

Few-Shot Learning with Embedded Class Models and Shot-Free Meta Training

Avinash Ravichandran
Amazon Web Services
ravi.nash@amazon.com

Rahul Bhotika
Amazon Web Services
bhotikar@amazon.com

Stefano Soatto
Amazon Web Services and UCLA
soattos@amazon.com

Abstract

We propose a method for learning embeddings for few-shot learning that is suitable for use with any number of shots (shot-free). Rather than fixing the class prototypes to be the Euclidean average of sample embeddings, we allow them to live in a higher-dimensional space (embedded class models) and learn the prototypes along with the model parameters. The class representation function is defined implicitly, which allows us to deal with a variable number of shots per class with a simple constant-size architecture. The class embedding encompasses metric learning, that facilitates adding new classes without crowding the class representation space. Despite being general and not tuned to the benchmark, our approach achieves state-of-the-art performance on the standard few-shot benchmark datasets.

Figure 1. One image of a mushroom (*Muscaria*) may be enough to recognize it in the wild (left); in other cases, there may be more subtle differences between an edible (*Russula*, shown in the center) and a deadly one (*Phalloides*, shown on the right), but still few samples are enough for humans.

1. Introduction

Consider Figure 1: Given one or few images of an *Amanita Muscaria* (left), one can easily recognize it in the wild. Identifying a *Russula* (center) may require more samples, enough to distinguish it from the deadly *Amanita Phalloides* (right), but likely not millions of them. We refer to this as few-shot learning. This ability comes from having seen and touched millions of other objects, in different environments, under different lighting conditions, partial oc-

clusions and other nuisances. We refer to this as meta-learning. We wish to exploit the availability of large annotated datasets to meta-train models so they can learn new concepts from few samples, or “shots.” We refer to this as meta-training for few-shot learning.

In this paper we develop a framework for both meta-training (learning a potentially large number of classes from a large annotated dataset) and few-shot learning (using the learned model to train new concepts from few samples), designed to have the following characteristics.

Open set: Accommodate an unknown, growing, and possibly unbounded number of new classes in an “open set” or “open universe” setting. Some of the simpler methods available in the literature, for instance based on nearest-neighbors of fixed embeddings [15], do so in theory. In these methods, however, there is no actual few-shot learning per se, as all learnable parameters are set at meta-training.

Continual: Enable leveraging few-shot data to improve the model parameters, even those inferred during meta-training. While each class may only have few samples, as the number of classes grows, the few-shot training set may grow large. We want a model flexible enough to enable “lifelong” or “continual” learning.

Shot Free: Accommodate a variable number of shots for each new category. Some classes may have a few samples, others a few hundred; we do not want to meta-train different models for different number of shots, nor to restrict ourselves to all new classes having the same number of shots, as many recent works do. This may be a side-effect of the benchmarks available that only test a few combinations of shots and “ways” (classes).

Embedded Class Models: Learn a representation of the classes that is not constrained to live in the same space as the representation of the data. All known methods for few-shot learning choose an explicit function to compute class representatives (a.k.a. “prototypes” [15], “proxies,” “means,” “modes,” or “templates”) as some form of averaging in the embedding (feature) space of the data. By decoupling the data (feature space) from the classes (class embedding), we free the latter to live in a richer space, where they can bet-

ter represent complex distributions, and possibly grow over time.

To this end, our contributions are described as follows:

- **Shot-free:** A meta-learning model and sampling scheme that is suitable for use with any number of ways and any number of shots, and can operate in an open-universe, life-long setting. When we fix the shots, as done in the benchmarks, we achieve essentially state-of-the-art performance, but with a model that is far more flexible.
- **Embedded Identities:** We abstract the identities to a different space than the features, thus enabling capturing more complex classes.
- **Implicit Class Representation:** The class representation function has a variable number of arguments, the shots in the class. Rather than fixing the number of shots, or choosing a complex architecture to handle variable numbers, we show that learning an implicit form of the class function enables seamless meta-training, while requiring a relatively simple optimization problem to be solved at few-shot time. We do not use either recurrent architectures that impose artificial ordering, or complex set-functions.
- **Metric Learning** is incorporated in our model, enabling us to add new classes without crowding the class representation space.
- **Performance:** Since there is no benchmark to showcase all the features of our model, we use existing benchmarks for few-shot learning that fix the number of ways and shots to a few samples. Some of the top performing methods are tailored to the benchmark, training different models for different number of shots, which does not scale, and does not enable handling the standard case where each way comes with its own number of shots. Our approach, while not tuned to any benchmark, achieves state-of-the-art performance and is more general.

In the next section we present a formalism for ordinary classification that, while somewhat pedantic, allows us to generalize to life-long, open universe, meta- and few-shot training. The general model allows us to analyze existing work under a common language, and highlights limitations that motivate our proposed solution in Sect. 2.3.

1.1. Background, Notation; Ordinary Classification

In ordinary classification, we call $B = \{(x_i, y_i)\}_{i=1}^M$, with $y_i \in \{1, \dots, B\}$ a “large-scale” training set; $(x_j, y_j) \sim P(x, y)$ a sample from the same distribution. If it is in the training set, we write formally $P(y = k|x_i) =$

$(k - y_i)$. Outside the training set, we approximate this probability with

$$P_w(y = k|x) := \frac{\exp(-w(x)_k)}{\sum_k \exp(-w(x)_k)} \quad (1)$$

where the discriminant $w : X \rightarrow \mathbb{R}^K$ is an element of a sufficiently rich parametric class of functions with parameters, or “weights,” w , and the subscript k indicates the k -th component. The empirical cross-entropy loss is defined as

$$\begin{aligned} L(w) &:= \sum_{(x_i, y_i) \in B} \sum_{k=1}^K -P(y = k|x_i) \log P_w(y = k|x_i) \\ &= \sum_{(x_i, y_i) \in B} -\log P_w(y_i|x_i) \end{aligned} \quad (2)$$

minimizing which is equivalent to maximizing $\prod_i P_w(y_i|x_i)$. If B is i.i.d., this yields the maximum-likelihood estimate \hat{w} , that depends on the dataset B and approximates $\sum_y w(x)_y \log P(y|x)$. We write cross-entropy explicitly as a function of the discriminant as

$$L(w) = \sum_{(x_i, y_i) \in B} (w(x_i)_{y_i}) \quad (3)$$

by substituting (1) into (2), where \log is given, with a slight abuse of notation, by

$$(v_i) := -v_i + \text{LSE}(v) \quad (4)$$

with the log-sum-exp $\text{LSE}(v) := \log \sum_{k=1}^K \exp(v_k)$. Next, we introduce the general form for few-shot and life-long learning, used later to taxonomize modeling choices made by different approaches in the literature.

1.2. General Few-Shot Learning

Let $F = \{(x_j, y_j)\}_{j=1}^{N(k)}$ be the few-shot training set, with $k \leq N$ the classes, or “ways,” and $N(k)$ the “shots,” or samples per class. We assume that meta- and few-shot data x_i, x_j live in the same domain (e.g., natural images), while the meta- and few-shot classes are disjoint, which we indicate with $y \in B + \{1, \dots, K\}$.¹

During meta-training, from the dataset B we learn a parametric representation (feature, or embedding) of the data $w(x)$, for use later for few-shot training. During few-shot training, we use $N(k)$ samples for each new category

¹The number of ways K is a-priori unknown and potentially unbounded. It typically ranges from a few to few hundreds, while $N(k)$ is anywhere from one to a few thousands. The meta-training set has typically M in the millions and B in the thousands. Most benchmarks assume the same number of shots for each way, so there is a single number N , an artificial and unnecessary restriction. There is no loss of generality in assuming the classes are disjoint, as few-shot classes that are shared with the meta-training set can just be incorporated into the latter.

$k > B$ to train a classifier, with k potentially growing unbounded (life-long learning). First, we define “useful” and then formalize a criterion to learn the parameters w , both during meta- and few-shot training.

Unlike standard classification, discussed in the previous section, here we do not know the number of classes ahead of time, so we need a representation that is more general than a K -dimensional vector w . To this end, consider two additional ingredients: A representation of the classes c_k (identities, prototypes, proxies), and a mechanism to associate a datum x_j to a class k through its representative c_k . We therefore have three functions, all in principle learnable and therefore indexed by parameters w . The data representation $w : \mathbf{X} \rightarrow \mathbf{R}^F$ maps each datum to a fixed-dimensional vector, possibly normalized,

$$z = w(x). \quad (5)$$

We also need a class representation, that maps the $N(k)$ features z_j sharing the same identity $y_j = k$, to some representative c_k through a function $w : \mathbf{R}^{F \cdot N(k)} \rightarrow \mathbf{R}^C$ that yields, for each $k = B + 1, \dots, B + K$

$$c_k = w(\{z_j \mid y_j = k\}) \quad (6)$$

where $z_j = w(x_j)$. Note that the argument of w has variable dimension. Finally, the class membership can be decided based on the posterior probability of a datum belonging to a class, approximated with a sufficiently rich parametric function class in the exponential family as we did for standard classification,

$$P_w(y = k \mid x_j) := \frac{\exp(-w(z_j, c_k))}{\sum_k \exp(-w(z_j, c_k))} \quad (7)$$

where $w : \mathbf{R}^F \times \mathbf{R}^C \rightarrow \mathbf{R}$ is analogous to (1). The cross-entropy loss (2) can then be written as

$$L(w) = \sum_{k=B+1}^{B+K} \sum_{j=1}^{N(k)} (-w(z_j, c_k)) \quad (8)$$

with w given by (4) and c_k by (6). The loss is minimized when $w(z_j, c_k) = \log P(y_j = k \mid x_j)$, a function of the few-shot set F . Note, however, that this loss can also be applied to the meta-training set, by changing the outer sum to $k = 1, \dots, B$, or to any combination of the two, by selecting subsets of $\{1, \dots, B + K\}$. Different approaches to few-shot learning differ in the choice of model M and mixture of meta- and few-shot training sets used in one iteration of parameter update, or training “episode.”

2. Stratification of Few-shot Learning Models

Starting from the most general form of few-shot learning described thus far, we restrict the model until there is no few-shot learning left, to capture the modeling choices made in the literature.

2.1. Meta Training

In general, during meta-training for few-shot learning, one solves some form of

$$\begin{aligned} \hat{w} &= \arg \min_w \sum_{(x_i, y_i) \in \mathcal{B}} L(w, c) \\ &\text{s. t. } z_i = w(x_i); c_i = w(\{z_j \mid y_j = i\}). \end{aligned}$$

Implicit class representation function: Instead of the explicit form in (6), one can infer the function w implicitly: Let $r = \min_w L(w, w)$ be the minimum of the optimization problem above. If we consider $c = \{c_1, \dots, c_B\}$ as free parameters in $L(w, c)$, the equation $r = L(\hat{w}, c)$ defines c implicitly as a function of \hat{w} , w . One can then simply find \hat{w} and c simultaneously by solving

$$\hat{w}, \hat{c} = \arg \min_{w, c} \sum_{\substack{k=1 \\ i \mid y_i = k}}^B (-w(x_i, c_k)) \quad (9)$$

which is equivalent to the previous problem, even if there is no explicit functional form for the class representation w . As we will see, this simplifies meta-learning, as there is no need to design a separate architecture with a variable number of inputs w , but requires solving a (simple) optimization during few-shot learning. This is unlike all other known few-shot learning methods, that learn or fix w during meta-learning, and keep it fixed henceforth.

Far from being a limitation, the implicit solution has several advantages, including bypassing the need to explicitly define a function with a variable number of inputs (or a set function) w . It also enables the identity representation to live in a different space than the data representation, again unlike existing work that assumes a simple functional form such as the mean.

2.2. Few-shot Training

Lifelong few-shot learning: Once meta-training is done, one can use the same loss function in (9) for $k > B$ to achieve life-long, few-shot learning. While each new category $k > B$ is likely to have few samples $N(k)$, in the aggregate the number of samples is bound to grow beyond M , which we can exploit to update both the embedding w , the metric w and the class function $c_k = w$.

Metric learning: A simpler model consists of fixing the parameters of the data representation $\hat{w} := w$ and using the same loss function, but summed for $k > B$, to learn from few shots N_k the new class proxies c_k and change the metric w as the class representation space becomes crowded. If we fix the data representation, during the few-shot training

phase, we solve

$$\hat{w}, \hat{c} = \arg \min_{w, c} \sum_{k=B+1}^{B+K} \sum_{j|y_j=k} (w(\hat{x}_j), c_k) \quad (10)$$

where the dependency on the meta-training phase is through $\hat{\cdot}$ and both \hat{w} and \hat{c} depend on the few-shot dataset F .

New class identities: One further simplification step is to also fix the metric ψ , leaving only the class representatives to be estimated

$$\hat{c} = \arg \min_c \sum_{k=B+1}^{B+K} \sum_{j|y_j=k} (\psi(\hat{x}_j), c_k). \quad (11)$$

The above is the implicit form of the parametric function ψ_w , with parameters $w = c$, as seen previously. Thus evaluating $\hat{c}_k = \psi_c(\{\hat{x}_j | y_j = k\})$ requires solving an optimization problem.

No few-shot learning: Finally, one can fix even the function ψ explicitly, forgoing few-shot learning and simply computing

$$\hat{c}_k = \psi(\{\hat{x}_j | y_j = k\}), \quad k > B \quad (12)$$

that depends on B through $\hat{\cdot}$, and on F through Y_k .

We articulate our modeling and sampling choices in the next section, after reviewing the most common approaches in the literature in light of the stratification described.

2.3. Related Prior Work

Most current approaches fall under the case (12), thus involving no few-shot learning, forgoing the possibility of lifelong learning and imposing additional undue limitations by constraining the prototypes to live in the same space of the features. Many are variants of Prototypical Networks [15], where only one of the three components of the model is learned: ψ is fixed to be the mean, so $c_k := \frac{1}{|Y_k|} \sum_{j|y_j=k} z_j$ and $(z, c) = \|z - c\|^2$ is the Euclidean distance. The only learning occurs at meta-training, and the trainable portion of the model ψ_w is a conventional neural network. In addition, the sampling scheme used for training makes the model dependent on the number of shots, again unnecessarily.

Other work can be classified into two main categories: gradient based [11, 3, 9, 14] and metric based [15, 20, 10, 4]. In the first, a meta-learner is trained to adapt the parameters of a network to match the few-shot training set. [11] uses the base set to learn long short-term memory (LSTM) units [6] that update the base classifier with the data from the few-shot training set. MAML [3] learns an initialization for the network parameters that can be adapted by gradient descent in a few steps. LEO [14] is similar to MAML, but uses a task specific initial condition and performs the adaptation in

a lower-dimensional space. Most of these algorithms adapt $\psi_w(x)$ and use an ordinary classifier at few-shot test time. There is a different $\psi_w(x)$ for every few-shot training set, with little re-use or any continual learning.

On the metric learning side, [20] trains a weighted classifier using an attention mechanism [22] that is applied to the output of a feature embedding trained on the base set. This method requires the shots at meta- and few-shot training to match. Prototypical Networks [15] are trained with episodic sampling and a loss function based on the performance of a nearest-mean classifier [19] applied to a few-shot training set. [4] generates classification weights for a novel class based on a feature extractor using the base training set. Finally, [1] incorporates ridge regression in an end-to-end manner into a deep-learning network. These methods learn a single $\psi_w(x)$, which is reused across few-shot training tasks. The class identities are then either obtained through a function defined a-priori such as the sample mean in [15], an attention kernel [20], or ridge regression [1]. The form of ψ_w or ψ do not change at few-shot training. [10] uses task-specific adaptation networks to facilitate the adapting embedding network with output on a task-dependent metric space. In this method, the form of ψ and ψ_w are fixed and the output of ψ_w is modulated based on the few-shot training set.

Next, we describe our model that, to the best of our knowledge, is the first and only to learn each component of the model: The embedding ψ_w , the metric ψ , and implicitly the class representation ψ_w .

3. Proposed Model

Using the formalism of Sect. 2 we describe our modeling choices. Note that there is redundancy in the model class M , as one could fix the data representation $\psi(x) = x$, and devolve all modeling capacity to ψ_w , or vice-versa. The choice depends on the application context. We outline our choices, motivated by limitations of prior work.

Embedding ψ_w : In line with recent work, we choose a deep convolutional network. The details of the architecture are in Sect. 4.

Class representation function ψ_w : We define it implicitly by treating the class representations c_k as parameters along with the weights w . As we saw earlier, this means that at few-shot training, we have to solve a simple optimization problem (11) to find the representatives of new classes, rather than computing the mean as in Prototypical Networks and its variants:

$$c_k = \arg \min_c \sum_{j|y_j=k} (\psi_w(\hat{x}_j), c) = \psi_c(k). \quad (13)$$

Note that the class estimates depend on the parameters w in ψ_w . If few-shot learning is resource constrained, one can

still learn the class representations implicitly during meta-training, and approximate them with a fixed function, such as the mean, during the few-shot phase.

Metric : we choose a discriminant induced by the Euclidean distance in the space of class representations, to which data representations are mapped by a learnable parameter matrix W :

$$w_w(z_j, c_k) = \|W \hat{x}_j - c_k\|^2 \quad (14)$$

Generally, we pick the dimension of c larger than the dimension of z , to enable capturing complex multi-modal identity representations. Note that this choice encompasses metric learning: If $Q = Q^T$ was a symmetric matrix representing a change of inner product, then $\|W \hat{x}_j - c_k\|_Q^2 = \hat{x}_j^T W^T Q W \hat{x}_j - 2 \hat{x}_j^T W^T Q c_k + c_k^T Q c_k$ would be captured by simply choosing the weights $\tilde{W} = QW$. Since both the weights and the class proxies are free, there is no gain in generality in adding the metric parameters Q . Of course, W can be replaced by any non-linear map, effectively “growing” the model via

$$w_w(z_j, c_k) = \hat{f}_w(\hat{x}_j) - c_k \quad (15)$$

for some parametric family f_w such as a deep neural network.

4. Implementation

Embedding $w(x_j)$ We use two different architectures. The first [15, 20] is four-convolution blocks, each block with $64 \times 3 \times 3$ filters followed by batch-normalization and ReLU. This is passed through max-pooling of a 2×2 kernel. Following the convention in [4], we call this architecture C64. The other network is a modified ResNet [5], similar to [10]. We call this ResNet-12.

In addition, we normalize the embedding to live on the unit sphere, i.e. $\hat{x} \in S^{d-1}$, where d is the dimension of the embedding. This normalization is added as a layer to ensure that the feature embeddings are on the unit sphere, as opposed to applying it post-hoc. This adds some complications during meta-training due to poor scaling of gradients [21], and is addressed by a single parameter layer after normalization, whose sole purpose is scaling the output of the normalization layer. This layer is not required at test time.

Class representation: As noted earlier, this is implicit during meta-training. In order to show the flexibility of our framework, we increase the dimension of the class representation.

Metric We choose the angular distance in feature space, which is the d -hypersphere:

$$(z_j, c_k) = \|W z_j - c_k\|^2 = 2s^2(1 - \cos \theta), \quad (16)$$

where s is the scaling factor used during training and the angle between the normalized arguments. As the representation $z = w(x)$ is normalized, the class-conditional model is a Fisher-Von Mises (spherical Gaussian). However, as $W w(x_i) \in S^{d-1}$, we need $W w \in S^{d-1}$. During meta-training we apply the same normalization and scale function to the implicit representation as well.

$$P_w(y = k|x) = \exp(W w(x), c_k) \quad (17)$$

up to the normalization constant.

Sampling At each iteration during meta-training, images from the training set B are presented to the network in the form of episodes [20, 11, 15]; each episode consists of images sampled from K classes. The images are selected by first sampling K classes from B and then sampling N_e images from each of the sampled classes. The loss function is now restricted to the K classes present in the episode as opposed to the entire set of classes available at meta-training. This setting allows for the network to learn a better embedding for an open set classification as shown in [2, 20]

Unlike existing sampling methods that use episodic sampling [11, 15], we do not split the images within an episode into a meta-train set and a meta-test set. For instance, prototypical networks [15] use the elements in the meta-train set to learn the mean of the class representation. [11] learns the initial conditions for optimization. This requires a notion of training “shot,” and results in multiple networks to match the shots one expects at few-shot training.

Regularization First, we notice that the loss function (9) has a degenerate solution where all the centers and the embeddings are the same. In this case, $P_w(y = k|x_j) = P_w(y = k|x_j)$ for all k and x_j , i.e., $P_w(y = k|x_j)$ is a uniform distribution. For this degenerate case, the entropy is maximum, so we use entropy to bias the solution away from the trivial one. We also use Dropout [16] on top of the embedding $w(x)$ during meta-training. Even when using episodic sampling, the embedding tends to over-fit on the base set in the absence of dropout. We do not use this at few-shot train and test time.

Figure 2 summarizes our architecture for the loss function during meta training. This has layers that are only needed for training such as the scale layer, Dropout and the loss. During few-shot training, we only use the learned embedding $w(x)$.

5. Experimental Results

We test our algorithm on three datasets: miniImagenet [20], tieredImagenet [12] and CIFAR Few-Shot [1]. The miniImagenet dataset consists of images of size 84×84 sampled from 100 classes of the ILSVRC [13] dataset, with

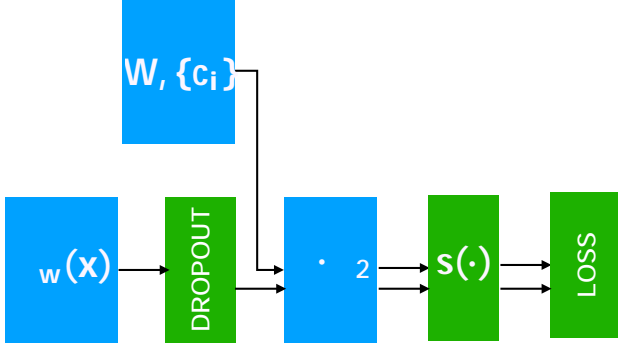


Figure 2. Our meta-training loss flow: The layers represented in blue are the layers that remain after meta-training. While the green layers are used only for training. Here \cdot_2 represents an L_2 normalization layer and $s(\cdot)$ represents a scaling layer

600 images per class. We used the data split outlined in [11], where 64 classes are used for training, 16 classes are used for validation, and 20 classes are used for testing.

We also use tieredImageNet [12]. This is a larger subset of ILSVRC, and consists of 779,165 images of size 84×84 representing 608 classes hierarchically grouped into 34 high-level classes. The split of this dataset ensures that sub-classes of the 34 high-level classes are not spread over the training, validation and testing sets, minimizing the semantic overlap between training and test sets. The result is 448,695 images in 351 classes for training, 124,261 images in 97 classes for validation, and 206,209 images in 160 classes for testing. For a fair comparison, we use the same training, validation and testing splits as in [12], and use the classes at the lowest level of the hierarchy.

Finally, we use CIFAR Few-Shot, (CIFAR-FS) [1] containing images of size 32×32 , a reorganized version of the CIFAR-100 [8] dataset. We use the same data split as in [1], dividing the 100 classes into 64 for training, 16 for validation, and 20 for testing.

5.1. Comparison to Prototypical Networks

Many recent methods are variants of Prototypical Networks, so we perform detailed comparison with it. We keep the training procedure, network architecture, batch-size as well as data augmentation the same. The performance gains are therefore solely due to the improvements in our method.

We use ADAM [7] for training with an initial learning rate of 10^{-3} , and a decay factor of 0.5 every 2,000 iterations. We use the validation set to determine the best model. Our data augmentation consists of mean subtraction, standard-deviation normalization, random cropping and random flipping during training. Each episode contains 15 query samples per class during training. In all our experiments, we set $\alpha = 1$ and did not tune this parameter.

Except otherwise noted, we always test few-shot algo-

rithms on 2000 episodes, with 30 query classes per point per episode. At few-shot training, we experimented with setting the class identity to be implicit (optimized) or average prototype (fixed). The latter may be warranted when the few-shot phase is resource-constrained and yields similar performance. To compare computation time, we use the fixed mean. Note that, in all cases, the class prototypes are learned implicitly during meta-training.

The results of this comparison are shown in Table 1. From this table we see that for the 5-shot 5-way case we perform similarly to Prototypical Network. However, for the 1-shot case we see significant improvements across all three datasets. Also, the performance of Prototypical Networks drops when the train and test shot are changed. Table 1 shows a significant drop in performance when we test models with a 5-shot setting and train with 1-shot. Notice that, from the table, our method is able to maintain the same performance. Consequently, we only train **one** model and test it across the different shot scenarios, hence the moniker “shot-free.”

5.2. Effect of Dimension of Class Identities

Class identities c_k can live in a space of different dimensions than the feature embedding. This can be done in two ways: by lifting the embedding into a higher dimension space or by projecting the class identity into the embedding dimension. If the dimension of the class identity changes, we also need to modify W according to (14). The weight matrix $W \in \mathbb{R}^{d \times \mu}$, where d is the dimension of the embedding and μ is the dimension of the class identities, can be learned during meta-training. This is equivalent to adding a fully connected layer through which the class identities are passed before normalization. Thus, we now learn w_k and w . We show experimental results with the C64 architecture on the miniImageNet datasets in Table 2. Here, we tested the dimension of the class identities to be $2\times$, $5\times$ and $10\times$ the dimension of the embedding. From this table we see that increasing the dimensions gives us a performance boost. However, this increase saturates at a dimension of $2\times$ the dimension of the embedding space.

5.3. Comparison to the State-of-the-art

In order to compare with the state-of-the-art, we use the ResNet-12 base architecture, train our approach using SGD with Nesterov momentum with an initial learning rate of 0.1, weight decay of $5e-4$, momentum of 0.9 and eight episodes per batch. Our learning rate was decreased by a factor of 0.5 every time the validation error did not improve for 1000 iterations. We did not tune these parameters based on the dataset. As mentioned earlier, we train **one** model and test across various shots. We also compare our method with class identities in a space with twice the dimension of the embedding. Lastly, we compare our method

Dataset	Testing Scenario	Training Scenario	Our implementation of [15]	Our Method
miniImagenet	1-shot 5-way	1-shot 5-way	43.88 \pm 0.40	49.07 \pm 0.43
	5-shot 5-way	1-shot 5-way	58.33 \pm 0.35	64.98 \pm 0.35
	5-shot 5-way	5-shot 5-way	65.49 \pm 0.35	65.73 \pm 0.36
tieredImagenet	1-shot 5-way	1-shot 5-way	41.36 \pm 0.40	48.19 \pm 0.43
	5-shot 5-way	1-shot 5-way	55.93 \pm 0.39	64.60 \pm 0.39
	5-shot 5-way	5-shot 5-way	65.51 \pm 0.38	65.50 \pm 0.39
CIFAR Few-Shot	1-shot 5-way	1-shot 5-way	50.74 \pm 0.48	55.14 \pm 0.48
	5-shot 5-way	1-shot 5-way	64.63 \pm 0.42	70.33 \pm 0.40
	5-shot 5-way	5-shot 5-way	71.57 \pm 0.38	71.66 \pm 0.39

Table 1. Comparison of results from our method to that of our implementation of Prototypical Network [15] using the C64 network architecture. The table shows the accuracy and 95% percentile confidence interval of our method averaged over 2,000 episodes on different datasets. Note that our method does not have a notion of shot, here we when we imply training by different shot, we mean that the batch sizes is the same as that of the prescribed method.

Dimension	1x	2x	5x	10x
Performance	49.07	51.46	51.46	51.32

Table 2. Performance of our method on miniImagenet with the class identity dimension as a function of the embedding dimension using the C64 network architecture. The table shows the accuracy averaged over 2,000 episodes.

with a variant of ResNet where we change the filter sizes to (64,160,320,640) from (64,128,256,512).

The results of our comparison for miniImagenet is shown in Table 3. Modulo empirical fluctuations, our method performs at the state-of-the-art and in some cases exceeds it. We wish to point out that SNAIL [9], TADAM [10, 17], LEO [14], MTLF [17] pre-train the network for a 64 way classification task on miniImagenet and 351 way classification on tieredImagenet. However, all the models trained for our method are trained from scratch and use no form of pre-training. We also do not use the meta-validation set for tuning any parameters other than selecting the best trained model using the error on this set. Furthermore, unlike all other methods, we did not have to train multiple networks and tune the training strategy for each case. Lastly, LEO [14] uses a very deep 28 layer Wide-ResNet as a base model compared to our shallower ResNet-12. A fair comparison would involve training our methods with the same base network. However, we include this comparison for complete transparency.

The performance of our method on tieredImagenet is shown in Table 4. This table shows that we are the top performing method for 1-shot 5-way and 5-shot 5-way. We test on this dataset as it is much larger and does not have semantic overlap between meta training and few-shot training even though fewer baselines exist for this dataset compared to miniImagenet. Also shown in Table 4 is the perfor-

Algorithm	1-shot 5-way	5-Shot 5-way	10-shot 5-way
Meta LSTM [11]	43.44	60.60	-
Matching networks [20]	44.20	57.0	-
MAML [3]	48.70	63.1	-
Prototypical Networks [15]	49.40	68.2	-
Relation Net [18]	50.40	65.3	-
R2D2 [1]	51.20	68.2	-
SNAIL [9]	55.70	68.9	-
Gidaris et al. [4]	55.95	73.00	-
TADAM [10]	58.50	76.7	80.8
MTFL [17]	61.2	75.5	-
LEO [14]	61.76	77.59	-
Our Method (ResNet-12)	59.00	77.46	82.33
Our Method (ResNet-12) 2x dims.	60.64	77.02	80.80
Our Method (ResNet-12 Variant)	59.04	77.64	82.48
Our Method (ResNet-12 Variant) 2x dims	60.71	77.26	81.34

Table 3. Performance of 4 variants of our method on miniImagenet compared to the state-of-the-art. The table shows the accuracy averaged over 2,000 episodes.

mance of our method on the CIFAR Few-Shot dataset. We show results on this dataset to illustrate that our method can generalize across datasets. From this table we see that our method performs the best for CIFAR Few-Shot.

5.4 Effect of Choices in Training

As a final remark, there is no consensus on the few-shot training and testing paradigm in the literature. There are too many variables that can affect performance. To illustrate this, we show the effect of few training choices.

Effect of Optimization algorithm In the original implementation of Prototypical Networks [15], ADAM [7] was used as the optimization algorithm. However, most newer algorithms such as [10, 4] use SGD as their optimization algorithm. This result of using different optimization al-

Algorithm	1-shot 5-way	5-Shot 5-way	10-shot 5-way
tieredImageNet			
MAML [3]	51.67	70.30	-
Prototypical Networks [12]	53.31	72.69	-
Relation Net [18]	54.48	71.32	-
LEO [14]	65.71	81.31	-
Our Method (ResNet-12)	63.99	81.97	85.89
Our Method (ResNet-12) 2x dims.	66.87	82.64	85.53
Our Method (ResNet-12) Variant	63.52	82.59	86.62
Our Method (ResNet-12) Variant 2x dims	66.87	82.43	85.74
CIFAR Few-Shot			
MAML [3]	58.9	71.5	-
Prototypical Networks [15]	55.5	72.0	-
Relation Net	55.0	69.3	-
R2D2 [1]	65.3	79.4	-
Our Method (ResNet-12)	69.15	84.70	87.64

Table 4. Performance of our method on tieredImageNet and CIFAR Few-Shot datasets as compared to the state-of-the-art. The performance numbers for CIFAR Few-Shot are from [1]. The table shows the accuracy averaged over 2,000 episodes. Note that the training setting for the prior work is different.

gorithms is shown in Table 5. Here, we show the performance of our algorithm on the miniImagenet dataset using a ResNet-12 model. From this table we see that, while for the 1-shot 5-way the results are better with ADAM as opposed to SGD, we see that the same does not hold for the 5-shot 5-way and 10-shot 5-way scenarios. This shows that SGD generalizes better for our algorithm as compared to ADAM.

Optimization Algorithm	1-shot 5-way	5-Shot 5-way	10-shot 5-way
ADAM	59.41	76.75	81.33
SGD	59.00	77.46	82.33

Table 5. Performance of our method on miniImagenet using the ResNet-12 model with different choices of optimization algorithm. The table shows the accuracy averaged over 2,000 episodes.

Effect of number of tasks per iteration. TADAM [10] and Gidaris et al. [4] use multiple episodes per iteration. They refer to this as tasks in TADAM [10], which uses 2 tasks for 5-shot, 1 task for 10-shot and 5 task for 1-shot. We did not perform any such tuning and instead defaulted it to 8 episodes per iteration based on Gidaris et al. [4]. We also experimented with 16 episodes per iteration. However, this led to a loss in performance across all testing scenarios. Table 6, shows the performance numbers on miniImagenet dataset using the ResNet-12 architecture and trained using ADAM [7] as the optimization algorithm. From this table we see that for all the scenarios 8 episodes per iteration has

a better performance.

Choice	1-shot 5-way	5-Shot 5-way	10-shot 5-way
8 episodes per iteration	59.41	76.75	81.33
16 episodes per iteration	58.22	74.53	78.61

Table 6. Performance of our method on miniImagenet using a ResNet-12 model with different choices of episodes per iteration. The table shows the accuracy averaged over 2,000 episodes.

Even with all major factors such as network architecture, training procedure, batch size remaining the same, factors such as the number of query points used for testing these methods affect the performance and methods in existing literature uses anywhere between 15-30 points for testing, and for some methods it is unclear what this choice was. This calls for stricter protocols for evaluation, and richer benchmark datasets.

6. Discussion

We have presented a method for meta-learning for few-shot learning where all three ingredients of the problem are learned: The representation of the data \mathbf{w} , the representation of the classes \mathbf{c} , and the metric or membership function \mathbf{w} . The method has several advantages compared to prior approaches. First, by allowing the class representation and the data representation spaces to be different, we can allocate more representative power to the class prototypes. Second, by learning the class models implicitly we can handle a variable number of shots without having to resort to complex architectures, or worse, training different architectures, one for each number of shots. Finally, by learning the membership function we implicitly learn the metric, which allows class prototypes to redistribute during few-shot learning.

While some of these benefits are not immediately evident due to limited benchmarks, the improved generality allows our model to extend to a continual learning setting where the number of new classes grows over time, and is flexible in allowing each new class to come with its own number of shots. Our model is simpler than some of the top performing ones in the benchmarks. A single model performs on-par or better in the few-shot setting and offers added generality.

References

- [1] Luca Bertinetto, João F. Henriques, Philip H. S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. CoRR, abs/1805.08136, 2018. **4, 5, 6, 7, 8**
- [2] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look

- at few-shot classification. In International Conference on Learning Representations, 2019. 5
- [3] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In ICML, 2017. 4, 7, 8
- [4] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In CVPR, 2018. 4, 5, 7, 8
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, pages 770–778. IEEE Computer Society, 2016. 5
- [6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. 4
- [7] Diederik P. Kingma and Jimmy Lei Ba. ADAM: A method for stochastic optimization. International Conference on Learning Representations 2015, 2015. 6, 7, 8
- [8] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 6
- [9] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In ICLR, 2018. 4, 7
- [10] Boris N. Oreshkin, Pau Rodríguez, and Alexandre Lacoste. Improved few-shot learning with task conditioning and metric scaling. In NIPS, 2018. 4, 5, 7, 8
- [11] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In ICLR, 2017. 4, 5, 6, 7
- [12] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. CoRR, abs/1803.00676, 2018. 5, 6, 8
- [13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015. 5
- [14] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. CoRR, abs/1807.05960, 2018. 4, 7, 8
- [15] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In NIPS, pages 4080–4090, 2017. 1, 4, 5, 7, 8
- [16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014. 5
- [17] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. CoRR, abs/1812.02391, 2018. 7
- [18] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018. 7, 8
- [19] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences*, 99(10):6567–6572, 2002. 4
- [20] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In NIPS, 2016. 4, 5, 7
- [21] Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In Proceedings of the 25th ACM International Conference on Multimedia, MM ’17, pages 1041–1049, New York, NY, USA, 2017. ACM. 5
- [22] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In ICML, pages 2048–2057, 2015. 4