A Appendix

A.1 Related work

Pioneered by the seminal work of Utgoff (1986), Schmidhuber (1987, 1992), Bengio et al. (1992) and Thrun (1996), the general concept of meta-learning is several decades old (for a survey see Vilalta & Drissi (2002); Hospedales et al. (2020)). However, in the last few years it has experienced a surge in popularity and it has become the most used paradigm for learning from very few examples. Between 2016 and 2017, several methods addressing the FSL problem by learning on episodes were proposed. MANN (Santoro et al., 2016) uses a Neural Turing Machine (Graves et al., 2014) to save and access the information useful to meta-learn; Bertinetto et al. (2016) propose a deep network in which a "teacher" branch is tasked with predicting the parameters of a "student" branch; Matching Networks (Vinyals et al., 2016) and Prototypical Networks (Snell et al., 2017) are two non-parametric methods in which the contributions of different examples in the support set are weighted by either an LSTM or a simple average, respectively; Ravi & Larochelle (2017) propose instead to use an LSTM to learn the hyper-parameters of SGD, while MAML (Finn et al., 2017) learns to fine-tune an entire deep network by backpropagating through SGD. Despite these works widely differing in nature, they all stress on the importance of organising training in a series of small learning problems (*episodes*) that are similar to those encountered during inference at test time.

In contrast with this trend, a handful of papers have recently shown that simple approaches that forego episodes and meta-learning can perform well on FSL benchmarks. These methods all have in common that they pre-train a feature extractor with the cross-entropy loss on the "meta-training classes" of the dataset. Then, at test time a classifier is adapted to the support set by weight imprinting (Qi et al., 2018; Dhillon et al., 2020), fine-tuning (Chen et al., 2019), transductive fine-tuning (Dhillon et al., 2020) or logistic regression (Tian et al., 2020). Wang et al. (2019) suggest performing test-time classification by using the label of the closest centroid to the query image.

Differently from these papers, we try to shed some light on one of the possible causes behind the poor performance of episodic-based algorithms like Prototypical Networks. An analysis similar to ours in spirit is the one of Raghu et al. (2020). After showing that the efficacy of MAML in FSL is due to the adaptation of the final layer and the "reuse" of the features of previous layers, they propose a variant with the same accuracy and computational advantages. In this paper, we focused on an FSL algorithm just as popular and uncovered inefficiencies that allow for a notable conceptual simplification of Prototypical Networks, which surprisingly also brings a significant boost in performance.

A.2 Few-shot classification during evaluation

Once f_θ has been trained, there are many possible ways to perform few-shot classification during evaluation. In our experiments, we considered three simple approaches that are particularly suitable for embeddings learned via metric-based losses such as Eq. 2 or Eq. 3.

k-NN. To classify an image $\mathbf{q}_i \in Q$, we first compute the Euclidean distance to each support point $\mathbf{s}_j \in S$: $d_{ij} = \|f_\theta(\mathbf{q}_i) - f_\theta(\mathbf{s}_j))\|^2$. Then, we simply assign $y(\mathbf{q}_i)$ to be majority label of the k-nearest neighbours. A downside here is that k is a hyper-parameter that has to be chosen, although a reasonable choice in the FSL setup is to set it equal to the number of "shots" n.

1-NN with Class Centroids. Similar to k-NN, we can perform classification by inheriting the label of the closest class centroid, i.e. $y(\mathbf{q}_i) = \arg\min_{j \in \{1,\dots,k\}} \|f_\theta(\mathbf{x}_i) - \mathbf{c}_j\|$. This is the approach used at test-time by Snell et al. (2017) and Wang et al. (2019).

Soft Assignments. This is what the original NCA paper (Goldberger et al., 2005) used for evaluation. To classify an image $\mathbf{q}_i \in Q$, we compute the values $p_{ij} = \exp(-\|f_\theta(\mathbf{q}_i) - f_\theta(\mathbf{s}_j))\|^2) / \sum_{\mathbf{s}_k \in S} \exp(-\|f_\theta(\mathbf{q}_i) - f_\theta(\mathbf{s}_k)\|^2)$ for all $\mathbf{s}_j \in S$, which is the probability that image i is sampled from image j. We then compute the likelihood for each class k: $\sum_{s_j \in S_k} p_{ij}$, and choose the class with the highest likelihood $y(\mathbf{q}_i) = \arg\max_k \sum_{s_j \in S_k} p_{ij}$. This approach is the only one closely aligned with the training procedure, and has a direct probabilistic interpretation.

We also experimented with using the NCA loss on the support set to perform adaptation at test time, which we discuss in Appendix A.3. We compared the inference methods on *mini*ImageNet and

CIFAR-FS. Results can be found in Table 2. We chose to use 1-NN with class centroids in all our experiments, as it performs significantly better than k-NN or Soft Assignment. This might sound surprising, as the Soft Assignment approach closely reflects the training protocol and outputs class probabilities. We speculate its inferior performance could be caused by poor model calibration (Guo et al., 2017), which for NCA would affect the contribution of the images of the support set. However, it is not immediately clear how one could use the tools provided by the calibration literature given that the set of classes between dataset splits are disjoint.

A.3 Adapting to the support set

Prototypical Networks does not perform any kind of parameter adaptation at test time. On the one hand this is convenient, as it allows fast inference; on the other hand, useful information from the support set S might remain unexploited.

In the 5-shot case it is possible to minimise the NCA loss since it can directly be computed on the support set: $\mathcal{L}_{\text{NCA}}(S)$. We tried training a positive semi-definite matrix A on the outputs of the trained neural network, which corresponds to learning a Mahalanobis distance metric as in Goldberger et al. (2005). However, we found that there was no meaningful increase in performance. Differently, we did find that fine-tuning the whole neural network f_θ by $\arg\min_\theta \mathcal{L}_{\text{NCA}}(S)$ brought a slight improvement. However, given the computational cost, we opted for non performing adaptation to the support sets in our experiments of Sec. 3. Results on *mini*ImageNet an CIFAR-FS are shown in Table 1.

A.4 Details about the ablation studies of Section 3.3

Referring to the three key differences between the Prototypical Networks and the NCA losses listed in Sec. 2.3, in this section we detail how to obtain the ablations we used to perform the experiments of Sec. 3.3.

We can "disable" the creation of prototypes (point 1), which will change the prototypical loss of Eq. 2 to

$$\mathcal{L}(S,Q) = -\frac{1}{|Q|+|S|} \sum_{(\mathbf{q}_i,y)\in Q} \log \left(\frac{\sum\limits_{(\mathbf{s}_j,y')\in S_y} \exp -\|\mathbf{q}_i - \mathbf{s}_j\|^2}{\sum\limits_{(\mathbf{s}_k,y'')\in S} \exp -\|\mathbf{q}_i - \mathbf{s}_k\|^2} \right). \tag{4}$$

This is similar to $\mathcal{L}_{\text{NCA}}$ (Eq. 3), where the positives are represented by the distances from Q to S_k, and the negatives by the distances from Q to $S \setminus S_k$. The only difference now is the separation of the batch into a query and support set.

Independently, we can "disable" point 2, which gives us

$$\mathcal{L}(S,Q) = -\frac{1}{|Q|+|S|} \sum_{(\mathbf{z}_i,y_i)\in Q\cup C} \log \left(\frac{\sum\limits_{\substack{(\mathbf{z}_j,y_j)\in Q\cup C \\ y_j=y_i \\ i\neq j}} \exp -\|\mathbf{z}_i - \mathbf{z}_j\|^2}{\sum\limits_{\substack{(\mathbf{z}_k,y_k)\in Q\cup C \\ k\neq i}} \exp -\|\mathbf{z}_i - \mathbf{z}_k\|^2} \right), \tag{5}$$

which essentially combines the prototypes with the query set, and computes the NCA loss on that total set of embeddings.

Finally, we can "disable" both point 1 and 2, which gives us

$$\mathcal{L}(S,Q) = -\frac{1}{|Q|+|S|} \sum_{(\mathbf{z}_i,y_i)\in Q\cup S} \log \left(\frac{\sum\limits_{\substack{(\mathbf{z}_j,y_j)\in Q\cup S \\ y_j=y_i \\ i\neq j}} \exp -\|\mathbf{z}_i - \mathbf{z}_j\|^2}{\sum\limits_{\substack{(\mathbf{z}_k,y_k)\in Q\cup S \\ k\neq i}} \exp -\|\mathbf{z}_i - \mathbf{z}_k\|^2} \right). \tag{6}$$

This almost exactly corresponds to the NCA loss, where the only difference is the construction of batches with a fixed number of classes and a fixed number of images per class.

	*mini*ImageNet		CIFAR-FS	
method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
NCA	62.52 ± 0.24	78.3 ± 0.14	72.48 ± 0.40	85.13 ± 0.29
NCA multi-layer	63.21 ± 0.08	79.27 ± 0.08	72.44 ± 0.36	85.42 ± 0.29
NCA (ours) multi-layer + ss	-	$\mathbf{79.79 \pm 0.08}$	-	$\mathbf{85.66 \pm 0.32}$

Table 1: Comparison between vanilla NCA, NCA using multiple evaluation layers and NCA performing optimisation on the support set (ss). The NCA can only be optimised in the 5-shot case, since there are not enough positives distances in the 1-shot case. Support set is optimised for 5 epochs using Adam with learning rate 0.0001 and weight decay 0.0005. For details, see Sec. A.3

A.5 Differences between the NCA and contrastive losses

Eq. 3 is similar to the contrastive loss functions (Khosla et al., 2020; Chen et al., 2020a). The main differences are that 1.) In contrastive losses, the denominator only contains negative pairs and 2.) the inner sum in the numerator is moved outside of the logarithm in the supervised contrastive loss function from Khosla et al. (2020). We opted to work with the NCA loss because we found it performs better than the supervised constrastive loss in a few-shot learning setting. Using the supervised contrastive loss we only managed to obtain 51.05% 1-shot and 63.36% 5-shot performance on the *mini*Imagenet test set.

A.6 Implementation details

Benchmarks. In our experiments, we use three popular FSL benchmarks. ***mini*ImageNet** (Vinyals et al., 2016) is a subset of ImageNet generated by randomly sampling 100 classes, each with 600 randomly sampled images. We adopt the commonly used splits of Ravi & Larochelle (2017) who use 64 classes for meta-training, 16 for meta-validation and 20 for meta-testing. **CIFAR-FS** was proposed by Bertinetto et al. (2019) as an anagolous version of *mini*Imagenet for CIFAR-100. It uses the same sized splits and same number of images per split as *mini*ImageNet. ***tiered*ImageNet** (Ren et al., 2018) is also constructed from ImageNet, but contains 608 classes, with 351 training classes, 97 validation classes and 160 test classes. The class split have been generated using WordNet (Miller, 1995) to ensure that the training classes are semantically "distant" to the validation and test classes. For all datasets, we use images of size 84×84.

Architecture. In all our experiments, f_θ is represented by a ResNet12 with widths $[64, 160, 320, 640]$. We chose this architecture, initially introduced by Lee et al. (2019), as it is the one which is most frequently adopted by recent FSL methods. Unlike most methods, we do not use a DropBlock regulariser (Ghiasi et al., 2018), as we did not notice it to meaningfully contribute to performance.

Optimisation. To train all the models used for our experiments, unless differently specified, we used a SGD optimiser with Nesterov momentum, weight decay of 0.0005 and initial learning rate of 0.1. For *mini*ImageNet and CIFAR-FS we decrease the learning rate by a factor of 10 after 70% of epochs have been trained, and train for a total of 120 epochs. As data augmentations, we use random horizontal flipping and centre cropping.

Only for the experiments of A.8, we slightly change our training setup. On CIFAR-FS, we increase the number of training epochs from 120 to 240, which improved accuracy of about 0.5%. For *tiered*ImageNet, we train for 120 epochs and decrease the learning rate by a factor of 10 after 50% and 75% of the training progress. These changes affect all our methods and baselines: NCA, Prototypical Networks (with both old and new batch setup), and SimpleShot (Wang et al., 2019).

Projection network. Similarly to (Khosla et al., 2020; Chen et al., 2020a), we also experimented with a *projection network* (but *only* for the comparison of Sec. A.8). The projection network is a single linear layer $A \in \mathbb{R}^{M \times P}$ that is placed on top of f_θ at training time, where M is the output dimension of the neural network f_θ and P is the output dimension of A, which can be considered as a hyper-parameter. The output of A is only used during training. At test time, we do not use the output of A and directly use the output of f_θ. For CIFAR-FS and *tiered*ImageNet, we found this did

	miniImageNet	
method	**5-shot val**	**5-shot test**
Soft Assignment	79.11 ± 0.27	77.16 ± 0.10
k-NN	75.82 ± 0.21	73.52 ± 0.12
1-NN centroid	$\mathbf{80.61 \pm 0.20}$	$\mathbf{78.30 \pm 0.14}$
	CIFAR-FS	
Soft Assignment	76.12 ± 0.32	83.31 ± 0.37
k-NN	73.46 ± 0.39	80.94 ± 0.38
1-NN centroid	$\mathbf{77.80 \pm 0.35}$	$\mathbf{85.13 \pm 0.32}$

Table 2: Comparison of different evaluation methods. Average results over 5 trained models.

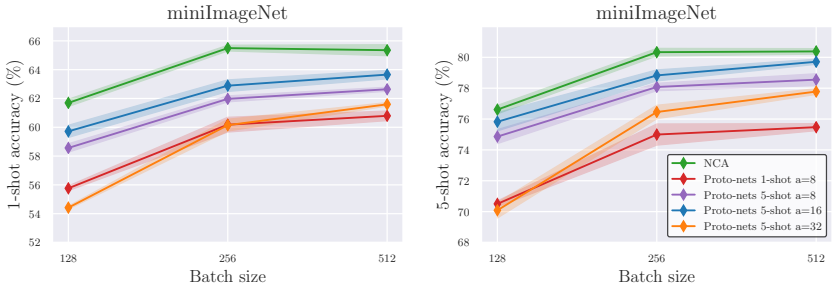

Figure 5: 1-shot (left) and 5-shot accuracies (right) on the validation set of *mini*ImageNet for different batch sizes. Models are trained using NCA or Proto-nets with different configurations: 1-shot with $a = 8$ and 5-shot with $a = 8$, 16 or 32. Reported values correspond to the mean accuracy of five models trained with different random seeds. Please see Sec. 3.2 for details.

not help performance. For *mini*ImageNet however we found that this improved performance, and we set $P = 128$.

Note that this is not an unfair advantage over other methods. Compared to SimpleShot (Wang et al., 2019) and other simple baselines, we actually use fewer parameters without the projection network (effectively making our ResNet12 a ResNet11) since they use an extra fully connected layer to minimise cross entropy during pre-training.

A.7 Additional results for Section 3.2

Fig. 5 complements the results of Fig. 2 from Sec. 3.2

A.8 Comparison with the state-of-the-art

We benchmark our models on three FSL datasets. When considering which methods to compare against, we chose those *a)* which have been recently published, *b)* that use a ResNet12 architecture and *c)* with a setup that is not significantly more complicated than ours. For example, we only report the results of the main approach proposed by Tian et al. (2020) and not their *sequential self-distillation* (Furlanello et al., 2018) variant, which is computationally expensive and can be applied to most methods. For NCA, we found that concatenating the output of intermediate layers modestly improves performance at (almost) no additional cost. We use the output of the average pool layers from all ResNet blocks except from the first and we refer to this variant as NCA multi-layer.

Results can be found in Table 3 for *mini*ImageNet and CIFAR-FS and Table 4 for *tiered*ImageNet. In Table 3 we report Prototypical Networks results for both the episodic setup from Snell et al. (2017) and the best one (batch size 512, 5-shot, a=16) found from the experiment of Fig. 2, which brings a considerable improvement over the original. We did not optimise for a new setup for Prototypical Networks on *tiered*ImageNet, as the larger dataset and the higher number of classes would have made the hyper-parameter search too demanding. Notice how our variants of the NCA

	*mini*ImageNet		CIFAR-FS	
method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Episodic Methods				
adaResNet (Munkhdalai et al., 2018)	56.88 ± 0.62	71.94 ± 0.57	-	-
TADAM(Oreshkin et al., 2018)	58.50 ± 0.30	76.70 ± 0.30	-	-
Shot-Free (Ravichandran et al., 2019)	$60.71\pm$ n/a	$77.64\pm$ n/a	$69.2\pm$ n/a	$84.7\pm$ n/a
TEAM (Qiao et al., 2019)	$60.07\pm$ n/a	$75.90\pm$ n/a	-	-
MTL (Sun et al., 2019)	61.20 ± 1.80	75.50 ± 0.80	-	-
TapNet (Yoon et al., 2019)	61.65 ± 0.15	76.36 ± 0.10	-	-
MetaOptNet-SVM(Lee et al., 2019)	62.64 ± 0.61	78.63 ± 0.46	72.0 ± 0.7	84.2 ± 0.5
Variatonal FSL (Zhang et al., 2019)	61.23 ± 0.26	77.69 ± 0.17	-	-
Simple Baselines				
Transductive finetuning (Dhillon et al., 2020)	62.35 ± 0.66	74.53 ± 0.54	70.76 ± 0.74	81.56 ± 0.53
RFIC-simple (Tian et al., 2020)	62.02 ± 0.63	79.64 ± 0.44	71.5 ± 0.8	86.0 ± 0.5
Meta-Baseline (Chen et al., 2020b)	63.17 ± 0.23	79.26 ± 0.17	-	-
Our implementations:				
Proto-nets (Snell et al. (2017) setup)	59.93 ± 0.23	75.89 ± 0.16	70.20 ± 0.22	83.96 ± 0.16
Proto-nets (our setup)	61.32 ± 0.23	77.77 ± 0.15	70.41 ± 0.31	84.46 ± 0.29
SimpleShot (Wang et al., 2019)	62.16 ± 0.23	78.33 ± 0.17	70.01 ± 0.21	84.50 ± 0.11
NCA (ours)	62.52 ± 0.24	78.3 ± 0.14	72.48 ± 0.40	85.13 ± 0.29
NCA (ours) multi-layer	63.21 ± 0.08	79.27 ± 0.08	72.44 ± 0.36	85.42 ± 0.29

Table 3: Comparison of methods that use ResNet12 on *mini*ImageNet and CIFAR-FS (test set).

	*tiered*ImageNet	
method	**1-shot**	**5-shot**
Shot-Free (Ravichandran et al., 2019)	$63.52\pm$ n/a	$82.59\pm$ n/a
RFIC-simple (Tian et al., 2020)	69.74 ± 0.72	84.41 ± 0.55
Meta-Baseline (Chen et al., 2020b)	68.62 ± 0.27	83.29 ± 0.18
MetaOptNet-SVM(Lee et al., 2019)	65.99 ± 0.72	81.56 ± 0.53
Our implementations:		
Proto-nets (Snell et al. (2017) setup)	65.45 ± 0.23	81.14 ± 0.17
SimpleShot (Wang et al., 2019)	66.17 ± 0.15	80.64 ± 0.20
NCA (ours)	68.36 ± 0.11	83.20 ± 0.18
NCA (ours) multi-layer	68.76 ± 0.07	83.38 ± 0.11

Table 4: Comparison of methods that use ResNet12 on *tiered*ImageNet (test set).

are comparable or superior to the current state-of-the-art, despite being extremely simple. They fair surprisingly well against methods that use meta-learning (and episodic learning), and also against the recently proposed high-performing baselines based on pre-training with the cross-entropy loss.