $M = |\{f_{\boldsymbol{\theta}}(\boldsymbol{x}_m^\tau; \boldsymbol{\psi}_f) \,|\boldsymbol{x}_m^\tau \in D^\tau, \boldsymbol{y}_m^\tau = c\}|$, is supplied as input to the network $\boldsymbol{\psi}_w(\cdot)$. 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 $\boldsymbol{z}_c' = \frac{1}{M} \sum f_{\boldsymbol{\theta}}\left(\boldsymbol{x}_m^{\tau\prime}; \boldsymbol{\psi}_f\right)$ the pooled average resulting from M new training examples $\boldsymbol{x}_m^{\tau\prime}$ for class c. We then update $\boldsymbol{z}_c$ with the weighted average: $\boldsymbol{z}_c \leftarrow \frac{M\boldsymbol{z}_c + N\boldsymbol{z}_c}{M+N}$. 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 were able to achieved good performance. However, for continual learning experiments that contain multiple datasets, we anticipate that this issue will need to be addressed.

G Additional Continual Learning Results

In Section 5 we provided results for continual learning experiments with Split MNIST [41] and Split CIFAR100 [42]. The results showed the average performance as more tasks were observed for the single and multi head settings. Here, we provide more complete results, detailing the performance through "time" at the task level. Figure G.11 details the performance of CNAPs (with varying

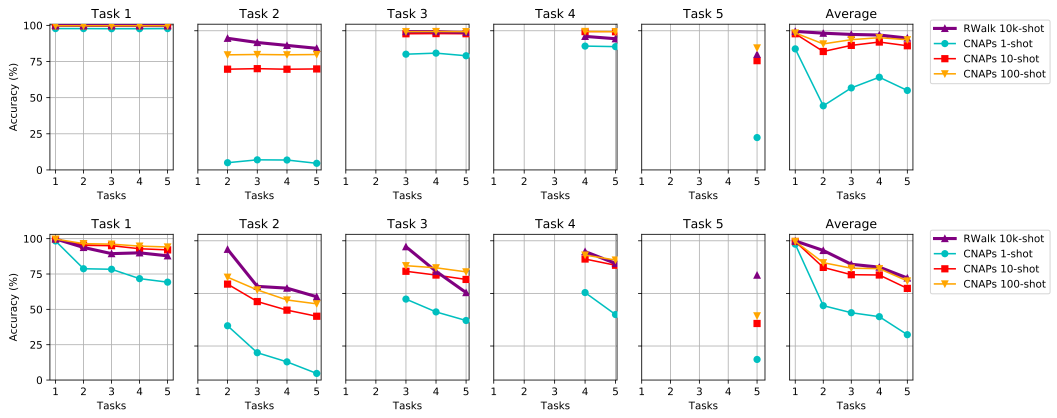


Figure G.11: Continual learning results on Split MNIST. Top row is multi-head, bottom row is single-head.

number of observed examples) and Riemannian Walk (RWalk) [42] 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 G.11 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 G.12 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.

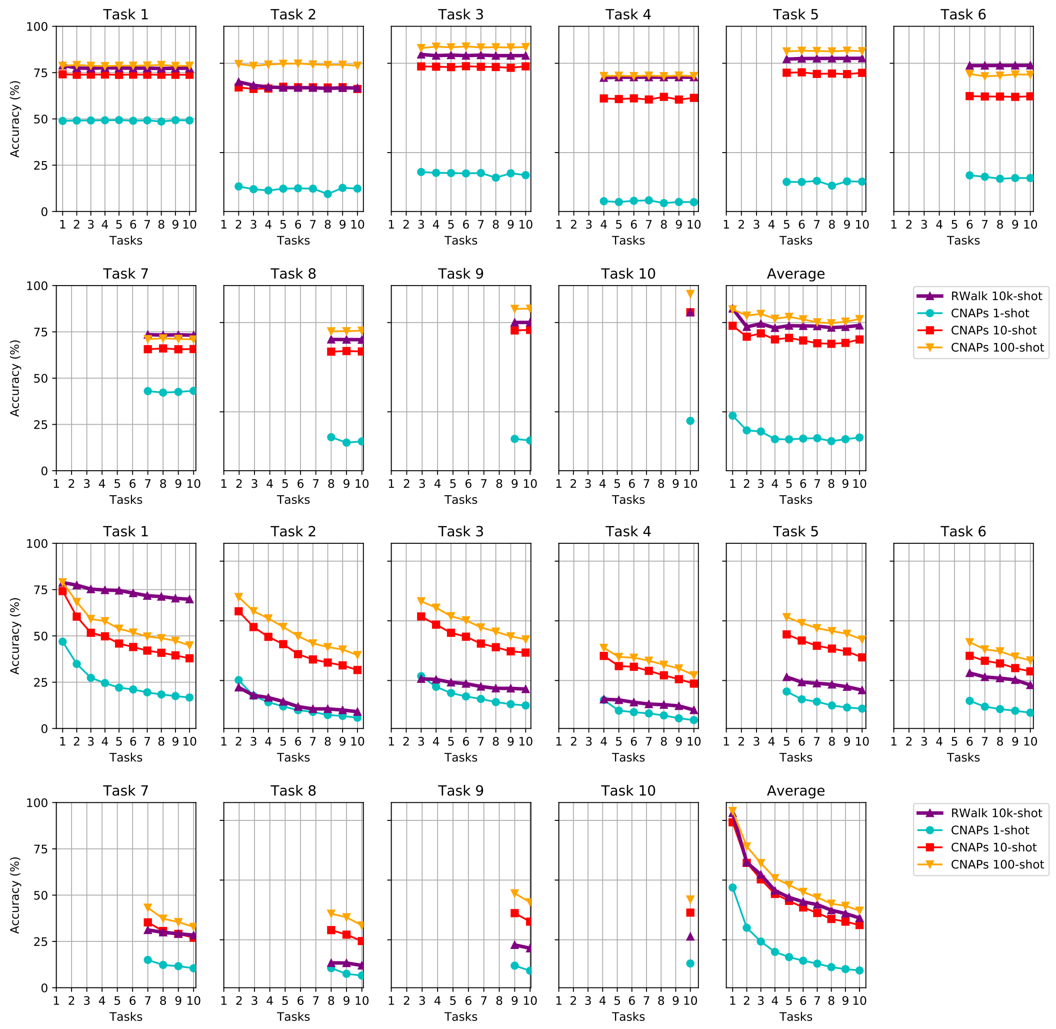


Figure G.12: Continual learning results on Split CIFAR100. Top two rows are multi-head, bottom two rows are single-head.

H Additional Active Learning Results

In Section 5 we provided active learning results for CNAPS and Prototypical Networks on the VGG Flowers dataset and three held out test languages from the Omniglot dataset. Here, we provide the results from all twenty held-out languages in Omniglot.

Figure H.13 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 [46] 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.

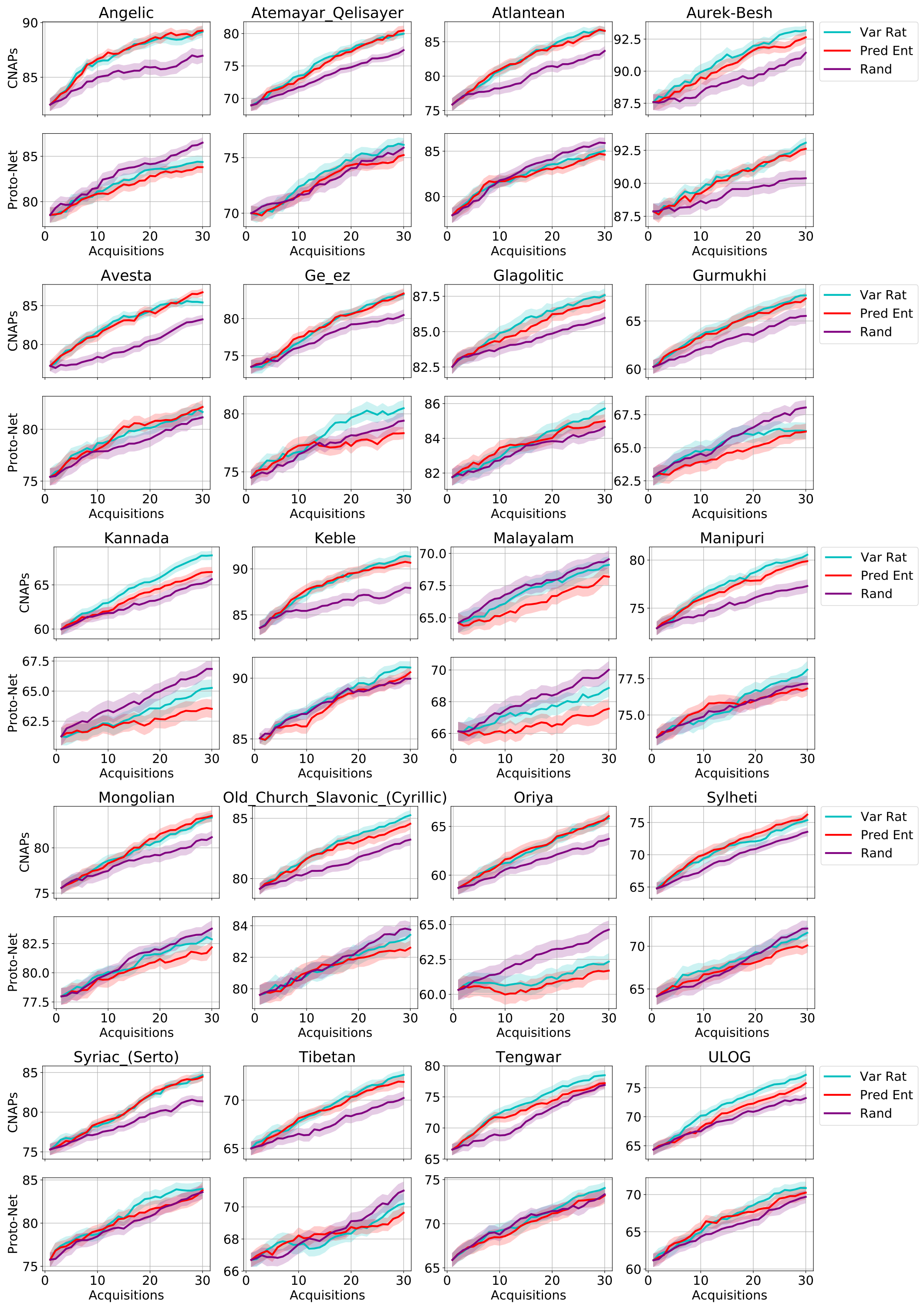


Figure H.13: Active learning results on all twenty held-out OMNIGLOT languages.