Supplemental Material for 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

Appendices

A. Experimental Setting

Section 3.2 of [39] explains the sampling procedure to generate tasks from the Meta-Dataset [39], used during both training and testing. This results in tasks with varying of number of shots/ways. Figure 9a and 9b show the ways/shots frequency graphs at test time. For evaluating on Meta-Dataset and mini/tiered-ImageNet datasets, we use *episodic training* [36] to train models to remain consistent with the prior works [3, 30, 36, 39]. We train for 110K tasks, 16 tasks per batch, totalling 6,875 gradient steps using Adam with learning rate of 0.0005. We validate (on 8 in-domain and 1 out-of-domain datasets) every 10K tasks, saving the best model/checkpoint for testing. Please visit the Pytorch implementation of Simple CNAPS for details.

B. (Simple) CNAPS in Details

B.1. Auto-Regressive CNAPS

In [30], an additional auto-regressive variant for adapting the feature extractor is proposed, referred to as AR-CNAPS.

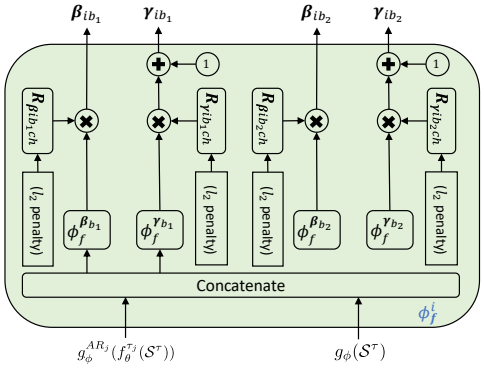
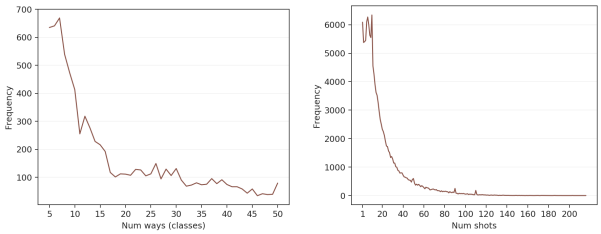


Figure 8: **Architectural overview of the feature extractor adaptation network ψ_ϕ^f:** Figure has been adapted from [30] and showcases the neural architecture used for each adaptation module ψ_ϕ^j (corresponding to residual block j) in the feature extractor adaptation network ψ_ϕ^f.



(a) Number of Tasks vs. Ways (b) Number of Classes vs. Shots

Figure 9: **Test-time distribution of tasks:** a) Frequency of number of tasks as grouped by the number of classes in the tasks (ways). b) Frequency of the number of classes grouped by the number examples per class (shots). Both figures are for test tasks sampled when evaluating on the Meta-Dataset [39].

As shown in Figure 10, 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.

B.2. FiLM Layers

Proposed by [27], 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 11, 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 in transforming the extracted feature space. In our work and

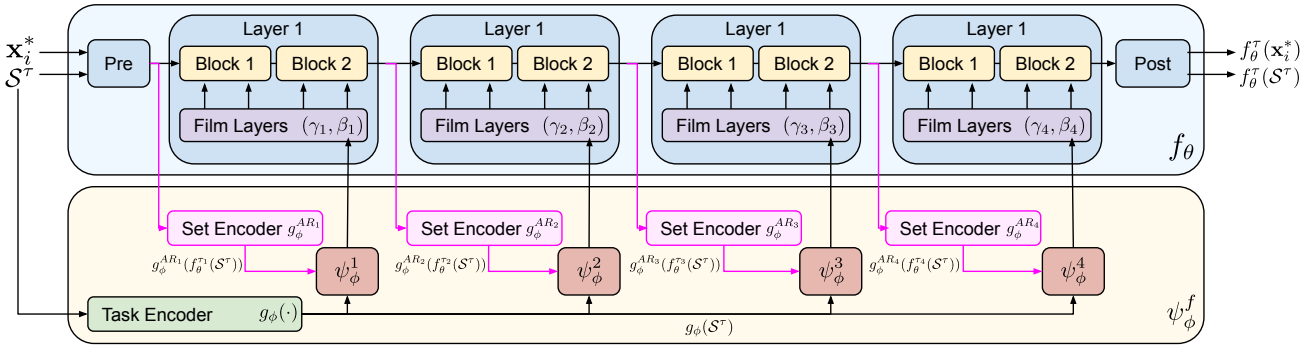


Figure 10: **Overview of the auto-regresive feature extractor adaptation in CNAPS:** in addition to the structure shown in Figure 3, AR-CNAPS takes advantage of a series of pre-block set encoders $g_\phi^{AR_j}$ to furthermore condition the output of each ψ_ϕ^j on the set representation $g_\phi^{AR_j}(f_\theta^{\tau_j}(\mathcal{S}^\tau))$. 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.

[30], 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 7 on CNAPS and Simple CNAPS).

B.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 [30]. 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 12-a and Figure 12-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 12-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:

$$ELU(x) = \begin{cases} x & x > 0 \\ e^x - 1 & x \leq 0 \end{cases} \tag{6}$$

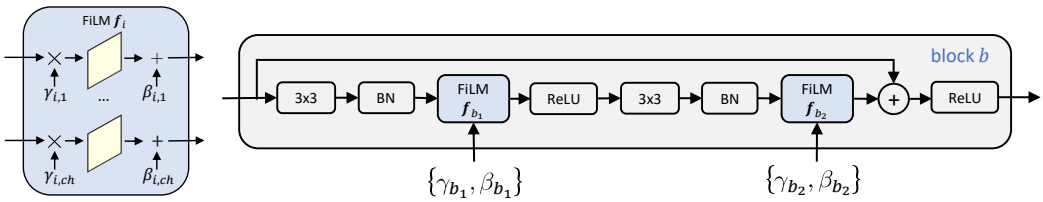
As mentioned previously, without the need to learn the three FC layers in ψ_ϕ^c, Simple CNAPS has 788,485 fewer parameters while outperforming CNAPS by considerable margins.

C. Cross Validation

The Meta-Dataset [39] 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 following 8 datasets: ILSVRC-2012 (ImageNet) [31], Omniglot [18], FGVC-Aircraft [22], CUB-200-2011 (Birds) [41], Describable Textures (DTD) [2], QuickDraw [14], FGVCx Fungi [35] and VGG Flower [25]. 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 6, 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 8 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



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

Figure 11: **Overview of FiLM layers:** Figure is from [30]. 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 [10] basic block.

	Classification Accuracy (%)							
Model	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 6: 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.

these results, we illustrate that our gains are not specific to the Meta-Dataset setup.

D. Ablation study of the Feature Extractor Adaptation Network

In addition to the choice of metric ablation study referenced in Section 6.2, 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 [10] f_θ. Consistent to [30], we refer to this setting as "No Adaptation" (or "No Adapt" for short). We compare the "No Adapt" variant to the feature extractor adaptive case for each of the metrics/model variants examined in Section 6.2. The in-domain, out-of-domain and overall classification accuracies are shown in Table 7. 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 performance 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 Mahalanobis 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.

E. 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 experiments to be the following:

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

Metric/Model Variant	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	**73.5±0.8**	**69.6±0.8**	**72.0±0.8**	52.3±1.0	61.7±0.9	55.9±1.0
Simple CNAPS	**73.8±0.8**	**69.7±0.8**	**72.2±0.8**	**56.0±1.0**	**64.8±0.8**	**59.3±0.9**

Table 7: Comparing in-domain, out-of-domain and overall accuracy averages of each metric/model variant 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 variant) compared to the CNAPS baseline.

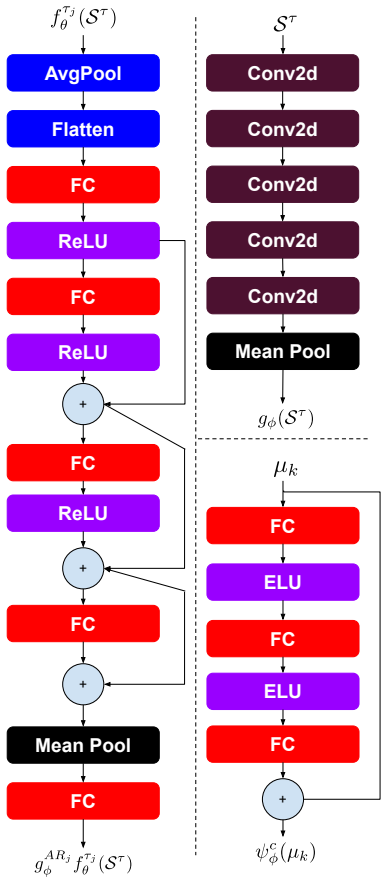


Figure 12: **Overview of architectures used in (Simple) CNAPS:** a) Auto-regressive set encoder $g_\phi^{AR_j}$. Note that since this is conditioned on the channel outputs of the convolutional filter, it's not convolved any further. b) Task encoder g_ϕ that mean-pools convolutionally filtered support examples to produce the task representation. c) architectural overview of the classifier adaptation network ψ_ϕ^c consisting of a 3 layer MLP with a residual connection. Diagrams are based on Table E.8, E.9, and E.11 in [30].

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.

We refer to this variant of our model as "Simple CNAPS +P" with the "+P" tag signifying the addition of the projection function u_ϕ. The results for this variant of Simple CNAPS are compared to the base Simple CNAPS in Table 9. 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.

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 8: 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.

Model	Average Classification Accuracy (%)		
	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 9: 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.