

Generating Classification Weights with GNN Denoising Autoencoders for Few-Shot Learning

Spyros Gidaris^{1,2} and Nikos Komodakis¹

¹University Paris-Est, LIGM, Ecole des Ponts ParisTech

²valeo.ai

Abstract

Given an initial recognition model already trained on a set of base classes, the goal of this work is to develop a meta-model for few-shot learning. The meta-model, given as input some novel classes with few training examples per class, must properly adapt the existing recognition model into a new model that can correctly classify in a unified way both the novel and the base classes. To accomplish this goal it must learn to output the appropriate classification weight vectors for those two types of classes. To build our meta-model we make use of two main innovations: we propose the use of a Denoising Autoencoder network (DAE) that (during training) takes as input a set of classification weights corrupted with Gaussian noise and learns to reconstruct the target-discriminative classification weights. In this case, the injected noise on the classification weights serves the role of regularizing the weight generating meta-model. Furthermore, in order to capture the co-dependencies between different classes in a given task instance of our meta-model, we propose to implement the DAE model as a Graph Neural Network (GNN). In order to verify the efficacy of our approach, we extensively evaluate it on ImageNet based few-shot benchmarks and we report state-of-the-art results.

1. Introduction

Over the last few years, deep learning has achieved impressive results on various visual understanding tasks, such as image classification [19, 35, 38, 15], object detection [29], or semantic segmentation [5]. However, their success heavily relies on the ability to apply gradient based optimization routines, which are computationally expensive, and having access to a large dataset of training data, which often is very difficult to acquire. For example, in the case of image classification, it is required to have available thousands or

hundreds of training examples per class and the optimization routines consume hundreds of GPU days. Moreover, the set of classes that a deep learning based model can recognize remains fixed after its training. In case new classes need to be recognized, then it is typically required to collect thousands / hundreds of training examples for each of them, and re-train or fine-tune the model on those new classes. Even worse, this latter training stage would lead the model to “forget” the initial classes on which it was trained. In contrast, humans can rapidly learn a novel visual concept from only one or a few examples and reliably recognize it later on. The ability of fast knowledge acquisition is assumed to be related with a meta-learning process in the human brain that exploits past experiences about the world when learning a new visual concept. Even more, humans do not forget past visual concepts when learning a new one. Mimicking that behavior with machines is a challenging research problem, with many practical advantages, and is also the topic of this work.

Research on this subject is usually termed *few-shot object recognition*. More specifically, few-shot object recognition methods tackle the problem of learning to recognize a set of classes given access to only a few training examples for each of them. In order to compensate for the scarcity of training data they employ meta-learning strategies that learn how to efficiently recognize a set of classes with few training data by being trained on a distribution of such few-shot tasks (formed from the dataset available during training) that are similar (but not the same) to the few-shot tasks encountered at test time [39]. Few-shot learning is also related to transfer learning since the learned meta-models solve a new task by exploiting the knowledge previously acquired by solving a different set of similar tasks. There is a broad class of few-shot learning approaches, including, among many, metric-learning-based approaches that learn a distance metric between a test example and the training examples [39, 36, 18, 42, 41], methods that learn to map a test example to a class label by accessing memory modules that

store the training examples of that task [8, 22, 32, 16, 23], approaches that learn how to generate model parameters for the new classes given access to the few available training data of them [9, 25, 10, 26, 12], gradient descent-based approaches [27, 7, 2] that learn how to rapidly adapt a model to a given few-shot recognition task via a small number of gradient descent iterations, and training data hallucination methods [14, 41] that learn how to hallucinate more examples of a class given access to its few training data.

Our approach. In our work we are interested in learning a meta-model that is associated with a recognition model already trained on set of classes (these will be denoted as *base classes* hereafter). Our goal is to train this meta-model so as to learn to adapt the above recognition model to a set of novel classes, for which there are only very few training data available (e.g., one or five examples), while at the same time maintaining the recognition performance on the base classes. Note that, with few exceptions [14, 41, 9, 25, 26], most prior work on few-shot learning neglects to fulfill the second requirement. In order to accomplish this goal we follow the general paradigm of model parameter generation from few-data [9, 25, 10, 26]. More specifically, we assume that the recognition model has two distinctive components, a feature extractor network, which (given as input an image) computes a feature representation, and a feature classifier, which (given as input the feature representation of an image) classifies it to one of the available classes by applying a set of classification weight vectors (one per class) to the input feature. In this context, in order to be able to recognize novel classes one must be able to generate classification weight vectors for them. So, the goal of our work is to learn a meta-model that fulfills exactly this task: i.e., given a set of novel classes with few training examples for each of them, as well as the classification weights of the base classes, it learns to output a new set of classification weight vectors (both for the base and novel classes) that can then be used from the feature classifier in order to classify in a unified way both types of classes.

DAE based model parameters generation. Learning to perform such a meta-learning task, i.e., inferring the classification weights of a set of classes, is a difficult meta-problem that requires plenty of training data in order to be reliably solved. However, having access to such a large pool of data is not always possible; or otherwise stated the training data available for learning such meta-tasks might never be enough. In order to overcome this issue we build our meta-model based on a Denoising Autoencoder network (DAE). During training, this DAE network takes as input a set of classification weights corrupted with additive Gaussian noise, and is trained to reconstruct the target-discriminative classification weights. The injected noise on the inputs of our DAE-based

Figure 1: Some classes (e.g., wild-cat animals, birds, or sea creatures) are semantically or visually similar. Thus it is reasonable to assume that there are correlations between their classification weight vectors that could be exploited in order to reconstruct a more discriminative classification weight vector for each of them.

parameter generation meta-model helps in the regularization of the learning procedure thus allowing us to avoid the danger of overfitting on the training data. Furthermore, thanks to the theoretical interpretation of DAEs provided in [1], our DAE-based meta-model is able to approximate gradients of the log of conditional distribution of the classification weights given the available training data by computing the difference between the input weights and the reconstructed weights [1]. Thus, starting by an initial (but not very accurate) estimate of classification weights, our meta-model is able to perform gradient ascents step(s) that will move the classification weights towards more likely configurations (when conditioned on the given training data). In order to properly apply the DAE framework in the context of few-shot learning, we also adapt it so as to follow the episodic formulation [39] typically used in few-shot learning. This further improves the performance of the parameters generation meta-task by forcing it to reconstruct more discriminative classification weights.

Building the model parameters DAE as a Graph Neural Network. Reconstructing classification weights conditioned only on a few training data, e.g., one training example per class, is an ill-defined task for which it is crucial to exploit as much of the available information on the training data of base and novel classes as possible. In our context, one way to achieve that is by allowing the DAE model to learn the structure of the *entire* set of classification weights that has to reconstruct on each instance (i.e., episode) of that task. We believe that such an approach is more principled and can reconstruct more discriminative classification weight vectors than reconstructing the classification weight of each class *independently*. For example, considering that some of the classes (among the classes whose classification weight vectors must be reconstructed by the DAE model in a given task instance) are semantically or visually similar, such as

different species of birds or sea creatures (see Figure 1), it would make sense to assume that there are correlations to their classification weight vectors that could be exploited in order to reconstruct a more discriminative classification weight vector for each of them. In order to capture such co-dependencies between different classes (in a given task instance of our meta-model), we implement the DAE model as a Graph Neural Network (GNN). This is a family of deep learning networks designed to process an unordered set of entities (in our case a set of classes) associated with a graph G such that they take into account their inter-entity relationships (in our case inter-class relationships) when making predictions about them [11, 34, 20, 6, 37] (for a recent survey on models and applications of deep learning on graphs see also Bronstein *et al.* [4]).

Related to our work, Gidaris and Komodakis [9] also tried to capture such class dependencies in the context of few-shot learning, by predicting the classification weight of each novel class as a mixture of the base classification weights through an attention mechanism. In contrast to them, we consider the dependencies that exist between all the classes, both novel and base (and not of a novel class with the base ones as in [9]) and try to capture them in a more principled way through GNN architectures, which are more expressive compared to the simple attention mechanism proposed in [9]. Graph Neural Networks have been also used in the context of few-shot learning by Garcia and Juan [8]. However, in their work, they give as input to the GNN the labeled training examples and the unlabeled test examples of a few-shot problem and they train it to predict the label of the test examples. Differently from them, in our work we provide as input to the GNN some initial estimates of the classification weights of the classes that we want to learn and we train them to reconstruct more discriminative classification weights. Finally, Graph Neural Networks have been applied on a different but related problem, that of zero-shot learning [40, 17], for regressing classification weights. However, in this line of work they apply the GNNs to knowledge graphs provided by external sources (e.g., word-entity hierarchies) while the input that is given to the GNN for a novel class is its word-embedding. Differently from them, in our formulation we do not consider any side-information (i.e., knowledge graphs or word-embeddings), which makes our approach more agnostic to the domain of problems that can solve and the existence of such knowledge graphs.

To sum up, our contributions are, (1) the application of Denoising Autoencoders in the context of few-shot learning, (2) the use of Graph Neural Network architectures for the classification weight generation task, and (3) performing detailed experimental evaluation of our model on ImageNet-FS [13] and MiniImageNet [39] datasets and achieving state-of-the-art results on ImageNet-FS, MiniImageNet, and *tiered*-MiniImageNet [28] datasets.

Figure 2: Given a few training data of some novel classes, our meta-model adapts an existing recognition model such that it can classify in a unified way both novel and base classes by generating classification weights for both types of classes. We learn to perform this task by employing a Denoising Autoencoders (DAE) for classification weight vectors. Specifically, given some initial estimate of classification weights injected with additive Gaussian noise, the DAE is trained to reconstruct target-discriminative classification weights, where the injected noise serves the role of regularizing the weight generation meta-model. Furthermore, in order to capture co-dependencies between different classes (in a given task instance of our meta-model), we implement the DAE model by use of a Graph Neural Network (GNN) architecture.

In the following sections, we describe our classification weights generation methodology in §2, we provide experimental results in §3, and finally we conclude in §4.

2. Methodology

We define as $C(F(\cdot|\cdot)|w)$ a recognition model, where $F(\cdot|\cdot)$ is the feature extractor part of the network with parameters \cdot , and $C(\cdot|w)$ is the feature classification part with parameters w . The parameters w of the classifier consists of N classification weight vectors, $w = \{w_i\}_{i=1}^N$, where N is the number of classes that the network can recognize, and $w_i \in \mathbb{R}^d$ is the d -dimensional classification weight vector of the i -th class. Given an image x , the feature extractor will output a d -dimensional feature $z = F(x|\cdot)$ and then the classifier will compute the classification scores $[s_1, \dots, s_N] = [z \cdot w_1, \dots, z \cdot w_N] := C(z|w)$ of the N classes. In our work we use cosine similarity-based feature classifiers [9, 25]¹ that have been shown to exhibit better performance when transferred to a few-shot recognition task and to be more appropriate for performing unified recognition of both base and novel classes. Therefore, for the above formulation to be valid, we assume that the features z of the feature extractor and the classification weights $w_i \in w$ of the classifier are already L_2 normalized.

Following the formulation of Gidaris and Komodakis [9], we assume that the recognition network has already been trained to recognize a set of N_{bs} base classes using a training set D_{tr}^{bs} . The classification weight vectors that correspond to those N_{bs} classes are defined as $w^{bs} = \{w_i^{bs}\}_{i=1}^{N_{bs}}$. Our goal is to learn a parameter-generating function $g(\cdot)$ that, given as input the classification weights w^{bs} of the base

¹This practically means that the features z and the classification weights $w_i \in w$ are L_2 normalized

classes, and a few training data $D_{tr}^{nv} = \{x_{k,i}\}_{k=1}^K$ of N_{nv} novel classes, it will be able to output a new set of classification weight vectors

$$w = \{w_i\}_{i=1}^{N=N_{bs}+N_{nv}} = g(D_{tr}^{nv}, w^{bs} |) \quad (1)$$

for both the base and the novel classes, where K is the number of training examples per novel class, $x_{k,i}$ is the k -th training example of i -th novel, $N = N_{bs} + N_{nv}$ is the total number of classes, and $g(\cdot)$ are the learnable parameters of the weight generating function. This new set of classification weights w will be used from the classifier $C(\cdot|w)$ for recognizing from now on in a unified way both the base and the novel classes.

The parameter-generating function consists of a Denoising Autoencoder for classification weight vectors implemented with a Graph Neural Network (see Figure 2 for an overview). In the remainder of this section we will describe in more detail how exactly we implement this parameter-generating function.

2.1. Denoising Autoencoders for model parameters generation

In our work we perform the task of generating classification weights by employing a Denoising Autoencoder (DAE) for classification weight vectors. The injected noise on the classification weights that the DAE framework prescribes serves the role of regularizing the weights generation model $g(\cdot)$ and thus (as we will see in the experimental section) boost its performance. Furthermore, the DAE formulation allows to perform the weights generation task as that of (iteratively) refining some initial (but not very accurate) estimates of the weights in a way that moves them to more probable configurations (when conditioned on the available training data). Note that the learnable parameters in $g(\cdot)$ refer to the learnable parameters of the employed DAE model. In the remainder of this section will briefly provide some preliminaries about DAE models and then explain how they are being used in our case.

Preliminaries about DAE. Denoising autoencoders are neural networks that, given inputs corrupted by noise, are trained to reconstruct “clean” versions of them. By being trained on this task they learn the structure of the data to which they are applied. It has been shown [1] that a DAE model, when trained on inputs corrupted with additive Gaussian noise, can estimate the gradient of the energy function of the density $p(w)$ of its inputs w :

$$\frac{\log p(w)}{w} \approx \frac{1}{2} \cdot (r(w) - w), \quad (2)$$

where σ^2 is the amount of Gaussian noise injected during training, and $r(\cdot)$ is the autoencoder. The approximation becomes exact as $\sigma^2 \rightarrow 0$, and the autoencoder is given enough

capacity and training examples. The direction $(r(w) - w)$ points towards more likely configurations of w . Therefore, the DAE learns a vector field pointing towards the manifold where the input data lies. Those theoretical results are independent of the parametrization of the autoencoder.

Applying DAE for classification weight generation. In our case we are interested in learning a DAE model that, given an initial estimate of the classification weight vectors w , would provide a vector field that points towards more probable configurations of w conditioned on the training data $D_{tr} = \{D_{tr}^{nv}, D_{tr}^{bs}\}$. Therefore, we are interested in a DAE model that learns to estimate:

$$\frac{\log p(w|D_{tr})}{w} \approx \frac{1}{2} \cdot (r(w) - w), \quad (3)$$

where $p(w|D_{tr})$ is the conditional distribution of w given D_{tr} , and $r(w)$ is a DAE for the classification weights. So, after having trained our DAE model for the classification weights, $r(w)$, we can perform gradient ascent in $\log p(w|D_{tr})$ in order to (iteratively) reach a mode of the estimated conditional distribution $p(w|D_{tr})$:

$$w \leftarrow w + \eta \cdot \frac{\log p(w|D_{tr})}{w} = w + \eta \cdot (r(w) - w), \quad (4)$$

where η is the gradient ascent step size.

The above iterative inference mechanism of the classification weights requires to have available an initial estimate of them. This initial estimate is produced using the training data D_{tr}^{nv} of the novel classes and the existing classification weights $w^{bs} = \{w_i^{bs}\}_{i=1}^{N_{bs}}$ of the base classes. Specifically, for the base classes we build this initial estimate by using the classification weights already available in w^{bs} and for the novel classes by averaging for each of them the feature vectors of their few training examples:

$$w_i = \begin{cases} w_i^{bs}, & \text{if } i \text{ is a base class} \\ \frac{1}{K} \sum_{k=1}^K F(x_{k,i} |), & \text{otherwise} \end{cases}, \quad (5)$$

where $x_{k,i}$ is the k -th training example of the novel class i .

To summarize, our weight generation function $g(D_{tr}^{nv}, w^{bs} |)$ is implemented by first producing an initial estimate of the new classification weights by applying equation (5) (for which it uses D_{tr}^{nv} , and w^{bs}) and then refining those initial estimates by applying the weight update rule of equation (4) using the classification weights DAE model $r(w)$ (see an overview of this procedure in Figure 3).

Episodic training of classification weights DAE model.

For training, the DAE framework prescribes to apply Gaussian noise on some target weights and then train the DAE model $r(\cdot)$ to reconstruct them. However, in our case, it is more effective to follow a training procedure that more

Figure 3: Overview of how our meta-model (bottom) is applied at test time in order to properly adapt an existing recognition model (top-left) into a new model (top-right) that can classify in a unified way both novel and base classes (where only a few training data are provided at test time for the novel classes).

closely mimics how the DAE model would be used during test time. Therefore, we propose to train the DAE model using a learning procedure that is based on training episodes [39]. More specifically, during training we form training episodes by sampling \tilde{N}_{nv} “fake” novel classes from the available N_{bs} classes in the training data D_{tr}^{bs} and use the remaining $\tilde{N}_{bs} = N_{bs} - \tilde{N}_{nv}$ classes as base ones. We call the sampled novel classes “fake” because they actually belong to the set of base classes but during this training episode are treated as being novel. So, for each “fake” novel class we sample K training examples forming a training set for the “fake” novel classes of this training episode. We also sample M training examples both from the “fake” novel and the remaining base classes forming the validation set $D_{val} = \{(x_m, y_m)\}_{m=1}^M$ of this training episode, where (x_m, y_m) is the image x_m and the label y_m of the m -th validation example. Then, we produce an initial estimate \tilde{w} of the classification weights of the sampled classes (both “fake” novel and base) using the mechanics of equation (5), and we inject to it Gaussian additive noise $\tilde{w} = \{\tilde{w}_i + \}_{i=1}^{\tilde{N}=\tilde{N}_{bs}+\tilde{N}_{nv}}$, where $N(0, \cdot)$. We give \tilde{w} as input to the DAE model in order to output the reconstructed weights $\hat{w} = \{\hat{w}_i\}_{i=1}^{\tilde{N}}$. Having computed \hat{w} we apply to them a squared reconstruction loss of the target weights $w = \{w_i\}_{i=1}^{\tilde{N}}$ and a classification loss of the M validation examples of this training episode:

$$\frac{1}{\tilde{N}} \sum_{i=1}^{\tilde{N}} \|\hat{w}_i - w_i\|^2 + \frac{1}{M} \sum_{m=1}^M \text{loss}(x_m, y_m | \hat{w}), \quad (6)$$

where $\text{loss}(x_m, y_m | \hat{w}) = -z_m \hat{w}_{y_m} + \log(\sum_{i=1}^{\tilde{N}} e^{z_m \hat{w}_i})$ is the cross entropy loss of the m -th validation example and $z_m = F(x_m)$ is the feature vector of the m -th example. Note that the target weights w are the corresponding base class weight vectors already learned by the recognition model.

2.2. Graph Neural Network based Denoising Autoencoder

Here we describe how we implement our DAE model. Reconstructing the classification weights of the novel classes, for which training data are scarce, is an ill-defined problem. One way to boost DAE’s reconstruction performance is by allowing it to take into account the inter-class relationships when reconstructing the classification weights of a set of classes. Given the unordered nature in a set of classes, we chose to implement the DAE model by use of a Graph Neural Network (GNN). In the remainder of this subsection we describe how we employed a GNN for the reconstruction task and what type of GNN architecture we used.

GNNs are multi-layer networks that operate on graphs $G = (V, E)$ by structuring their computations according to the graph connectivity. i.e., at each GNN layer the feature responses of a node are computed based on the neighboring nodes defined by the adjacency graph (see Figure 4a for an illustration). In our case, we associate the set of classes $Y = \{i\}_{i=1}^N$ (for which we want to reconstruct their classification weights) with a graph $G = (V, E)$, where each node $v_i \in V$ corresponds to the class i in Y (either base or novel). To define the set of edges $(i, j) \in E$ of the graph, we connect each class i with its J ² closest classes in terms of the cosine-similarity of the initial estimates of their classification weight vectors (before the injection of the Gaussian noise). The edge strength $a_{ij} \in [0, 1]$ of each edge $(i, j) \in E$ is computed by applying the softmax operation over the cosine similarities scores of its neighbors $N(i) = \{j, (i, j) \in E\}$, thus forcing $\sum_{j \in N(i)} a_{ij} = 1$ ³. We define as $h^{(l)} = \{h_i^{(l)}\}_{i=1}^N$ the set of feature vectors that represent the N graph nodes (i.e., the N classes) at the l -th level of the GNN. In this case, the input set $h^{(0)}$ to the GNN is the set of classification weight

²In our experiments we use $J = 10$ classes as neighbors.

³For this softmax operation we used an inverse temperature value of 5.

(a) The general architecture of a GNN layer.

(b) The architecture of the hidden GNN layers in our work.

Figure 4: (a) A GNN layer typically consists of two functions, an *aggregation* function that, for a node of interest (e.g., node A in the figure), aggregates information from its neighboring nodes, and an *update* function that updates the state (i.e., the features) of that node by taking into account both the state of that node and the aggregated information from its neighborhood. (b) The GNN layer architecture that we use in our work implements the aggregation function as a small Relation-Net network [33]. The two linear layers in the pairwise function $q(\cdot, \cdot)$ are the same (i.e., share parameters).

vectors $w = \{w_i\}_{i=1}^N = h^{(0)}$ that the GNN model will refine. Each GNN layer, receives as input the set $h^{(l)}$ and outputs a new set $h^{(l+1)}$ as:

$$h_{N(i)}^{(l)} = \text{AGGREGATE} \{h_j^{(l)}, j \in N(i)\}, \quad (7)$$

$$h_i^{(l+1)} = \text{UPDATE} \{h_i^{(l)}, h_{N(i)}^{(l)}\}, \quad (8)$$

where $\text{AGGREGATE}(\cdot)$ is a parametric function that for each node i aggregates information from its node neighbors $N(i)$ to generate the message feature $h_{N(i)}^{(l)}$, and $\text{UPDATE}(\cdot, \cdot)$ is a parametric function that for each node i will get as input the features $h_i^{(l)}, h_{N(i)}^{(l)}$ and will compute the new feature vector $h_i^{(l+1)}$ of that node.

Relation-Net based aggregation function. Generally, the aggregation function is implemented as linear combination of message vectors received from the node neighbors:

$$h_{N(i)}^{(l)} = \sum_{j \in N(i)} a_{ij} \cdot q^{(l)}(h_i^{(l)}, h_j^{(l)}), \quad (9)$$

where $q^{(l)}(h_i^{(l)}, h_j^{(l)})$ is a function that computes the message vector that node i receives from its neighbor j . Inspired by relation networks [33], we implement $q^{(l)}(h_i^{(l)}, h_j^{(l)})$ as

a non-linear parametric function of the feature vectors of both the sender and the receiver nodes. Specifically, given the two input vectors, $q^{(l)}$ forwards each of them through the same fully connected linear layer, adds their outputs, and then applies BatchNorm + Dropout + LeakyReLU units (see Figure 4b). Note that in this implementation the message between two nodes, is independent of the direction of the message, i.e., $q^{(l)}(h_i^{(l)}, h_j^{(l)}) = q^{(l)}(h_j^{(l)}, h_i^{(l)})$.

Update function. The update function of the *hidden GNN layers* is implemented as:

$$h_i^{(l+1)} = h_i^{(l)} \parallel u^{(l)}(h_i^{(l)} \parallel h_{N(i)}^{(l)}), \quad (10)$$

where \parallel is the concatenation of vectors and $u^{(l)}(\cdot)$ is a non-linear parametric function that, given as input a vector, forwards it through a fully connected linear layer followed by BatchNorm + Dropout + LeakyReLU + L_2 -normalization units (see Figure 4b). For the *last prediction GNN layer*, the update function is implemented as:

$$w_i, o_i = u^{(L-1)}(h_i^{(L-1)} \parallel h_{N(i)}^{(L-1)}), \quad (11)$$

where $u^{(L-1)}(\cdot)$ is a non-linear parametric function that, given an input vector, outputs the two d -dimensional vectors w_i and o_i . $u^{(L-1)}(\cdot)$ is implemented as a fully connected linear layer followed by a L_2 -normalization unit for the w_i output, and a Sigmoid unit for the o_i output. The final output of the GNN is computed with the following operation:

$$\hat{w}_i = w_i + o_i \odot w_i. \quad (12)$$

As can be seen, we opted for residual-like predictions of the new classification weights \hat{w}_i , since this type of operations are more appropriate for the type of refining/denoising that must be performed by our DAE model. Our particular implementation uses the gating vectors o_i to control the amount of contribution of the residuals w_i to the input weights w_i .

We named this GNN based DAE model for weights reconstruction *wDAE-GNN* model. Alternatively, we also explored a simpler DAE model that is implemented to reconstruct each classification weight vector (in a given task instance of our meta-model) independently with a MLP network (*wDAE-MLP* model). More specifically, the *wDAE-MLP* model is implemented with layers similar to those of the GNN that *only include the update function part and not the aggregation function part*. So, it only includes fully connected layers and the skip connections (i.e., 2nd and 3rd boxes in update function part of Figure 4b).

3. Experimental Evaluation

In this section we first compare our method against prior work in §3.2 and then in §3.3 we perform a detailed experimental analysis of it.

Datasets and Evaluation Metrics. We evaluate our approach on three datasets, ImageNet-FS [13, 41], MiniImageNet [39], and *tiered*-MiniImageNet [28]. **ImageNet-FS** is a few-shot benchmark created by splitting the ImageNet classes [30] into 389 base classes and 611 novel classes; 193 of the base classes and 300 of the novel classes are used for validation and the remaining 196 base classes and 311 novel classes are used for testing. In this benchmark the models are evaluated based on (1) the recognition performance of the 311 test novel classes (i.e., 311-way classification task), and (2) recognition of all the 507 classes (i.e., both the 196 test base classes and the 311 novel classes; for more details we refer to [13, 41]). We report result for $K = 1, 2, 5, 10$, or 20 training examples per novel class. For each of those K -shot settings we sample 100 test episodes (where each episode consists of sampling K training examples per novel class and then evaluating on the validation set of ImageNet) and compute the mean accuracies over all episodes. **MiniImageNet** consists of 100 classes randomly picked from the ImageNet with 600 images per class. Those 100 classes are divided into 64 base classes, 16 validation novel classes, and 20 test novel classes. The images in MiniImageNet have size 84×84 pixels. *tiered*-**MiniImageNet** consists of ImageNet 608 classes divided into 351 base classes, 97 validation novel classes, and 160 novel test classes. In total there are 779, 165 images again with size 84×84 . In MiniImageNet and *tiered*-MiniImageNet, the models are evaluated on several 5-way classification tasks (i.e., test episodes) created by first randomly sampling 5 novel classes from the available test novel classes, and then $K = 1$, or 5 training examples and $M = 15$ test examples per novel class. To report results we use 20000 such test episodes and compute the mean accuracies over all episodes. Note that when learning the novel classes we also feed the base classes to our *wDAE-GNN* models in order to take into account the class dependencies between the novel and base classes.

3.1. Implementation details

Feature extractor architectures. For the ImageNet-FS experiments we use a ResNet-10 [15] architecture that given images of size 224×224 outputs 512-dimensional feature vectors. For the MiniImageNet and *tiered*-MiniImageNet experiments we used a 2-layer Wide Residual Network [43] (WRN-28-10) that receives images of size 80×80 (resized from 84×84) and outputs 640-dimensional feature vectors.

wDAE-GNN and wDAE-MLP architectures. In all our experiments we use a wDAE-GNN architecture with two GNN layers. In ImageNet-FS (MiniImageNet) the $q^{(l)}(\cdot, \cdot)$ parametric function of all GNN layers and the $u^{(l)}(\cdot)$ parametric function of the hidden GNN layer output features of 1024 (320) channels. All dropout units have 0.7 (0.95) dropout ratio, and the σ of the Gaussian noise in DAE is

0.08 (0.1). A similar architecture is used in wDAE-MLP but without the aggregation function parts.

For training we use SGD optimizer with momentum 0.9 and weight decay $5e - 4$. We train our models only on the 1-shot setting and then use them for all the K -shot settings. During test time we apply only 1 refinement step of the initial estimates of the classification weights (i.e., only 1 application of the update rule (4)). In ImageNet-FS the step size of the update rule (4) is set to 1.0, 1.0, 0.6, 0.4, and 0.2 for the $K = 1, 2, 5, 10$, and 20 shot settings respectively. In MiniImageNet is set to 1.0 and 0.5 for the $K = 1$ and $K = 5$ settings respectively. All hyper-parameters were cross-validated in the validation splits of the datasets.

We provide the implementation code at https://github.com/gidari/ss/wDAE_GNN_FewShot

3.2. Comparison with prior work

Here we compare our *wDAE-GNN* and *wDAE-MLP* models against prior work on the ImageNet-FS, MiniImageNet, and *tiered*-MiniImageNet datasets. More specifically, on ImageNet-FS (see Table 1) the proposed models achieve in most cases superior performance than prior methods - especially on the challenging and interesting scenarios of having less than 5 training examples per novel class (i.e., $K = 5$). For example, the *wDAE-GNN* model improves the 1-shot accuracy for novel classes of the previous state-of-the-art [9] by around 1.8 accuracy points. On MiniImageNet and *tiered*-MiniImageNet (see Tables 2 and 4 respectively) the proposed models surpass the previous methods on all settings and achieve new state-of-the-art results. Also, for MiniImageNet we provide in Table 3 the classification accuracies of both the novel and base classes and we compare with the LwoF [9] prior work. Again, we observe that our models surpass the prior work.

3.3. Analysis of our method

Ablation study of DAE framework. Here we perform ablation study of various aspects of our DAE framework on the ImageNet and the MiniImageNet datasets (see corresponding results on Tables 1 and 2). Specifically, we examine the cases of (1) training the reconstruction models without noise (entries with suffix *No Noise*), (2) during training providing as input to the model noisy versions of the target classification weights that has to reconstruct (entries with suffix *Noisy Targets as Input*), (3) training the models without classification loss on the validation examples (i.e., using only the first term of the loss (6); entries with suffix *No Cls. Loss*), and (4) training the models with only the classification loss on the validation examples and without the reconstruction loss (i.e., using only the second term of the loss (6); entries with suffix *No Rec. Loss*). (5) We also provide the recognition performance of the initial estimates of the classification weight vectors without being refined by our DAE models

Approach	Novel classes					All classes				
	K=1	2	5	10	20	K=1	2	5	10	20
Prior work										
Prototypical-Nets (from [41])	39.3	54.4	66.3	71.2	73.9	49.5	61.0	69.7	72.9	74.6
Matching Networks (from [41])	43.6	54.0	66.0	72.5	76.9	54.4	61.0	69.0	73.7	76.5
Logistic regression [13]	38.4	51.1	64.8	71.6	76.6	40.8	49.9	64.2	71.9	76.9
Logistic regression w/ H [13]	40.7	50.8	62.0	69.3	76.5	52.2	59.4	67.6	72.8	76.9
SGM w/ H [13]	-	-	-	-	-	54.3	62.1	71.3	75.8	78.1
Batch SGM [13]	-	-	-	-	-	49.3	60.5	71.4	75.8	78.5
Prototype Matching Nets w/ H [41]	45.8	57.8	69.0	74.3	77.4	57.6	64.7	71.9	75.2	77.5
LwoF [9]	46.2	57.5	69.2	74.8	78.1	58.2	65.2	72.7	76.5	78.7
Ours										
wDAE-GNN	48.0	59.7	70.3	75.0	77.8	59.1	66.3	73.2	76.1	77.5
wDAE-MLP	47.6	59.2	70.0	74.8	77.7	59.0	66.1	72.9	75.8	77.4
Ablation study on wDAE-GNN										
Initial estimates	45.4	56.9	68.9	74.5	77.7	57.0	64.3	72.3	75.6	77.3
wDAE-GNN - No Noise	47.6	59.0	70.0	74.9	77.8	60.0	66.0	72.9	75.8	77.4
wDAE-GNN - Noisy Targets as Input	47.8	59.4	70.1	74.8	77.7	58.7	66.0	73.1	76.0	77.5
wDAE-GNN - No Cls. Loss	47.7	59.1	69.8	74.6	77.6	58.4	65.5	72.7	75.8	77.5
wDAE-GNN - No Rec. Loss	47.8	59.4	70.1	75.0	77.8	58.7	66.0	73.1	76.1	77.6

Table 1: Top-5 accuracies on the novel and on all classes for the ImageNet-FS benchmark [13]. To report results we use 100 test episodes. For all our models the 95% confidence intervals on the $K = 1, 2, 5, 10$, and 20 settings are (around) ± 0.21 , ± 0.15 , ± 0.08 , ± 0.06 , and ± 0.05 respectively for Novel classes and ± 0.13 , ± 0.10 , ± 0.05 , ± 0.04 , and ± 0.03 for All classes.

Models	Backbone	1-shot	5-shot
Prior work			
MAML [7]	Conv-4-64	48.70 \pm 1.84%	63.10 \pm 0.92%
Prototypical Nets [36]	Conv-4-64	49.42 \pm 0.78%	68.20 \pm 0.66%
LwoF [9]	Conv-4-64	56.20 \pm 0.86%	72.81 \pm 0.62%
RelationNet [42]	Conv-4-64	50.40 \pm 0.80%	65.30 \pm 0.70%
GNN [8]	Conv-4-64	50.30%	66.40%
R2-D2 [3]	Conv-4-64	48.70 \pm 0.60%	65.50 \pm 0.60%
R2-D2 [3]	Conv-4-512	51.20 \pm 0.60%	68.20 \pm 0.60%
TADAM [24]	ResNet-12	58.50 \pm 0.30%	76.70 \pm 0.30%
Munkhdalai <i>et al.</i> [23]	ResNet-12	57.10 \pm 0.70%	70.04 \pm 0.63%
SNAIL [33]	ResNet-12	55.71 \pm 0.99%	68.88 \pm 0.92%
Qiao <i>et al.</i> [26] ^y	WRN-28-10	59.60 \pm 0.41%	73.74 \pm 0.19%
LEO [31] ^y	WRN-28-10	61.76 \pm 0.08%	77.59 \pm 0.12%
LwoF [9] (our implementation)	WRN-28-10	60.06 \pm 0.14%	76.39 \pm 0.11%
Ours			
wDAE-GNN	WRN-28-10	61.07 \pm 0.15%	76.75 \pm 0.11%
wDAE-MLP	WRN-28-10	60.61 \pm 0.15%	76.56 \pm 0.11%
wDAE-GNN ^y	WRN-28-10	62.96 \pm 0.15%	78.85 \pm 0.10%
wDAE-MLP ^y	WRN-28-10	62.67 \pm 0.15%	78.70 \pm 0.10%
Ablation study on wDAE-GNN			
Initial estimate	WRN-28-10	59.68 \pm 0.14%	76.48 \pm 0.11%
wDAE-GNN - No Noise	WRN-28-10	60.29 \pm 0.14%	76.49 \pm 0.11%
wDAE-GNN - Noisy Targets as Input	WRN-28-10	60.92 \pm 0.15%	76.69 \pm 0.11%
wDAE-GNN - No Cls. Loss	WRN-28-10	60.96 \pm 0.15%	76.75 \pm 0.11%
wDAE-GNN - No Rec. Loss	WRN-28-10	60.76 \pm 0.15%	76.64 \pm 0.11%
Ablation study on wDAE-MLP			
wDAE-MLP - No Noise	WRN-28-10	60.16 \pm 0.15%	76.50 \pm 0.11%
wDAE-MLP - Noisy Targets as Input	WRN-28-10	60.43 \pm 0.15%	76.49 \pm 0.11%
wDAE-MLP - No Cls. Loss	WRN-28-10	60.55 \pm 0.15%	76.62 \pm 0.11%
wDAE-MLP - No Rec. Loss	WRN-28-10	60.45 \pm 0.15%	76.50 \pm 0.11%

Table 2: Top-1 accuracies on the novel classes of MiniImageNet test set with 95% confidence intervals. ^y: using also the validation classes for training.

(entries *Initial Estimates*). We observe that each of those ablations to our DAE models lead to worse few-shot recognition performance. Among them, the models trained without noise on their inputs achieves the worst performance, which demonstrates the necessity of the DAE formulation.

Models	Novel classes		All classes	
	1-shot	5-shot	1-shot	5-shot
LwoF [9]	60.03 \pm 0.14%	76.35 \pm 0.11%	55.70 \pm 0.08%	66.27 \pm 0.07%
wDAE-GNN	61.07 \pm 0.15%	76.75 \pm 0.11%	56.55 \pm 0.08%	67.00 \pm 0.07%
wDAE-MLP	60.61 \pm 0.14%	76.56 \pm 0.11%	56.07 \pm 0.08%	67.05 \pm 0.07%

Table 3: Top-1 accuracies on the novel and on all classes of Mini-ImageNet test set with 95% confidence intervals.

Models	Backbone	1-shot	5-shot
MAML [7] (from [21])	Conv-4-64	51.67 \pm 1.81%	70.30 \pm 0.08%
Prototypical Nets [36]	Conv-4-64	53.31 \pm 0.89%	72.69 \pm 0.74 %
RelationNet [42] (from [21])	Conv-4-64	54.48 \pm 0.93%	71.32 \pm 0.78%
Liu <i>et al.</i> [21]	Conv-4-64	57.41 \pm 0.94%	71.55 \pm 0.74
LEO [31]	WRN-28-10	66.33 \pm 0.05%	81.44 \pm 0.09 %
LwoF [9] (our implementation)	WRN-28-10	67.92 \pm 0.16%	83.10 \pm 0.12%
wDAE-GNN (Ours)	WRN-28-10	68.18 \pm 0.16%	83.09 \pm 0.12%

Table 4: Top-1 accuracies on the novel classes of *tiered*-MiniImageNet test set with 95% confidence intervals.

Impact of GNN architecture. By comparing the classification performance of the *wDAE-GNN* models with the *wDAE-MLP* models in Tables 1 and 2, we observe that indeed, taking into account the inter-class dependencies with the proposed GNN architecture is beneficial to the few-shot recognition performance. Specifically, the GNN architecture offers a small (e.g., around 0.40 percentage points in the 1-shot case) but consistent improvement that according to the confidences intervals of Tables 1 and 2 is in almost all cases statistically significant.

4. Conclusion

We proposed a meta-model for few-shot learning that takes as input a set of novel classes (with few training examples for each of them) and then generates classification weight vectors for them. Our model is based on the use of a Denoising Autoencoder (DAE) network. During training, the injected noise on the classification weights given as input to the DAE network allows the regularization of the learning procedure and helps in boosting the performance of the meta-model. After training, the DAE model is used for refining an initial set of classification weights with the goal of making them more discriminative with respect to the classification task at hand. We implemented the above DAE model by use of a Graph Neural Network architecture so as to allow our meta-model to properly learn (and take advantage of) the structure of the *entire* set of classification weights that must be reconstructed on each instance (i.e., episode) of the meta-learning task. Our detailed experiments on the ImageNet-FS [13] and MiniImageNet [39] datasets reveal (1) the significance of our DAE formulation for training meta-models capable to generate classification weights, and (2) that the GNN architecture manages to offer a consistent improvement on the few-shot classification accuracy. Finally, our model surpassed prior methods on all the explored datasets.

References

- [1] G. Alain and Y. Bengio. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research*, 15(1):3563–3593, 2014. **2, 4**
- [2] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, pages 3981–3989, 2016. **2**
- [3] L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019. **8**
- [4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. **3**
- [5] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018. **1**
- [6] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015. **3**
- [7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017. **2, 8**
- [8] V. Garcia and J. Bruna. Few-shot learning with graph neural networks. *arXiv preprint arXiv:1711.04043*, 2017. **2, 3, 8**
- [9] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018. **2, 3, 7, 8**
- [10] F. Gomez and J. Schmidhuber. Evolving modular fast-weight networks for control. In *International Conference on Artificial Neural Networks*, pages 383–389. Springer, 2005. **2**
- [11] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Neural Networks, 2005. IJCNN’05. Proceedings. 2005 IEEE International Joint Conference on*, volume 2, pages 729–734. IEEE, 2005. **3**
- [12] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016. **2**
- [13] B. Hariharan and R. Girshick. Low-shot visual recognition by shrinking and hallucinating features. *arXiv preprint arXiv:1606.02819*, 2016. **3, 7, 8**
- [14] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *ICCV*, pages 3037–3046, 2017. **2**
- [15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. **1, 7**
- [16] L. Kaiser, O. Nachum, A. Roy, and S. Bengio. Learning to remember rare events. *arXiv preprint arXiv:1703.03129*, 2017. **2**
- [17] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing. Rethinking knowledge graph propagation for zero-shot learning. *arXiv preprint arXiv:1805.11724*, 2018. **3**
- [18] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015. **1**
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. **1**
- [20] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015. **3**
- [21] Y. Liu, J. Lee, M. Park, S. Kim, and Y. Yang. Transductive propagation network for few-shot learning. *arXiv preprint arXiv:1805.10002*, 2018. **8**
- [22] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. 2018. **2**
- [23] T. Munkhdalai and H. Yu. Meta networks. *arXiv preprint arXiv:1703.00837*, 2017. **2, 8**
- [24] B. N. Oreshkin, A. Lacoste, and P. Rodriguez. Tadam: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*, 2018. **8**
- [25] H. Qi, M. Brown, and D. G. Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5822–5830, 2018. **2, 3**
- [26] S. Qiao, C. Liu, W. Shen, and A. Yuille. Few-shot image recognition by predicting parameters from activations. *arXiv preprint arXiv:1706.03466*, 2, 2017. **2, 8**
- [27] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations*, 2017. **2**
- [28] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018. **3, 7**
- [29] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. **1**
- [30] 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. **7**
- [31] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018. **8**
- [32] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016. **2**
- [33] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017. **6, 8**

- [34] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. 3
- [35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1
- [36] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4077–4087, 2017. 1, 8
- [37] S. Sukhbaatar, R. Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016. 3
- [38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1
- [39] O. Vinyals, C. Blundell, T. Lillicrap, and D. Wierstra. Matching networks for one shot learning. In *NIPS*, pages 3630–3638, 2016. 1, 2, 3, 5, 7, 8
- [40] X. Wang, Y. Ye, and A. Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6857–6866, 2018. 3
- [41] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. *arXiv preprint arXiv:1801.05401*, 2018. 1, 2, 7, 8
- [42] F. S. Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. 2018. 1, 8
- [43] S. Zagoruyko and N. Komodakis. Wide residual networks. In *Proc. British Machine Vision Conference*, 2016. 7