Improved Few-Shot Visual Classification

Peyman Bateni[1], Raghav Goyal[1,3], Vaden Masrani[1], Frank Wood[1,2,4], Leonid Sigal[1,3,4]
[1]University of British Columbia, [2]MILA, [3]Vector Institute, [4]CIFAR AI Chair
{pbateni, rgoyal14, vadmas, fwood, lsigal}@cs.ubc.ca

Abstract

Few-shot learning is a fundamental task in computer vision that carries the promise of alleviating the need for exhaustively labeled data. Most few-shot learning approaches to date have focused on progressively more complex neural feature extractors and classifier adaptation strategies, as well as the refinement of the task definition itself. In this paper, we explore the hypothesis that a simple class-covariance-based distance metric, namely the Mahalanobis distance, adopted into a state of the art few-shot learning approach (CNAPS [26]) can, in and of itself, lead to a significant performance improvement. We also discover that it is possible to learn adaptive feature extractors that allow useful estimation of the high dimensional feature covariances required by this metric from surprisingly few samples. The result of our work is a new "Simple CNAPS" architecture which has up to 9.2% fewer trainable parameters than CNAPS and performs up to 6.1% better than state of the art on the standard few-shot image classification benchmark dataset.

1. Introduction

Deep learning successes have led to major computer vision advances [11, 32, 9]. However, most methods behind these successes have to operate in fully-supervised, high data availability regimes. This limits the applicability of these methods, effectively excluding domains where data is fundamentally scarce or impossible to label en masse. This inspired the field of few-shot learning [37, 38], methods which aim to computationally mimic human reasoning and learning from limited data.

The goal of few-shot learning is to automatically adapt models such that they work well on instances from classes not seen at training time, given only a few labelled examples for each new class. In this paper, we focus on few-shot image classification where the ultimate aim is to develop a classification methodology that automatically adapts to new classification tasks at test time, and particularly in the case where only a very small number of labelled "support" im-

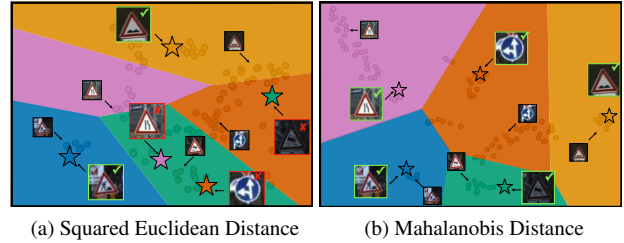(a) Squared Euclidean Distance   (b) Mahalanobis Distance

Figure 1: **Class-covariance metric:** t-SNE visualization of embedded image features (points), class decision boundaries, and test query instances (stars) of standard L_2-based (left) and our proposed class-covariance-based distance metric (right) classifier on top of CNAPS-adopted [26] image features. For convenience of comparison, the plot on the right is visualized in a transformed space where decision boundaries are linear. The features are obtained from a task taken from the out-of-domain Traffic Signs dataset. The task contains 112 support samples spanning across 5 classes, with 1 query instance depicted for each class. Illustrating the advantage deriving from our choice of the Mahalanobis distance for classification, the decision boundaries on the (right) are better aligned with the support embeddings, and query instances of three out of five classes are misclassified by L_2-based classifier (left). Correctly classified queries are denoted by green checks and incorrect ones by red ×.

ages are available per class.

Few-shot learning approaches typically take one of two forms: 1) nearest neighbor approaches and their variants, including matching networks [35], which effectively apply nearest-neighbor or weighted nearest neighbor classification on the samples themselves, either in a feature [13, 14, 29] or a semantic space [4]; or 2) embedding methods that effectively distill all of the examples to a single prototype per class, where a prototype may be learned [7, 26] or implicitly derived from the samples [31] (*e.g.* mean embedding). The prototypes are often defined in feature or semantic space (*e.g.* word2vec [39]). Most research in this

1

domain has focused on learning non-linear mappings (often expressed as neural nets) from images to the embedding space subject to a pre-defined metric in the embedding space used for final nearest class classification (usually cosine similarity between query image embedding and class embedding). Most recently CNAPS [26] achieved state of the art (SoTA) few-shot visual image classification by utilizing sparse FiLM [24] layers within the context of episodic training to get around problems that arise from trying to adapt the entire embedding neural network using few support samples.

Overall much less attention has been given to the metric used to compute distances for classification in the embedding space. Presumably this is because common wisdom dictates that flexible non-linear mappings are ostensibly able to adapt to any such metric, making the choice of metric apparently inconsequential. In practice, as we find in this paper, the choice of metric is quite important. In [31] the authors analyze the underlying distance function used in order to justify the use of sample means as prototypes. They argue that Bregman divergences [1] are the theoretically sound family of metrics to use in this setting, but only utilize a single instance within this class squared Euclidean distance, which they find to perform better than the more traditional cosine metric. However, the choice of Euclidean metric involves making two flawed assumptions: 1) that feature dimensions are un-correlated and 2) that they have uniform variance. Further it is insensitive to the distribution of within-class samples with respect to their prototype and recent results [23, 31] suggest that this is problematic. Modeling this distribution (in the case of [1] using extreme value theory) is, as we find, a key to better performance.

Our Contributions.

1. A robust empirical finding of a significant 6.1% improvement on average over SoTA (CNAPS [26]) in few-shot image classification, obtained by utilizing a test-time-estimated class-covariance-based distance metric, namely the Mahalanobis distance [5], in final, task-adapted classification.

2. The surprising finding that we are able to estimate such a metric even in the few shot classification setting, where the number of available support examples per class is far too few in theory to estimate the required class-specific covariances.

3. A new "Simple CNAPS" architecture that achieves this performance despite removing 788,485 parameters (3.2%-9.2% of the total) from original CNAPS architecture, replacing them with fixed, not-learned, deterministic covariance estimation and Mahalanobis distance computations.
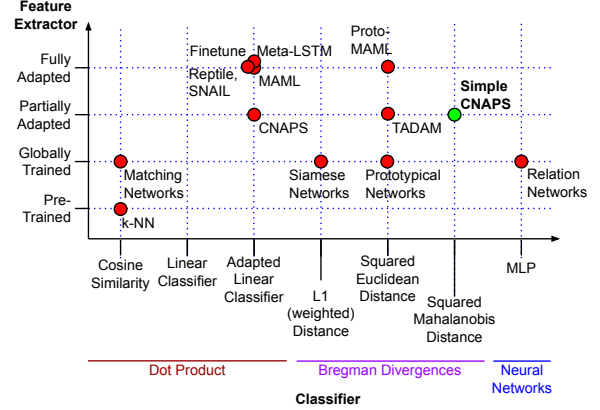

Figure 2: **Approaches to few-shot image classification:** organized by image feature extractor adaptation scheme (vertical axis) versus final classification method (horizontal axis). Our method (Simple CNAPS) partially adapts the feature extractor (which is architecturally identical to CNAPS) but is trained with, and uses, a fixed (rather than adapted) Mahalanobis metric for final classification.

4. Evidence that should make readers question the common understanding that CNN feature extractors of sufficient complexity can adapt to whatever the final metric is (be it cosine similarity/dot product or otherwise).

2. Related Work

The last decade's few-shot learning work [38] can be differentiated along two main axes: 1) how images are transformed into vectorized embeddings, and 2) how "distances" are computed between vectors in order to assign labels. This is shown in Figure 2.

Siamese networks [14], an early approach to few-shot learning and classification, used a shared feature extractor to produce embeddings for both the support and query images. Classification was then done by picking the smallest weighted L1 distance between query and labelled image embeddings. Relation networks [33] (and recent GCNN variations [13, 29]) extended this by parameterizing and learning the classification metric using a Multi-Layer Perceptron (MLP). Matching networks [35] learned distinct feature extractors for support images and query images which were then used to compute cosine similarities for classification.

The feature extractors used by these models were, notably, not adapted to test-time classification tasks. It has become established that adapting feature extraction to new tasks at test time is generally a good thing to do. Fine tuning transfer-learned networks [40] did this by continuing training of the feature extractor network using the task-specific
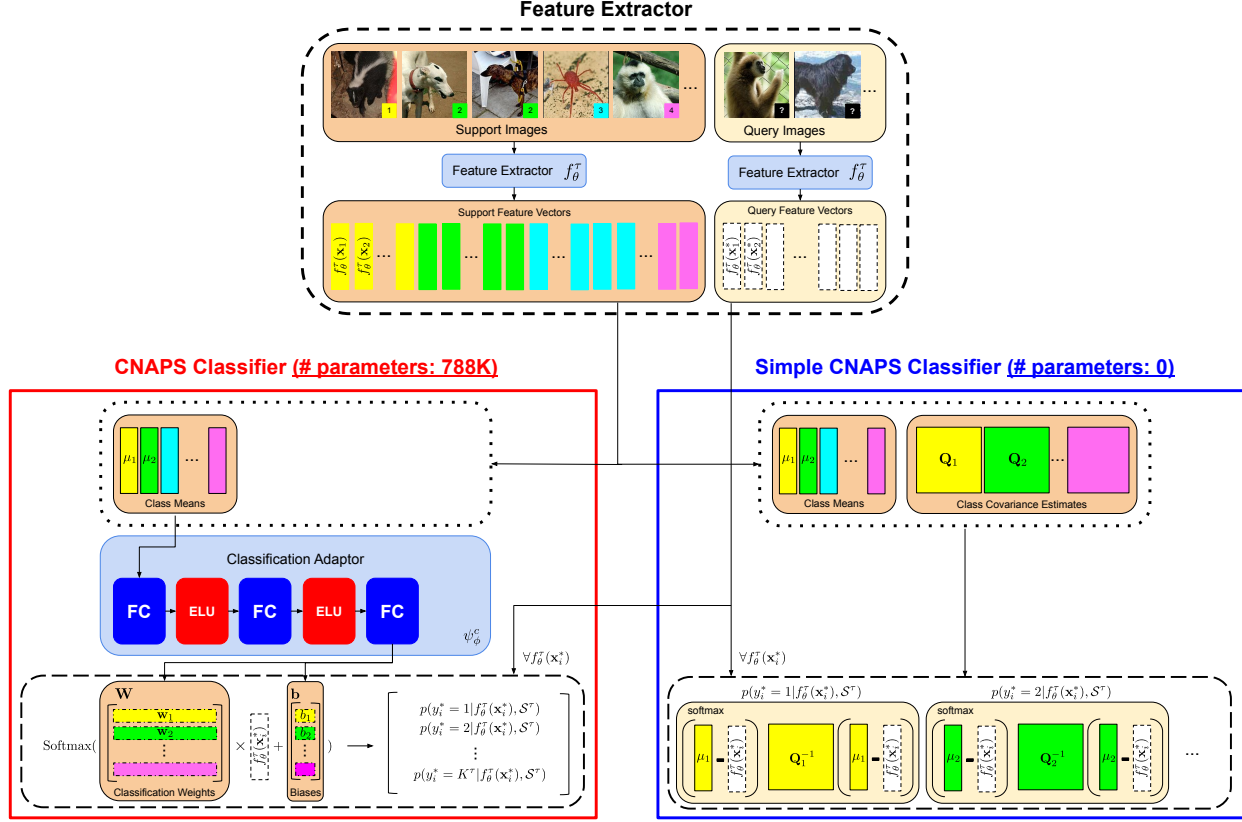
Figure 3: **Comparison of the feature extraction and classification in CNAPS versus Simple CNAPS:** Both CNAPS and Simple CNAPS share the feature extraction adaptation architecture detailed in Figure 4. CNAPS and Simple CNAPS differ in how distances between query feature vectors and class feature representations are computed for classification. CNAPS uses a trained, adapted linear classifier whereas Simple CNAPS uses a differentiable but fixed and parameter-free deterministic distance computation. Components in light blue are have parameters that are trained, namely f_θ^τ in both models and ψ_ϕ^c in the CNAPS adaptive classification. As shown, CNAPS classification requires 778k parameters while Simple CNAPS is fully deterministic.

support images but found limited success due to problems related to overfitting to, the generally very few, support examples. MAML [3] (and its many extensions [20, 21, 25]) mitigated this issue by learning a set of meta-parameters that specifically enabled the feature extractors to be adapted to new tasks given few support examples using very few gradient steps.

The two methods most similar to our own are CNAPS [26] (and the related TADAM [23]) and Prototypical networks [31]. CNAPS is a few-shot adaptive classifier based on conditional neural processes (CNP) [6]. It is the state of the art approach for few-shot image classification [26]. It uses a pre-trained feature extractor augmented with FiLM layers [24] that are adapted for each task using the support images specific to that task. CNAPS uses a dot-product distance in a final linear classifier; the parameters of which are also adapted at test-time to each new task. We describe CNAPS in greater detail when describing our method.

Prototypical networks [31] do not use a feature adaptation network, they instead use a simple mean pool operation to form class "prototypes." Euclidean distances to these prototypes are then subsequently used for classification. Their choice of the distance metric was motivated by the theoretical benefits of Bregman divergences [1], a family of functions of which the squared Euclidean distance is a member.

Our work differs from CNAPS [26] and Prototypical networks [31] in the following ways. First, while CNAPS has demonstrated the importance of adapting the feature extractor to a specific task, we show that adapting the classifier is actually unnecessary to obtain good performance. Second, we demonstrate that an improved choice of Bregman divergence can significantly impact accuracy. Specifically we show that regularized class-specific covariance estimation from task-specific adapted feature vectors allows the use of the Mahalanobis distance for classification, achieving

a significant improvement over state of the art. A high-level diagrammatic comparison of our "Simple CNAPS" architecture to CNAPS can be found in Figure 3.

3. Formal Problem Definition

We frame few-shot image classification as an amortized classification task. Assume that we have a large labelled dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of images $\mathbf{x}_i$ and labels y_i. From this dataset we can construct a very large number of classification *tasks* $\mathcal{D}^\tau \subseteq \mathcal{D}$ by repeatedly sampling without replacement from $\mathcal{D}$. Let $\tau \in \mathbb{Z}_+$ uniquely identify a classification task. We define the *support set* of a task to be $\mathcal{S}^\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N^\tau}$ and the *query set* $Q^\tau = \{(\mathbf{x}_i^*, y_i^*)\}_{i=1}^{N^{*\tau}}$ where $\mathcal{D}^\tau = \mathcal{S}^\tau \cup Q^\tau$ where $\mathbf{x}_i, \mathbf{x}_i^* \in \mathbb{R}^D$ are vectorized images and $y_i, y_i^* \in \{1, ..., K\}$ are class labels. Our objective is to find parameters θ of a classifier f_θ that maximizes $\mathbb{E}_\tau[\prod_{Q^\tau} p(y_i^* | f_\theta(\mathbf{x}_i^*, \mathcal{S}^\tau)]$.

In practice $\mathcal{D}$ is constructed by concatenating large image classification datasets and the set of classification tasks $\{\mathcal{D}^\tau\}_{\tau=1}$ is sampled in a more complex way than simply without replacement. In particular constraints are placed on the relationship of the image label pairs present in the support set and those present in the query set. For instance, in few-shot learning the constraint that the query set labels are a subset of the support set labels is imposed. With this constraint imposed the classification task reduces to correctly assigning each query set image to one of the classes present in the support set. Also, in this constrained few-shot classification case, the support set can be interpreted as being the "training data" for implicitly training (or adapting) a task-specific classifier of query set images.

4. Method

Our classifier shares feature adaptation architecture with CNAPS [26], but deviates from CNAPS by replacing their adaptive classifier with a simpler classification scheme based on estimating Mahalanobis distances. To explain our classifier, which we call "Simple CNAPS," we first detail CNAPS in 4.1, before presenting our classifier in 4.2.

4.1. CNAPS

Conditional Neural Adapative Processes (CNAPS) consist of two elements: a feature extractor and a classifier, both of which are task-adapted. Adaptation is performed by trained adaptation modules that take the support set.

The feature extractor architecture used in both CNAPS and Simple CNAPS is shown in Figure 4. It consists of a ResNet18 [8] network pre-trained on ImageNet [27] which also has been augmented with FiLM layers [24]. The parameters $\{\gamma_j, \beta_j\}_{j=1}^4$ of the FiLM layers can scale and shift the extracted features at each layer of the ResNet18, allowing the feature extractor to focus and disregard different fea-

tures on a task-by-task basis. A feature adaptation module ψ_ϕ^f is trained to produce $\{\gamma_j, \beta_j\}_{j=1}^4$ based on the support examples $\mathcal{S}^\tau$ provided for the task.

The feature extractor adaptation module ψ_ϕ^f consists of two stages: support set encoding followed by film layer parameter production. The set encoder $g_\phi(\cdot)$, parameterized by a deep neural network, produces a permutation invariant task representation $g_\phi(\mathcal{S}^\tau)$ based on the support images $\mathcal{S}^\tau$. This task representation is then passed to ψ_ϕ^j which then produces the FiLM parameters $\{\gamma_j, \beta_j\}$ for each block j in the ResNet. Once the FiLM parameters have been set, the feature extractor has been adapted to the task. We use f_θ^τ to denote the feature extractor adapted to task τ. The CNAPS paper [26] also proposes an auto-regressive adaptation method which conditions each adaptor ψ_ϕ^j on the output of the previous adapter ψ_ϕ^{j-1}. We refer to this variation as AR-CNAPS but for conciseness we omit the details of this architecture here, and instead refer the interested reader to [26] or to Appendix A.1.

Classification in CNAPS is performed by a task-adapted linear classifier where the class probabilities for a query image $\mathbf{x}_i^*$ are computed as $\text{softmax}(\mathbf{W} f_\theta^\tau(\mathbf{x}_i^*) + \mathbf{b})$. The classification weights $\mathbf{W}$ and biases $\mathbf{b}$ are produced by the classifier adaptation network ψ_ϕ^c forming $[\mathbf{W}, \mathbf{b}] = [\psi_\phi^c(\boldsymbol{\mu}_1) \ \psi_\phi^c(\boldsymbol{\mu}_2) \ \dots \ \psi_\phi^c(\boldsymbol{\mu}_K)]^T$ where for each class k in the task, the corresponding row of classification weights is produced by ψ_ϕ^c from the class mean $\boldsymbol{\mu}_k$. The class mean $\boldsymbol{\mu}_k$ is obtained by mean-pooling the feature vectors of the support examples for class k extracted by the adapted feature extractor f_θ^τ. A visual overview of the CNAPS adapted classifier architecture is shown in Figure 3, bottom left, red.

4.2. Simple CNAPS

In Simple CNAPS, we also use the same pre-trained ResNet18 for feature extraction with the same adaptation module ψ_ϕ^f, although, because of the classifier architecture we use, it becomes trained to do something different than it does in CNAPS. This choice, like for CNAPS, allows for a task-specific adaptation of the feature extractor. Unlike CNAPS, we directly compute

$$p(y_i^* = k | f_\theta^\tau(\mathbf{x}_i^*), \mathcal{S}^\tau) = \text{softmax}(-d_k(f_\theta^\tau(\mathbf{x}_i^*), \boldsymbol{\mu}_k)) \quad (1)$$

using a deterministic, fixed d_k

$$d_k(\mathbf{x}, \mathbf{y}) = \frac{1}{2}(\mathbf{x} - \mathbf{y})^T (\mathbf{Q}_k^\tau)^{-1} (\mathbf{x} - \mathbf{y}). \quad (2)$$

Here $\mathbf{Q}_k^\tau$ is a covariance matrix specific to the task and class. As we cannot know the value of $\mathbf{Q}_k^\tau$ ahead of time, it must be estimated from the feature embeddings of the task-specific support set. As the number of examples in any particular support set is likely to be much smaller than the di-
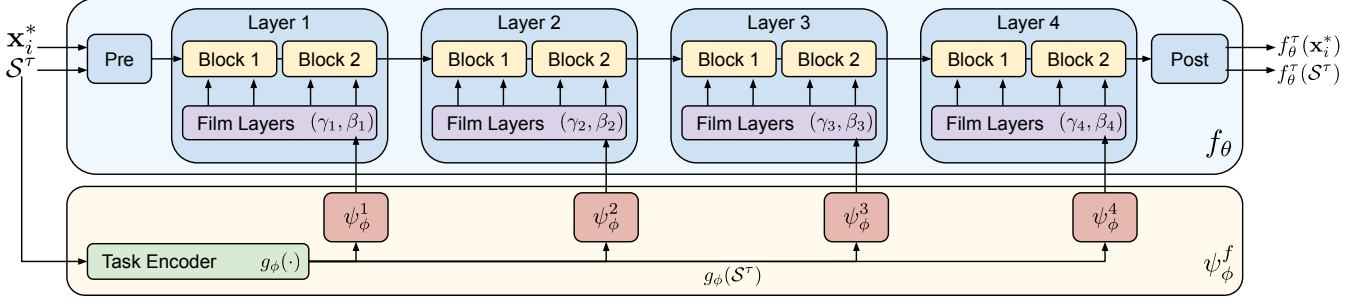
Figure 4: **Overview of the feature extractor adaptation methodology in CNAPS:** task encoder $g_\phi(\cdot)$ provides the adaptation network ψ_ϕ^i at each block i with the task representations ($g_\phi(\mathcal{S}^\tau)$ to produce FiLM parameters (γ_j, β_j). For details on the auto-regressive variation (AR-CNAPS), architectural implementations, and FiLM layers see Appendix A. For an in-depth explanation, refer to the original paper [26].

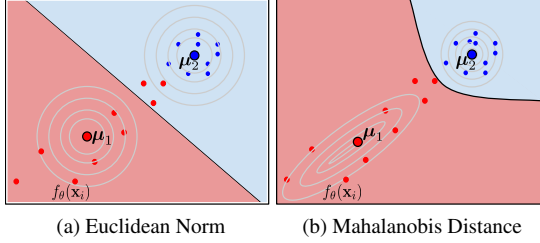(a) Euclidean Norm   (b) Mahalanobis Distance

Figure 5: **Problematic nature of the unit-normal assumption:** The Euclidean Norm (left) assumes embedded image features $f_\theta(\mathbf{x}_i)$ are distributed around class means $\boldsymbol{\mu}_k$ according to a unit normal. The Mahalanobis distance (right) considers cluster variance when forming decision boundaries, indicated by the background colour.

mension of the feature space, we use a regularized estimator

$$\mathbf{Q}_k^\tau = \lambda_k^\tau \mathbf{\Sigma}_k^\tau + (1 - \lambda_k^\tau)\mathbf{\Sigma}^\tau + \beta I. \qquad (3)$$

formed from a convex combination of the class-within-task and all-classes-in-task covariance matrices $\mathbf{\Sigma}_k^\tau$ and $\mathbf{\Sigma}^\tau$ respectively.

We estimate the class-within-task covariance matrix $\mathbf{\Sigma}_k^\tau$ using the feature embeddings $f_\theta^\tau(\mathbf{x}_i)$ of all $\mathbf{x}_i \in \mathcal{S}_k^\tau$ where $\mathcal{S}_k^\tau$ is the set of examples in $\mathcal{S}^\tau$ with class label k.

$$\mathbf{\Sigma}_k^\tau = \frac{1}{|\mathcal{S}_k^\tau|-1} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}_k^\tau} (f_\theta^\tau(\mathbf{x}_i) - \boldsymbol{\mu}_k)(f_\theta^\tau(\mathbf{x}_i) - \boldsymbol{\mu}_k)^T.$$

If the number of support instance of that class is one, i.e. $|\mathcal{S}_k^\tau| = 1$, then we define $\mathbf{\Sigma}_k^\tau$ to be the zero matrix of the appropriate size. The all-classes-in-task covariance $\mathbf{\Sigma}^\tau$ is estimated in the same way as the class-within-task except that it uses *all* the support set examples $\mathbf{x}_i \in \mathcal{S}^\tau$ regardless of their class.

We choose a particular, deterministic scheme for computing the weighting of class and task specific covariance

estimates, $\lambda_k^\tau = |\mathcal{S}_k^\tau|/(|\mathcal{S}_k^\tau|+1)$. This choice means that in the case of a single labeled instance for class in the support set, a single "shot," $\mathbf{Q}_k^\tau = 0.5\mathbf{\Sigma}^\tau + \beta\mathbf{I}$. This can be viewed as increasing the strength of the regularization parameter β relative to the task covariance $\mathbf{\Sigma}^\tau$. When $|\mathcal{S}_k^\tau| = 2$, λ_k^τ becomes $2/3$ and $\mathbf{Q}_k^\tau$ only partially favors the class-level covariance over the all-class-level covariance. In a high-shot setting, λ_k^τ tends to 1 and $\mathbf{Q}_k^\tau$ mainly consists of the class-level covariance. The intuition behind this formula for λ_k^τ is that the higher the number of shots, the better the class-within-task covariance estimate gets, and the more $\mathbf{Q}_k^\tau$ starts to look like $\mathbf{\Sigma}_k^\tau$. We considered other ratios and making λ_k^τ's learnable parameters, but found that out of all the considered alternatives the simple deterministic ratio above produced the best results. The architecture of the classifier in Simple CNAPS appears in Figure 3, bottom-right, blue.

5. Theory

The class label probability calculation appearing in Equation 1 corresponds to an equally-weighted exponential family mixture model [31], where the exponential family distribution is uniquely determined by a regular Bregman divergence [1]

$$D_F(\mathbf{z}, \mathbf{z}') = F(\mathbf{z}) - F(\mathbf{z}') - \nabla F(\mathbf{z}')^T(\mathbf{z} - \mathbf{z}') \qquad (4)$$

for a differentiable and strictly convex function F. The squared Mahalanobis distance in Equation 2 is a Bregman divergence generated by the convex function $F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{\Sigma}^{-1} \mathbf{x}$ and corresponds to the multivariate normal exponential family distribution. This allows us to view the class probabilities Equation 1 as the "responsibilities" in a Gaussian mixture model

$$p(y_i^* = k | f_\theta^\tau(\mathbf{x}_i^*), \mathcal{S}^\tau) = \frac{\pi_k \mathcal{N}(\boldsymbol{\mu}_k, \mathbf{Q}_k^\tau)}{\sum_{k'} \pi_k' \mathcal{N}(\boldsymbol{\mu}_{k'}, \mathbf{Q}_{k'}^\tau)} \qquad (5)$$

with equally weighted mixing coefficient $\pi_k = 1/k$.

This perspective immediately highlights a problem with the squared Euclidean norm, used by a number of approaches as shown in Fig. 2. The Euclidean norm, which corresponds to the squared Mahalanobis distance with $\mathbf{Q}_k^\tau = \mathbf{I}$, implicitly assumes each cluster is distributed according to a unit normal, as seen in Figure 5. By contrast, the squared Mahalanobis distance considers cluster covariance when computing distances to the cluster centers.

6. Experiments

We evaluate Simple CNAPS on the Meta-Dataset [34] family of datasets, demonstrating improvements compared to nine baseline methodologies including the current SoTA, CNAPS. Benchmark results reported come from [34, 26].

Dataset Meta-Dataset [34] is a benchmark for few-shot learning and image classification. It is the union of 10 labeled image datasets: ILSVRC-2012 (ImageNet) [27], Omniglot [16], FGVC-Aircraft (Aircraft) [19], CUB-200-2011 (Birds) [36], Describable Textures (DTD) [2], QuickDraw [12], FGVCx Fungi (Fungi) [30], VGG Flower (Flower) [22], Traffic Signs (Signs) [10] and MSCOCO [18]. In keeping with prior art we report results using the first 8 datasets for training, reserving Traffic Signs and MSCOCO for "out-of-domain" performance evaluation. Additionally, from the eight training datasets used for training, some classes are held out for testing, to evaluate "in-domain" performance. Following [26], we extend the out-of-domain evaluation with 3 more datasets: MNIST [17], CIFAR10 [15] and CIFAR100 [15]. We report results using standard test/train splits provided by [34], but, importantly, we have cross-validated our critical empirical claims using different test/train splits and our results are robust across folds (see Appendix B).

6.1. Results

In-domain performance The in-domain results for Simple CNAPS and Simple AR-CNAPS are shown in Table 1. Simple AR-CNAPS has the sample Mahalanobis-based classifier as Simple CNAPS but uses the autoregressive feature extraction version of CNAPS: AR-CNAPS. Simple AR-CNAPS outperforms previous SoTA models on 7 out of the 8 datasets while matching past SoTA on FGVCx Fungi (Fungi). Simple CNAPS outperforms baselines on 6 out of 8 datasets while matching performance on FGVCx Fungi (Fungi) and Describable Textures (DTD). Overall, in-domain performance gains are considerable in the few-shot learning domain with 2-6% margins. Simple CNAPS achieves an average 73.8% accuracy on in-domain few-shot classification, a 4.2% gain over CNAPS, while Simple AR-CNAPS achieves 73.5% accuracy, a 3.8% gain over AR-CNAPS.
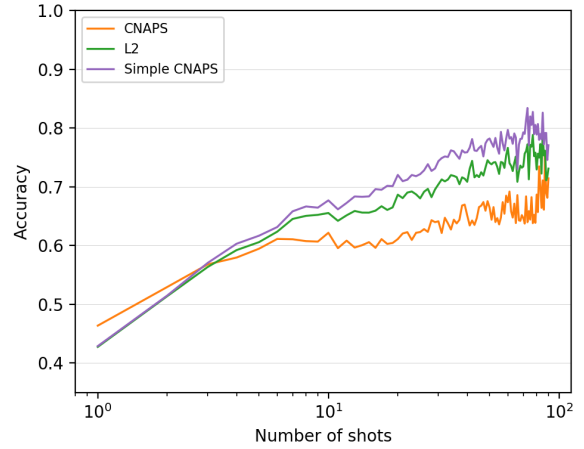

Figure 6: **Accuracy vs. Shots:** Average number of support examples (in log scale) per class v/s accuracy. The accuracies of all classes within tasks in the test set (both in-domain and out-of-domain) are grouped together according to number of support examples per class. Each group of accuracies is then averaged to obtain a value for each shot. The squared euclidean variation has been referred to as L2.

Out-of-domain performance As shown in Table 2, Simple CNAPS and Simple AR-CNAPS produce substantial gains in classification accuracy on out-of-domain datasets, each exceeding the SoTA baseline. With an average out-of-domain accuracy of 69.7% and 67.6%, Simple CNAPS and Simple AR-CNAPS outperform SoTA by 8.2% and 7.8%. This means that Simple CNAPS/AR-CNAPS generalizes to out-of-domain datasets better than baseline models. Note that Simple AR-CNAPS under-performs Simple CNAPS, suggesting that the auto-regressive feature adaptation approach may overfit to the domain of datasets it has been trained on.

Overall performance Overall, Simple CNAPS achieves the best classification accuracy at 72.2% with Simple AR-CNAPS trailing very closely at 71.2%. As the overall performance of the two variants are statistically indistinguishable, we recommend Simple CNAPS over Simple AR-CNAPS as it has fewer parameters.

Comparison with other distance metrics To test the significance of our choice of Mahalanobis distance we substitute it within our architecture with other distance metrics - absolute difference (L_1), squared Euclidean (L_2), and negative dot-product. Performance comparisons for in-domain datasets are shown in Table 3, and out-of-domain datasets and overall performance in Table 4. We observe that using the Mahalanobis distance results in the best in-domain, out-

	In-Domain Accuracy							
Model	ImageNet	Omniglot	Aircraft	Birds	DTD	QuickDraw	Fungi	Flower
MAML	32.4±1.0	71.9±1.2	52.8±0.9	47.2±1.1	56.7±0.7	50.5±1.2	21.0±1.0	70.9±1.0
RelationNet	30.9±0.9	86.6±0.8	69.7±0.8	54.1±1.0	56.6±0.7	61.8±1.0	32.6±1.1	76.1±0.8
k-NN	38.6±0.9	74.6±1.1	65.0±0.8	66.4±0.9	63.6±0.8	44.9±1.1	37.1±1.1	83.5±0.6
MatchingNet	36.1±1.0	78.3±1.0	69.2±1.0	56.4±1.0	61.8±0.7	60.8±1.0	33.7±1.0	81.9±0.7
Finetune	43.1±1.1	71.1±1.4	72.0±1.1	59.8±1.2	69.1±0.9	47.1±1.2	38.2±1.0	85.3±0.7
ProtoNet	44.5±1.1	79.6±1.1	71.1±0.9	67.0±1.0	65.2±0.8	64.9±0.9	40.3±1.1	86.9±0.7
ProtoMAML	47.9±1.1	82.9±0.9	74.2±0.8	70.0±1.0	67.9±0.8	66.6±0.9	42.0±1.1	88.5±0.7
CNAPS	51.3±1.0	88.0±0.7	76.8±0.8	71.4±0.9	62.5±0.7	71.9±0.8	**46.0±1.1**	89.2±0.5
AR-CNAPS	52.3±1.0	88.4±0.7	80.5±0.6	72.2±0.9	58.3±0.7	72.5±0.8	**47.4±1.0**	86.0±0.5
Simple AR-CNAPS	<u>**56.5±1.1**</u>	<u>**91.1±0.6**</u>	<u>81.8±0.8</u>	<u>**74.3±0.9**</u>	<u>**72.8±0.7**</u>	<u>75.2±0.8</u>	45.6±1.0	<u>**90.3±0.5**</u>
Simple CNAPS	<u>**58.6±1.1**</u>	<u>**91.7±0.6**</u>	<u>**82.4±0.7**</u>	<u>**74.9±0.8**</u>	67.8±0.8	<u>**77.7±0.7**</u>	46.9±1.0	<u>**90.7±0.5**</u>

Table 1: In-domain few-shot classification accuracy of Simple CNAPS and Simple AR-CNAPS compared to the baselines. All values are percentages and the error bars represent a 95% confidence interval over tasks. *Best performance(s) on each dataset is/are in bold, while values underlined are better than all baselines (excluding the non-corresponding CNAPS baseline) by statistically significant margins.*

	Out-of-Domain Accuracy					Average Accuracy		
Model	Signs	MSCOCO	MNIST	CIFAR10	CIFAR100	In-Domain	Out-Domain	Overall
MAML	34.2±1.3	24.1±1.1	NA	NA	NA	50.4±1.0	29.2±1.2	46.2±1.1
RelationNet	37.5±0.9	27.4±0.9	NA	NA	NA	58.6±0.9	32.5±0.9	53.3±0.9
k-NN	40.1±1.1	29.6±1.0	NA	NA	NA	59.2±0.9	34.9±1.1	54.3±0.9
MatchingNet	55.6±1.1	28.8±1.0	NA	NA	NA	59.8±0.9	42.2±1.1	56.3±1.0
Finetune	66.7±1.2	35.2±1.1	NA	NA	NA	60.7±1.1	51.0±1.2	58.8±1.1
ProtoNet	46.5±1.0	39.9±1.1	74.3±0.8	66.4±0.7	54.7±1.1	64.9±1.0	56.4±0.9	61.6±0.9
ProtoMAML	52.3±1.1	41.3±1.0	NA	NA	NA	67.5±0.9	46.8±1.1	63.4±0.9
CNAPS	60.1±0.9	42.3±1.0	88.6±0.5	60.0±0.8	48.1±1.0	69.6±0.8	59.8±0.8	65.9±0.8
AR-CNAPS	60.2±0.9	42.9±1.1	92.7±0.4	61.5±0.7	50.1±1.0	69.7±0.8	61.5±0.8	66.5±0.8
Simple AR-CNAPS	<u>**74.7±0.7**</u>	44.3±1.1	<u>**95.7±0.3**</u>	69.9±0.8	53.6±1.0	<u>**73.5±0.8**</u>	67.6±0.8	<u>**71.2±0.8**</u>
Simple CNAPS	<u>**73.5±0.7**</u>	<u>**46.2±1.1**</u>	93.9±0.4	<u>**74.3±0.7**</u>	<u>**60.5±1.0**</u>	<u>**73.8±0.8**</u>	<u>**69.7±0.8**</u>	<u>**72.2±0.8**</u>

Table 2: Middle) Out-of-domain few-shot classification accuracy of Simple CNAPS and Simple AR-CNAPS compared to the baselines. Right) In-domain, out-of-domain and overall mean classification accuracy of Simple CNAPS and Simple AR-CNAPS compared to the baselines. All values are percentages and the error bars represent a 95% confidence interval over tasks. *Best performance(s) on each dataset is/are in bold, while values underlined are better than baselines (excluding the non-corresponding CNAPS baseline) by statistically significant margins.*

of-domain, and overall average performance on all datasets.

Impact of the task regularizer Σ^τ We also consider a variant of Simple CNAPS where all-classes-within-task co-variance matrix Σ^τ is not included in the covariance regularization (denoted with the "-TR" tag). As shown in Table 4, we observe that, while removing the task level regularizer only marginally reduces overall performance, the difference on individual datasets such as ImageNet can be large.

Sensitivity to the number of support examples per class Figure 6 shows how the overall classification accuracy varies as a function of the average number of support examples per class (shots) over all tasks. We compare Simple CNAPS, original CNAPS, and the squared Euclidean variant of our method. As expected, the average number of sup-

port examples per class is highly correlated with the performance. All methods perform better with more labeled examples per support class, with Simple CNAPS performing substantially better as the number of shots increases. The surprising discovery is that Simple CNAPS is effective even when the number of labeled instances is as low as four, suggesting both that even poor estimates of the task and class specific covariance matrices are helpful and that the regularization scheme we have introduced works remarkably well.

Sensitivity to the number of classes in the task In Figure 7, we examine average accuracy as a function of the number of classes in the task. We find that, irrespective of the number of classes in the task, we maintain accuracy improvement over both CNAPS and our squared Euclidean norm variant.

| Metric | In-Domain Accuracy | | | | | | | |
	ImageNet	Omniglot	Aircraft	Birds	DTD	QuickDraw	Fungi	Flower
Negative Dot Product	48.0±1.1	83.5±0.9	73.7±0.8	69.0±1.0	66.3±0.6	66.5±0.9	39.7±1.1	88.6±0.5
Absolute Distance (L_1)	53.6±1.1	**90.6±0.6**	81.0±0.7	**73.2±0.9**	61.1±0.7	74.1±0.8	**47.0±1.0**	87.3±0.6
Squared Euclidean ($L_2{}^2$)	53.9±1.1	**90.9±0.6**	81.8±0.7	73.1±0.9	64.4±0.7	74.9±0.8	45.8±1.0	88.8±0.5
Simple CNAPS -TR	**56.7±1.1**	**91.1±0.7**	83.0±0.7	74.6±0.9	<u>70.2±0.8</u>	76.3±0.9	46.4±1.0	90.0±0.6
Simple CNAPS	<u>**58.6±1.1**</u>	**91.7±0.6**	82.4±0.7	74.9±0.8	67.8±0.8	<u>**77.7±0.7**</u>	46.9±1.0	<u>**90.7±0.5**</u>

Table 3: In-domain few-shot classification accuracy of Simple CNAPS compared to ablated alternatives of the negative dot product, absolute difference (L_1), squared Euclidean ($L_2{}^2$) and removing task regularization ($\lambda_k^\tau = 1$) denoted by "-TR". All values are percentages and the error bars represent a 95% confidence interval over tasks. *Best performance(s) on each dataset is/are in bold, while values underlined are better than all ablations by statistically significant margins.*

| Metric | Out-of-Domain Accuracy | | | | | Average Accuracy | | |
	Signs	MSCOCO	MNIST	CIFAR10	CIFAR100	In-Domain	Out-Domain	Overall
Negative Dot Product	53.9±0.9	32.5±1.0	86.4±0.6	57.9±0.8	38.8±0.9	66.9±0.9	53.9±0.8	61.9±0.9
Absolute Distance (L_1)	66.4±0.8	44.7±1.0	88.0±0.5	70.0±0.8	57.9±1.0	71.0±0.8	65.4±0.8	68.8±0.8
Squared Euclidean ($L_2{}^2$)	68.5±0.7	43.4±1.0	91.6±0.5	70.5±0.7	57.3±1.0	71.7±0.8	66.3±0.8	69.6±0.8
Simple CNAPS -TR	**74.1±0.6**	**46.9±1.1**	**94.8±0.4**	73.0±0.8	59.2±1.0	<u>73.5±0.8</u>	<u>**69.6±0.8**</u>	<u>**72.0±0.8**</u>
Simple CNAPS	**73.5±0.7**	**46.2±1.1**	<u>93.9±0.4</u>	**74.3±0.7**	**60.5±1.0**	<u>**73.8±0.8**</u>	<u>**69.7±0.8**</u>	<u>**72.2±0.8**</u>

Table 4: Middle) Out-of-domain few-shot classification accuracy of Simple CNAPS compared to ablated alternatives of the negative dot product, absolute difference (L_1), squared Euclidean ($L_2{}^2$) and removing task regularization ($\lambda_c = 1$) denoted by "-TR". Right) In-domain, out-of-domain and overall mean classification accuracies of the ablated models. All values are percentages and the error bars represent a 95% confidence interval over tasks. *Best performance(s) on each dataset is/are in bold, while values underlined are better than all ablations by statistically significant margins.*
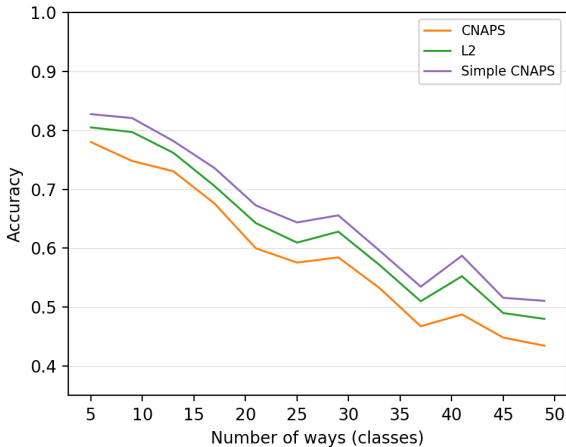

Figure 7: **Accuracy vs. Ways:** Number of ways (classes in the task) v/s accuracy. Tasks in the test set are grouped together by number of classes. The accuracies are averaged to obtain a value for each count of class. Note that sqaured Euclidean has been referred to as L2.

7. Discussion

Few shot learning is a fundamental task in modern AI research. In this paper we have introduced a new method for amortized few shot image classification which estab-
lishes a new SoTA performance benchmark by making a simplification to the current SoTA architecture. Our specific architectural choice, that of deterministically estimating and using Mahalanobis distances for classification of task-adjusted class-specific feature vectors seems to produce, via training, embeddings that generally allow for useful covariance estimates, even when the number of labeled instances per task and class is small. The effectiveness of the Mahalanobis distance in feature space for distinguishing classes suggests connections to hierarchical regularization schemes [28] that could enable performance improvements even in the zero-shot setting. In the future, exploration of other Bregman divergences can be an avenue of potentially fruitful research. Additional enhancements in the form of data and task augmentation can also boost performance.

8. Acknowledgements

References

[1] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of machine learning research*, 6(Oct):1705–1749, 2005. 2, 3, 5

[2] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3606–3613, 2014. 6, 12

[3] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017. 3

[4] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. A. Ranzato, and T. Mikolov. Devise: A deep visual-semantic embedding model. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2121–2129. Curran Associates, Inc., 2013. 1

[5] P. Galeano, E. Joseph, and R. E. Lillo. The mahalanobis distance for functional data with applications to classification. *Technometrics*, 57(2):281–291, 2015. 2

[6] M. Garnelo, D. Rosenbaum, C. J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. M. A. Eslami. Conditional neural processes. *CoRR*, abs/1807.01613, 2018. 3

[7] S. Gidaris and N. Komodakis. Generating classification weights with gnn denoising autoencoders for few-shot learning. *arXiv preprint arXiv:1905.01102*, 2019. 1

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 4, 12

[9] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga. A comprehensive survey of deep learning for image captioning. *ACM Comput. Surv.*, 51(6):118:1–118:36, Feb. 2019. 1

[10] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. 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. 6

[11] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu. A survey of deep learning-based object detection. *CoRR*, abs/1907.09408, 2019. 1

[12] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The quick, draw!-ai experiment.(2016), 2016. 6, 12

[13] J. Kim, T. Kim, S. Kim, and C. D. Yoo. Edge-labeling graph neural network for few-shot learning. *CoRR*, abs/1905.01436, 2019. 1, 2

[14] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015. 1, 2

[15] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 6

[16] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. 6, 12

[17] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. 6

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 6

[19] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 6, 12

[20] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017. 3

[21] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. 3

[22] M.-E. Nilsback and A. 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. 6, 12

[23] B. Oreshkin, P. Rodríguez López, and A. Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 721–731. Curran Associates, Inc., 2018. 2, 3

[24] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2, 3, 4, 11

[25] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. 3

[26] J. Requeima, J. Gordon, J. Bronskill, S. Nowozin, and R. E. Turner. Fast and flexible multi-task classification using conditional neural adaptive processes. *arXiv preprint arXiv:1906.07697*, 2019. 1, 2, 3, 4, 5, 6, 11, 12, 14

[27] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 4, 6, 12

[28] R. Salakhutdinov, J. Tenenbaum, and A. Torralba. One-shot learning with a hierarchical nonparametric bayesian model. In I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 195–206, Bellevue, Washington, USA, 02 Jul 2012. PMLR. 8

[29] V. G. Satorras and J. B. Estrach. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018. 1, 2

[30] B. Schroeder and Y. Cui. Fgvcx fungi classification challenge 2018. https://github.com/visipedia/fgvcx_fungi_comp, 2018. 6, 12

[31] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017. 1, 2, 3, 5

[32] M. Sornam, K. Muthusubash, and V. Vanitha. A survey on image classification and activity recognition using deep con-

volutional neural network architecture. In *2017 Ninth International Conference on Advanced Computing (ICoAC)*, pages 121–126, Dec 2017. 1

[33] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018. 2

[34] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019. 6, 11

[35] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016. 1, 2

[36] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011. 6, 12

[37] W. Wang, V. W. Zheng, H. Yu, and C. Miao. A survey of zero-shot learning: Settings, methods, and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2):13:1–13:37, Jan. 2019. 1

[38] Y. Wang and Q. Yao. Few-shot learning: A survey. *CoRR*, abs/1904.05046, 2019. 1, 2

[39] C. Xing, N. Rostamzadeh, B. N. Oreshkin, and P. O. Pinheiro. Adaptive cross-modal few-shot learning. *CoRR*, abs/1902.07104, 2019. 1

[40] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014. 2

Appendices

A. (Simple) CNAPS in Details

A.1. Auto-Regressive CNAPS

In [26], an additional auto-regressive variant for adapting the feature extractor is proposed, referred to as AR-CNAPS. As shown in Figure 9, AR-CNAPS extends CNAPS by introducing the block-level set encoder $g_\phi^{AR_j}$ at each block j. These set encoders use the output obtained by pushing the support $\mathcal{S}^\tau$ through all previous blocks $1 : j - 1$ to form the block level set representation $g_\phi^{AR_j}(f_\theta^{\tau_j}(\mathcal{S}^\tau))$. This representation is then subsequently used as input to the adaptation network ψ_ϕ^j in addition to the task representation $g_\phi(\mathcal{S}^\tau)$. This way the adaptation network is not just conditioned on the task, but is also aware of the potential changes in the previous blocks as a result of the adaptation being performed by the adaptation networks before it (*i.e.*, $\psi_\phi^1 : \psi_\phi^{j-1}$). The auto-regressive nature of AR-CNAPS allows for a more dynamic adaptation procedure that boosts performance in certain domains.

A.2. FiLM Layers

Proposed by [24], Feature-wise Linear Modulation (FiLM) layers were used for visual question answering, where the feature extractor could be conditioned on the question. As shown in Figure 10, these layers are inserted within residual blocks, where the feature channels are scaled and linearly shifted using the respective FiLM parameters $\gamma_{i,ch}$ and $\beta_{i,ch}$. This can be extremely powerful
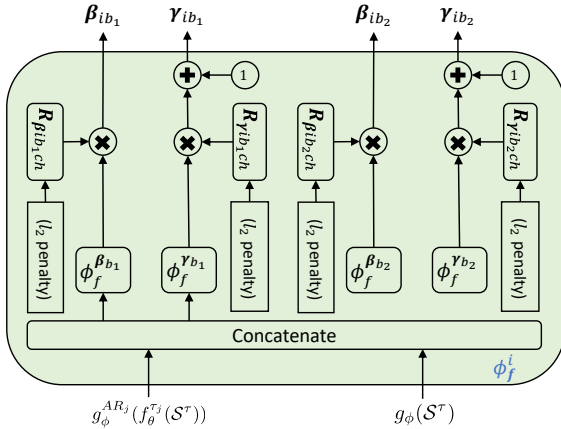

Figure 8: **Architectural overview of the feature extractor adaptation network ψ_ϕ^f:** Figure has been adapted from [26] and showcases the neural architecture used for each adaptation module ψ_ϕ^j (corresponding to residual block j) in the feature extractor adaptation network ψ_ϕ^f.

in transforming the extracted feature space. In our work and [26], these FiLM parameters are conditioned on the support images in the task $\mathcal{S}^\tau$. This way, the adapted feature extractor f_θ^τ is able to modify the feature space to extract the features that allow classes in the task to be distinguished most distinctly. This is in particular very powerful when the classification metric is changed to the Mahalanobis distance, as with a new objective, the feature extractor adaptation network ψ_ϕ^f is able to learn to extract better features (see difference between with and without ψ_ϕ^f in Table 6 on CNAPS and Simple CNAPS).

A.3. Network Architectures

We adapt the same architectural choices for the task encoder g_ϕ, auto-regressive set encoders $g_\phi^{AZ_1}, ..., g_\phi^{AZ_J}$ and the feature extractor adaptation network $\psi_\phi^f = \{\psi_\phi^1, ..., \psi_\phi^J\}$ as [26]. The neural architecture for each adaptation module inside of ψ_ϕ^f has been shown in Figure 8. The neural configurations for the task encoder g_ϕ and the auto-regressive set encoders $g_\phi^{AZ_1}, ..., g_\phi^{AZ_J}$ used in AR-CNAPS are shown in Figure 11-a and Figure 11-b respectively. Note that for the auto-regressive set encoders, there is no need for convolutional layers. The input to these networks come from the output of the corresponding residual block adapted to that level (denoted by $f_\theta^{\tau_j}$ for block j) which has already been processed with convolutional filters.

Unlike CNAPS, we do not use the classifier adaptation network ψ_ϕ^c. As shown in Figure 11-c, the classification weights adaptor ψ_ϕ^c consists of an MLP consisting of three fully connected (FC) layers with the intermediary nonelinearity ELU, which is the continuous approximation to ReLU as defined below:

$$ELU(x) = \begin{Bmatrix} x & x > 0 \\ e^x 1 & x \leq 0 \end{Bmatrix} \quad (6)$$

As mentioned previously, without the need to learn the three FC layers in ψ_ϕ^c, Simple CNAPS has 788,485 fewer parameters while outperforming CNAPS by considerable margins.

B. Cross Validation

The Meta-Dataset [34] and its 8 in-domain 2 out-of-domain split is a setting that has defined the benchmark for the baseline results provided. The splits, between the datasets, were intended to capture an extensive set of visual domains for evaluating the models.

However, despite the fact that all past work directly rely on the provided set up, we go further by verifying that our model is not overfitting to the proposed splits and is able to consistently outperform the baseline with different permutations of the datasets. We examine this through a 4-fold cross validation of Simple CNAPS and CNAPS on the
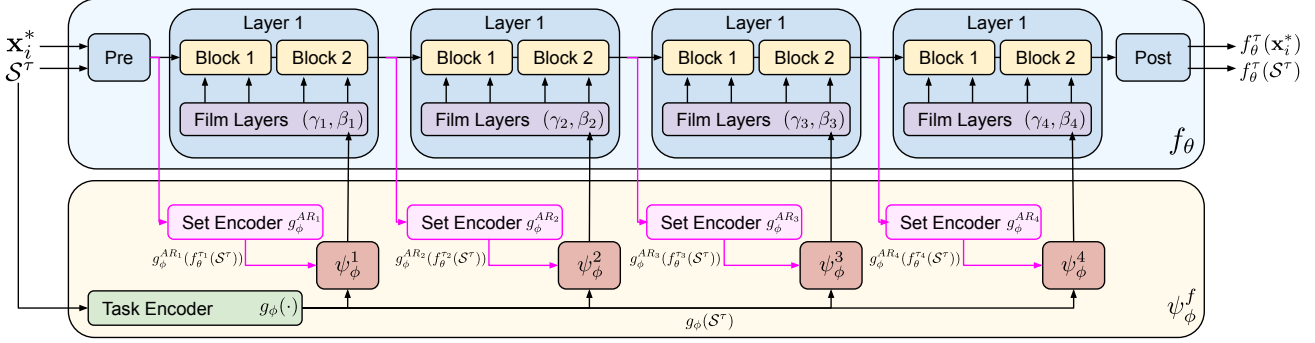
Figure 9: **Overview of the auto-regresive feature extractor adaptation in CNAPS:** in addition to the structure shown in Figure 4, AR-CNAPS takes advantage of a series of pre-block set encoders $g_\phi^{AR_j}$ to furthermore condition the output of each ψ_ϕ^j on the set representation $g_\phi^{AR_j}(f_\theta^{\tau_j}(\mathcal{S}^\tau))$. The set representation is formed by first adapting the previous blocks $1 : j-1$, then pushing the support set $\mathcal{S}$ through the adapted blocks to form an auto-regressive adapted set representation at block j. This way, adaptive functions later in the pipeline are more explicitly aware of the changes made by the previous adaptation networks, and can adjust better accordingly.
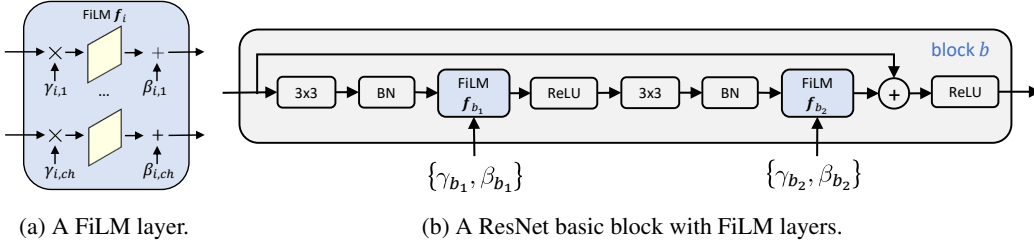

(a) A FiLM layer.   (b) A ResNet basic block with FiLM layers.

Figure 10: **Overview of FiLM Layers:** Figure is from [26]. Left) FiLM layer operating a series of channels indexed by ch, scaling and shifting the feature channels as defined by the respective FiLM parameters $\gamma_{i,ch}$ and $\beta_{i,ch}$. Right) Placement of these FiLM modules within a ResNet18 [8] basic block.

following 8 datasets: ILSVRC-2012 (ImageNet) [27], Omniglot [16], FGVC-Aircraft [19], CUB-200-2011 (Birds) [36], Describable Textures (DTD) [2], QuickDraw [12], FGVCx Fungi [30] and VGG Flower [22]. During each fold, two of the datasets are exluded from training, and both Simple CNAPS and CNAPS are trained and evaluated in that setting.

As shown by the classification results in Table 5, in all four folds of validation, Simple CNAPS is able to outperform CNAPS on 7-8 out of the 8 datasets. The in-domain, out-of-domain, and overall averages for each fold noted in Table 7 also show Simple CNAPS's accuracy gains over CNAPS with substantial margins. In fact, the fewer number of in-domain datasets in the cross-validation (6 vs. 8) actually leads to wider gaps between Simple CNAPS and CNAPS. This suggests Simple CNAPS is a more powerful alternative in the low domain setting. Furthermore, using these results, we illustrate that our gains are not specific to the Meta-Dataset setup.

C. Ablation study of the Feature Extractor Adaptation Network

In addition to the choice of metric ablation study referenced in Section 6.1, we examine the behaviour of the model when the feature extractor adaptation network ψ_ϕ^f has been turned off. In such setting, the feature extractor would only consist of the pre-trained ResNet18 [8] f_θ. Consistent to [26], we refer to this setting as "No Adaptation" (or "No Adapt" for short). We compare the "No Adapt" variation to the feature extractor adaptive case for each of the metrics/model variations examined in Section 6.1. The in-domain, out-of-domain and overall classification accuracies are shown in Table 6. As shown, without ψ_ϕ^f all models lose approximately 15, 5, and 12 percentage points across in-domain, out-of-domain and overall accuracy, while Simple CNAPS continues to hold the lead especially in out-of-domain classification accuracy. It's interesting to note that without the task specific regularization term (denoted as "-TR"), there's a considerable perfor-

Model	Classification Accuracy (%)							
	ILSVRC	Omniglot	Aircraft	CUB	DTD	QuickDraw	Fungi	Flower
CNAPS	49.6±1.1	87.2±0.8	81.0±0.7	69.7±0.9	61.3±0.7	72.0±0.8	*32.2±1.0	*70.9±0.8
Simple CNAPS	**55.6±1.1**	**90.9±0.8**	82.2±0.7	**75.4±0.9**	**74.3±0.7**	**75.5±0.8**	***39.9±1.0**	***88.0±0.8**
CNAPS	50.3±1.1	86.5±0.8	77.1±0.7	71.6±0.9	*64.3±0.7	*33.5±0.9	46.4±1.1	84.0±0.6
Simple CNAPS	**58.1±1.1**	90.8±0.8	**83.8±0.7**	75.2±0.9	***74.6±0.7**	***64.0±0.9**	47.7±1.1	**89.9±0.6**
CNAPS	51.5±1.1	87.8±0.8	*38.2±0.8	*58.7±1.0	62.4±0.7	72.5±0.8	46.9±1.1	89.4±0.5
Simple CNAPS	**56.0±1.1**	**91.1±0.8**	***66.6±0.8**	***68.0±1.0**	71.3±0.7	**76.1±0.8**	45.6±1.1	90.7±0.5
CNAPS	*42.4±0.9	*59.6±1.4	77.2±0.8	69.3±0.9	62.9±0.7	69.1±0.8	40.9±1.0	88.2±0.5
Simple CNAPS	***49.1±0.9**	***76.0±1.4**	**83.0±0.8**	74.5±0.9	74.4±0.7	74.8±0.8	44.0±1.0	91.0±0.5

Table 5: Cross-validated classification accuracy results. Note that * denotes that this dataset was excluded from training, and therefore, signifies out-of-domain performance. Simple CNAPS values in bold indicate significant statistical gains over CNAPS.

Metric/Model Variation	Average Accuracy with ψ_ϕ^f (%)			Average Accuracy without ψ_ϕ^f (%)		
	In-Domain	Out-Domain	Overall	In-Domain	Out-Domain	Overall
Negative Dot Product	66.9±0.9	53.9±0.8	61.9±0.9	38.4±1.0	44.7±1.0	40.8±1.0
CNAPS	69.6±0.8	59.8±0.8	65.9±0.8	**54.4±1.0**	55.7±0.9	54.9±0.9
Absolute Distance (L_1)	71.0±0.8	65.4±0.8	68.8±0.8	**54.9±1.0**	62.2±0.8	57.7±0.9
Squared Euclidean (L_2^2)	71.7±0.8	66.3±0.8	69.6±0.8	**55.3±1.0**	61.8±0.8	**57.8±0.9**
Simple CNAPS -TR	**<u>73.5±0.8</u>**	**<u>69.6±0.8</u>**	**<u>72.0±0.8</u>**	52.3±1.0	<u>61.7±0.9</u>	55.9±1.0
Simple CNAPS	**<u>73.8±0.8</u>**	**<u>69.7±0.8</u>**	**<u>72.2±0.8</u>**	56.0±1.0	<u>64.8±0.8</u>	<u>59.3±0.9</u>

Table 6: Comparing in-domain, out-of-domain and overall accuracy averages of each metric/model variation when feature extractor adaptation is performed (denoted as "with ψ_ϕ^f") vs. when no adaptation is performed (denoted as "without ψ_ϕ^f"). Values in bold signify best performance in the column while underlined values signify superior performance of Simple CNAPS (and the -TR variation) compared to the CNAPS baseline.

Fold	Model	Average Classification Accuracy (%)		
		In-Domain	Out-Domain	Overall
1	CNAPS	70.1±0.4	51.6±0.4	65.5±0.4
1	S. CNAPS	**75.7±0.3**	**64.0±0.4**	**72.7±0.3**
2	CNAPS	69.3±0.4	48.9±0.3	64.2±0.4
2	S. CNAPS	**74.3±0.4**	**69.3±0.4**	**73.0±0.3**
3	CNAPS	68.4±0.4	48.5±0.4	63.4±0.4
3	S. CNAPS	**71.8±0.4**	**67.3±0.5**	**70.7±0.4**
4	CNAPS	67.9±0.3	51.0±0.7	63.7±0.4
4	S. CNAPS	**73.6±0.3**	**62.6±0.6**	**70.9±0.4**
Avg	CNAPS	69.0±1.4	50.0±1.8	64.2±1.6
Avg	S. CNAPS	**73.8±1.3**	**65.8±1.8**	**71.8±1.4**

Table 7: Cross-validated in-domain, out-of-domain and overall classification accuracies averaged across each fold and combined. Note that for conciseness of the table, Simple CNAPS has been shortened to "S. CNAPS". Simple CNAPS values in bold indicate statistically significant gains over CNAPS.

mance drop in the "No Adaptation" setting; while when the feature extractor adaptation network ψ_ϕ^f is present, the difference is marginal. This signifies two important observations. First, it shows the importance of of learning the feature extractor adaptation module end-to-end with the Ma-

halanobis distance, as it's able adapt the feature space best suited for using the squared Mahalanobis distance. Second, the adaptation function ψ_ϕ^f can reduce the importance of the task regularizer by properly de-correlating and normalizing variance within the feature vectors. However, where this is not possible, as in the "No Adaptation" case, the all-classes-task-level covariance estimate as an added regularizer in Equation 2 becomes crucial in maintaining superior performance.

D. Projection Networks

We additionally explored metric learning where in addition to changing the distance metric, we considered projecting each support feature vector $f_\theta^\tau(\mathbf{x}_i)$ and query vector $f_\theta^\tau(\mathbf{x}_i^*)$ to a new decision space where then squared Mahalanobis distance was to be used for classification. Specifically, we trained a projection network u_ϕ such that for Equations 2 and 3, $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k^\tau$ and $\boldsymbol{\Sigma}^\tau$ were calculated based on the projected feature vectors $\{u_\phi(f_\theta^\tau(\mathbf{x}_i))\}_{\mathbf{x}_i \in \mathcal{S}_k^\tau}$ as oppose to the feature vector set $\{f_\theta^\tau(\mathbf{x}_i)\}_{\mathbf{x}_i \in \mathcal{S}_k^\tau}$. Similarly, the projected query feature vector $u_\phi(f_\theta^\tau(\mathbf{x}_i^*))$ was used for classifying the query example as oppose to the bare feature vector $f_\theta^\tau(\mathbf{x}_i^*)$ used within Simple CNAPS. We define u_ϕ in our
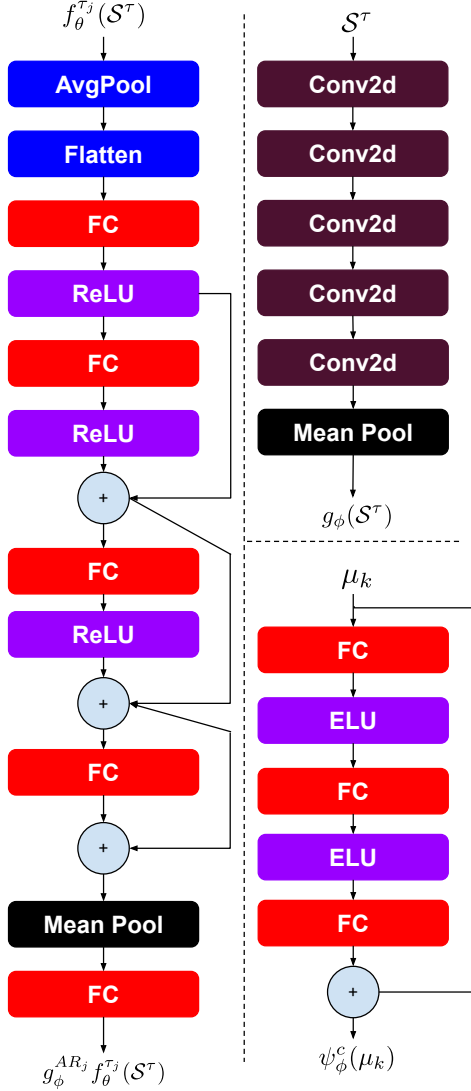
	Average Classification Accuracy (%)		
Model	In-Domain	Out-Domain	Overall
Simple CNAPS +P	72.4±0.9	67.1±0.8	70.4±0.8
Simple CNAPS	73.8±0.8	**69.7±0.8**	72.2±0.8

Table 8: Comparing the in-domain, out-of-domain and overall classification accuracy of Simple CNAPS +P (with projection networks) to Simple CNAPS. Values in bold show the statistically significant best result.

We refer to this variation of our model as "Simple CNAPS +P" with the "+P" tag signifying the addition of the projection function u_ϕ. The results for this variation of Simple CNAPS are compared to the base Simple CNAPS in Table 8. As shown, the projection network generally results in lower performance, although not to statistically significant degrees in in-domain and overall accuracies. Where the addition of the projection network results in substantial loss of performance is in the out-of-domain setting with Simple CNAPS +P's average accuracy of 67.1±0.8 compared to 69.7±0.8 for the Simple CNAPS. We hypothesize the significant loss in out-of-domain performance to be due to the projection network overfitting to the in-domain datasets.

Figure 11: **Overview of Neural Architectures used in (Simple) CNAPS:** a) Auto-regressive set encoder $g_\phi^{AR_j}$. Note that since this is conditioned on the channel outputs of the convolutional filter, it's not convolved any further. b) Task encoder g_ϕ that mean-pools convolutionally filtered support examples to produce the task representation. c) architectural overview of the classifier adaptation network ψ_ϕ^c consisting of a three layer MLP with a residual connection. Three diagrams are based on Table E.8, Table E.9, and Table E.11 in [26].

experiments to be the following:

$$u_\phi(f_\theta^\tau(\mathbf{x}_i^*)) = W_1(ELU(W_2(ELU(W_3 f_\theta^\tau(\mathbf{x}_i^*))))) \quad (7)$$

where ELU, a continuous approximation to ReLU as previously noted, is used as the choice of non-linearity and W_1, W_2 and W_3 are learned parameters.