Dense Classification and Implanting for Few-Shot Learning

Yann Lifchitz[1,2] Yannis Avrithis[1] Sylvaine Picard[2] Andrei Bursuc[3]
[1]Univ Rennes, Inria, CNRS, IRISA [2]Safran [3]valeo.ai

Abstract

Training deep neural networks from few examples is a highly challenging and key problem for many computer vision tasks. In this context, we are targeting knowledge transfer from a set with abundant data to other sets with few available examples. We propose two simple and effective solutions: (i) dense classification *over feature maps, which for the first time studies local activations in the domain of few-shot learning, and (ii)* implanting*, that is, attaching new neurons to a previously trained network to learn new, task-specific features. On miniImageNet, we improve the prior state-of-the-art on few-shot classification,* i.e.*, we achieve 62.5%, 79.8% and 83.8% on 5-way 1-shot, 5-shot and 10-shot settings respectively.*

1. Introduction

Current state of the art on image classification [41, 11, 15], object detection [21, 36, 10], semantic segmentation [52, 2, 20], and practically most tasks with some degree of learning involved, rely on deep neural networks. Those are powerful high-capacity models with trainable parameters ranging from millions to tens of millions, which require vast amounts of annotated data to fit. When such data is plentiful, supervised learning is the solution of choice.

Tasks and classes with limited available data, *i.e*. from the long-tail [48], are highly problematic for this type of approaches. The performance of deep neural networks poses several challenges in the low-data regime, in particular in terms of overfitting and generalization. The subject of few-shot learning is to learn to recognize previously unseen classes with very few annotated examples. This is not a new problem [4], yet there is a recent resurgence in interest through *meta-learning* [18, 44, 39, 1, 5] inspired by early work in *learning-to-learn* [43, 13].

In meta-learning settings, even when there is single large training set with a fixed number of class, it is treated as a collection of datasets of different classes, where each class has a few annotated examples. This is done so that both *meta-learning* and *meta-testing* are performed in a similar manner [44, 39, 5]. However this choice does not always come with best performance. We argue that a simple conventional pipeline using all available classes and data with a *parametric classifier* is effective and appealing.

Most few-shot learning approaches do not deal explicitly with spatial information since feature maps are usually flattened or pooled before the classification layer. We show that performing a *dense classification* over feature maps leads to more precise classification and consistently improves performance on standard benchmarks.

While *incremental learning* touches similar aspects with few-shot learning by learning to adapt to new tasks using the same network [26, 25] or extending an existing network with new layers and parameters for each new task [38], few of these ideas have been adopted in few shot learning. The main impediment is the reduced number of training examples which make it difficult to properly define a new task. We propose a solution for leveraging incremental learning ideas for few-shot learning.

Contributions: We present the following contributions. First, we propose a simple extension for few-shot learning pipelines consisting of *dense classification* over feature maps. Through localized supervision, it enables reaping additional knowledge from the limited training data. Second, we introduce neural *implants*, which are layers attached to an already trained network, enabling it to quickly adapt to new tasks with few examples. Both are easy to implement and show consistent performance gains.

2. Problem formulation and background

Problem formulation. We are given a collection of *training* examples $X := (\mathbf{x}_1, \dots, \mathbf{x}_n)$ with each $\mathbf{x}_i \in \mathcal{X}$, and corresponding labels $Y := (y_1, \dots, y_n)$ with each $y_i \in C$, where $C := [c]$[1] is a set of *base classes*. On this training data we are allowed to learn a representation of the domain $\mathcal{X}$ such that we can solve new tasks. This representation learning we shall call *stage 1*.

In few-shot learning, one new task is that we are given a collection of few *support* examples $X' := (\mathbf{x}'_1, \dots, \mathbf{x}'_{n'})$ with each $\mathbf{x}'_i \in \mathcal{X}$, and corresponding labels $Y' := (y'_1, \dots, y'_{n'})$ with each $y'_i \in C'$, where $C' := [c']$ is a set of *novel classes* disjoint from C and $n' \ll n$; with this new

[1]We use the notation $[i] := \{1, \dots, i\}$ for $i \in \mathbb{N}$.

data, the objective is to learn a classifier that maps a new *query* example from $\mathcal{X}$ to a label prediction in C'. The latter classifier learning, which does not exclude continuing the representation learning, we shall call *stage 2*.

Classification is called *c'-way* where c' is the number of novel classes; in case there is a fixed number k of support examples per novel class, it is called *k-shot*. As in standard classification, there is typically a collection of queries for evaluation of each task. Few-shot learning is typically evaluated on a large number of new tasks, with queries and support examples randomly sampled from (X', Y').

Network model. We consider a model that is conceptually composed of two parts: an embedding network and a classifier. The *embedding network* $\phi_\theta : \mathcal{X} \to \mathbb{R}^{r \times d}$ maps the input to an embedding, where θ denotes its parameters. Since we shall be studying the spatial properties of the input, the embedding is not a vector but rather a tensor, where r represents the spatial dimensions and d the feature dimensions. For a 2d input image and a convolutional network for instance, the embedding is a 3d tensor in $\mathbb{R}^{w \times h \times d}$ taken as the activation of the last convolutional layer, where $r = w \times h$ is its spatial resolution. The embedding can still be a vector in the special case $r = 1$.

The *classifier network* can be of any form and depends on the particular model, but it is applied on top of ϕ_θ and its output represents confidence over c (resp. c') base (resp. novel) classes. If we denote by $f_\theta : \mathcal{X} \to R^c$ (resp. $R^{c'}$) the *network function* mapping the input to class confidence, then a prediction for input $x \in \mathcal{X}$ is made by assigning the label of maximum confidence, $\arg\max_i f^i_\theta(x)$[2].

Prototypical networks. Snell *et al*. [42] introduce a simple classifier for novel classes that computes a single prototype per class and then classifies a query to the nearest prototype. More formally, given X', Y' and an index set $S \subset N' := [n']$, let the set $S_j := \{i \in S : y'_i = j\}$ index the support examples in S labeled in class j. The *prototype* of class j is given by the average of those examples

$$\mathbf{p}_j = \frac{1}{|S_j|} \sum_{i \in S_j} \phi_\theta(\mathbf{x}'_i) \tag{1}$$

for $j \in C'$. Then, the network function is defined as[3]

$$f_\theta[P](\mathbf{x}) := \boldsymbol{\sigma}\left([s(\phi_\theta(\mathbf{x}), \mathbf{p}_j)]_{j=1}^{c'}\right) \tag{2}$$

for $\mathbf{x} \in \mathcal{X}$, where $P := (\mathbf{p}_1, \ldots, \mathbf{p}_{c'})$ and s is a similarity function that may be cosine similarity or negative squared Euclidean distance and $\boldsymbol{\sigma} : \mathbb{R}^m \to \mathbb{R}^m$ is the *softmax function* defined by

$$\boldsymbol{\sigma}(\mathbf{x}) := \left[\frac{\exp(x^j)}{\sum_{i=1}^m \exp(x^i)}\right]_{j=1}^m \tag{3}$$

for $\mathbf{x} \in \mathbb{R}^m$ and $m \in \mathbb{N}$.

Given a new task with support data (X', Y') over novel classes C' (stage 2), the full index set N' is used and computing class prototypes (1) is the only learning to be done.

When learning from the training data (X, Y) over base classes C (stage 1), a number of fictitious tasks called *episodes* are generated by randomly sampling a number classes from C and then a number of examples in each class from X with their labels from Y; these collections, denoted as X', Y' respectively and of length n', are supposed to be support examples and queries of novel classes C', where labels are now available for the queries and the objective is that queries are classified correctly. The set $N' := [n']$ is partitioned into a *support set* $S \subset N'$ and a *query set* $Q := N' \setminus S$. Class prototypes P are computed on index set S according to (1) and the network function f_θ is defined on these prototypes by (2). The network is then trained by minimizing over θ the *cost function*

$$J(X', Y'; \theta) := \sum_{i \in Q} \ell(f_\theta[P](\mathbf{x}'_i), y'_i) \tag{4}$$

on the query set Q, where ℓ is the *cross-entropy* loss

$$\ell(\mathbf{a}, y) := -\log a^y \tag{5}$$

for $\mathbf{a} \in \mathbb{R}^m$, $y \in [m]$ and $m \in \mathbb{N}$.

Learning with imprinted weights. Qi *et al*. [32] follow a simpler approach when learning on the training data (X, Y) over base classes C (stage 1). In particular, they use a fully-connected layer without bias as a *parametric linear classifier* on top of the embedding function ϕ_θ followed by softmax and they train in a standard supervised classification setting. More formally, let $\mathbf{w}_j \in \mathbb{R}^{r \times d}$ be the weight parameter of class j for $j \in C$. Then, similarly to (2), the network function is defined by

$$f_{\theta,W}(\mathbf{x}) := \boldsymbol{\sigma}\left([s_\tau(\phi_\theta(\mathbf{x}), \mathbf{w}_j)]_{j=1}^{c}\right) \tag{6}$$

for $\mathbf{x} \in \mathcal{X}$, where $W := (\mathbf{w}_1, \ldots, \mathbf{w}_c)$ is the collection of class weights and s_τ is the *scaled cosine similarity*

$$s_\tau(\mathbf{x}, \mathbf{y}) := \tau \langle \hat{\mathbf{x}}, \hat{\mathbf{y}} \rangle \tag{7}$$

for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{r \times d}$; $\hat{\mathbf{x}} := \mathbf{x} / \|\mathbf{x}\|$ is the ℓ_2-normalized counterpart of $\mathbf{x}$ for $\mathbf{x} \in \mathbb{R}^{r \times d}$; $\langle \cdot, \cdot \rangle$ and $\|\cdot\|$ denote Frobenius inner product and norm respectively; and $\tau \in \mathbb{R}^+$ is a trainable scale parameter. Then, training amounts to minimizing over θ, W the *cost function*

$$J(X, Y; \theta, W) := \sum_{i=1}^n \ell(f_{\theta,W}(\mathbf{x}_i), y_i). \tag{8}$$

[2] Given vector $\mathbf{x} \in \mathbb{R}^m$, x^i denotes the i-th element of $\mathbf{x}$. Similarly for $f : A \to \mathbb{R}^m$, $f^i(a)$ denotes the i-the element of $f(a)$ for $a \in A$.

[3] We define $[e(i)]_{i=1}^n := (e(1), \ldots, e(n))$ for $n \in \mathbb{N}$ and any expression $e(i)$ of variable $i \in \mathbb{N}$.

Given a new task with support data (X', Y') over novel classes C' (stage 2), class prototypes P are computed on N' according to (1) and they are *imprinted* in the classifier, that is, W is replaced by $W' := (W, P)$. The network can now make predictions on $n + n'$ base and novel classes. The network is then fine-tuned based on (8), which aligns the class weights W with the prototypes P at the cost of having to store and re-train on the entire training data (X, Y).

Few-shot learning without forgetting. Gidaris and Komodakis [6], concurrently with [32], develop a similar model that is able to classify examples of both base and novel classes. The main difference to [33] is that only the weight parameters of the base classes are stored and not the entire training data. They use the same *parametric* linear classifier as [32] in both stages, and they also use episode-style training like [42] in stage 2.

3. Method

Given training data of base classes (stage 1), we use a parametric classifier like [32, 6], which however applies at all spatial locations rather than following flattening or pooling; a very simple idea that we call *dense classification* and discuss in §3.1. Given support data of novel classes (stage 2), we learn in episodes as in prototypical networks [42], but on the true task. As discussed in §3.2, the embedding network learned in stage 1 remains fixed but new layers called *implants* are trained to learn task-specific features. Finally, §3.3 discusses inference of novel class queries.

3.1. Dense classification

As discussed in §2, the *embedding network* $\phi_\theta : \mathcal{X} \to \mathbb{R}^{r \times d}$ maps the input to an embedding that is a tensor. There are two common ways of handling this high-dimensional representation, as illustrated in Figure 1.

The first is to apply one or more fully connected layers, for instance in networks C64F, C128F in few-shot learning [44, 42, 6]. This can be seen as *flattening* the activation into a long vector and multiplying with a weight vector of the same length per class; alternatively, the weight parameter is a tensor of the same dimension as the embedding. This representation is discriminative, but not invariant.

The second way is to apply *global pooling* and reduce the embedding into a smaller vector of length d, for instance in small ResNet architectures used more recently in few-shot learning [27, 6, 31]. This reduces dimensionality significantly, so it makes sense if d is large enough. It is an invariant representation, but less discriminative.

In this work we follow a different approach that we call *dense classification* and is illustrated in Figure 2. We view the embedding $\phi_\theta(\mathbf{x})$ as a collection of vectors $[\phi^{(k)}(\mathbf{x})]_{k=1}^r$, where $\phi^{(k)}(\mathbf{x}) \in \mathbb{R}^d$ for $k \in [r]$[4]. For a 2d

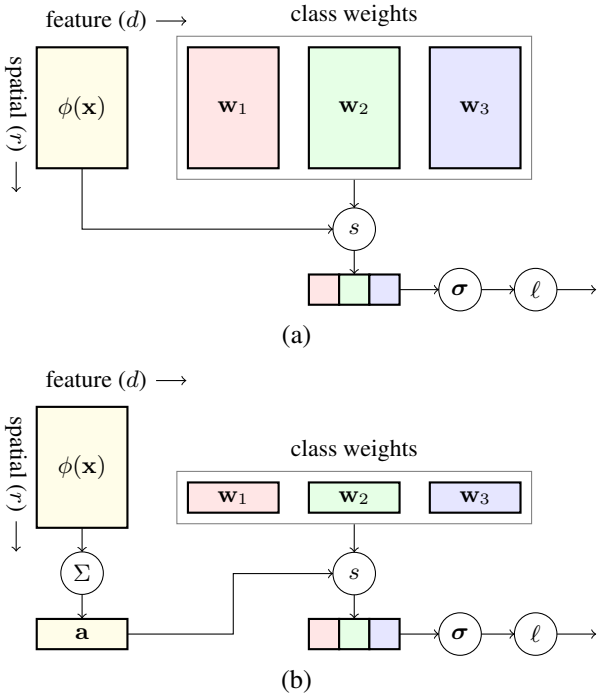


Figure 1. *Flattening and pooling*. Horizontal (vertical) axis represents feature (spatial) dimensions. Tensors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$ represent class weights, and $\phi(\mathbf{x})$ the embedding of example $\mathbf{x}$. An embedding is compared to class weights by similarity (s) and then softmax ($\boldsymbol{\sigma}$) and cross-entropy (ℓ) follow. (a) *Flattening* is equivalent to class weights having the same $r \times d$ shape as $\phi(\mathbf{x})$. (b) *Global pooling*. Embedding $\phi(\mathbf{x})$ is pooled (Σ) into vector $\mathbf{a} \in \mathbb{R}^d$ before being compared to class weights, which are in $\mathbb{R}^d$ too.

image input and a convolutional network, $\phi_\theta(\mathbf{x})$ consists of the activations of the last convolutional layer, that is a tensor in $\mathbb{R}^{w \times h \times d}$ where $r = w \times h$ is its spatial resolution. Then, $\phi^{(k)}(\mathbf{x})$ is an embedding in $\mathbb{R}^d$ that represents a single spatial location k on the tensor.

When learning from the training data (X, Y) over base classes C (stage 1), we adopt the simple approach of training a *parametric linear classifier* on top of the embedding function ϕ_θ, like [32] and the initial training of [6]. The main difference in our case is that the weight parameters do *not* have the same dimensions as $\phi_\theta(\mathbf{x})$; they are rather vectors in $\mathbb{R}^d$ and they are *shared* over all spatial locations. More formally, let $\mathbf{w}_j \in \mathbb{R}^d$ be the weight parameter of class j for $j \in C$. Then, similarly to (6), the classifier mapping $f_{\theta,W} : \mathcal{X} \to \mathbb{R}^{r \times c}$ is defined by

$$f_{\theta,W}(\mathbf{x}) := \left[\boldsymbol{\sigma} \left([s_\tau(\phi_\theta^{(k)}(\mathbf{x}), \mathbf{w}_j)]_{j=1}^c \right) \right]_{k=1}^r \qquad (9)$$

for $\mathbf{x} \in \mathcal{X}$, where $W := (\mathbf{w}_1, \dots, \mathbf{w}_c)$ is the collection of class weights and s_τ is the *scaled cosine similarity* defined by (7), with τ being a learnable parameter as in [32, 6][5].

[4] Given tensor $\mathbf{a} \in \mathbb{R}^{m \times n}$, denote by $\mathbf{a}^{(k)}$ the k-th n-dimensional slice along the first group of dimensions for $k \in [m]$.

[5] *Temperature scaling* is frequently encountered in various formats in

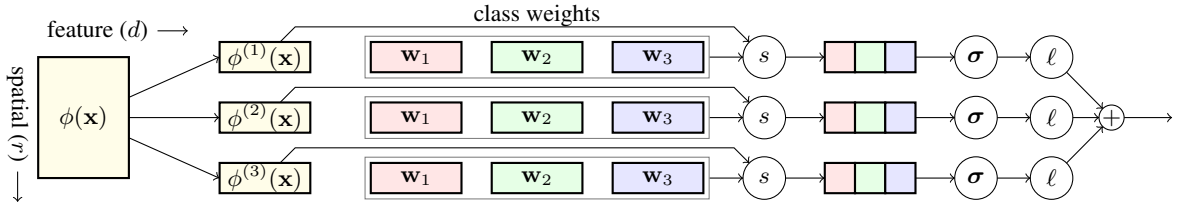
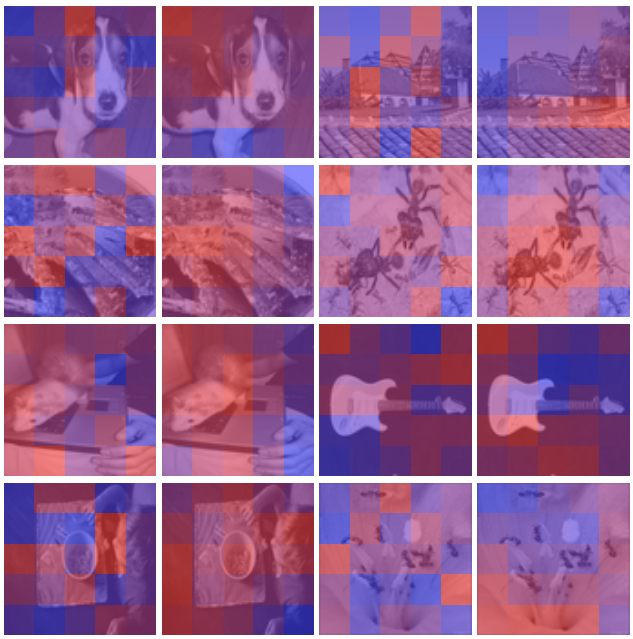


Figure 2. *Dense classification*. Notation is the same as in Figure 1. The embedding $\mathbf{a} := \phi(\mathbf{x}) \in \mathbb{R}^{r \times d}$ is seen as a collection of vectors $(\mathbf{a}^{(1)}, \dots, \mathbf{a}^{(r)})$ in $\mathbb{R}^d$ (here $r = 3$) with each being a vector in $\mathbb{R}^d$ and representing a region of the input image. Each vector is compared independently to the same class weights and the losses are added, encouraging all regions to be correctly classified.



pooling dense pooling dense

Figure 3. Examples overlaid with correct *class activation maps* [54] (red is high activation for ground truth) on Resnet-12 (*cf*. §5) trained with global average pooling or dense classification (*cf*. (9)). From top to bottom: base classes, classified correctly by both (walker hound, tile roof); novel classes, classified correctly by both (king crab, ant); novel classes, dense classification is better (ferret, electric guitar); novel classes, pooling is better (mixing bowl, ant). In all cases, dense classification results in smoother activation maps that are more aligned with objects.

Here $f_{\theta,W}(\mathbf{x})$ is a $r \times c$ tensor: index k ranges over spatial resolution $[r]$ and j over classes $[c]$.

This operation is a 1×1 *convolution followed by depth-wise softmax*. Then, $f^{(k)}_{\theta,W}(\mathbf{x})$ at spatial location k is a vector in $\mathbb{R}^c$ representing confidence over the c classes. On the other hand, $f^{(:,j)}_{\theta,W}(\mathbf{x})$ is a vector in $\mathbb{R}^r$ representing confidence of class j for $j \in [c]$ as a function of spatial location.[6] For a 2d image input, $f^{(:,j)}_{\theta,W}(\mathbf{x})$ is like a *class activation map* (CAM) [54] for class j, that is a 2d map roughly localizing the response to class j, but differs in that softmax suppresses all but the strongest responses at each location.

Given the definition (9) of $f_{\theta,W}$, training amounts to minimizing over θ, W the *cost function*

$$J(X, Y; \theta, W) := \sum_{i=1}^{n} \sum_{k=1}^{r} \ell(f^{(k)}_{\theta,W}(\mathbf{x}_i), y_i), \qquad (10)$$

where ℓ is cross-entropy (5). *The loss function applies to all spatial locations and therefore the classifier is encouraged to make correct predictions everywhere*.

Learning a new task with support data (X', Y') over novel classes C' (stage 2) and inference are discussed in §3.2.2 and §3.3 respectively.

Discussion. The same situation arises in *semantic segmentation* [23, 30], where given per-pixel labels, the loss function applies per pixel and the network learns to make localized predictions on upsampled feature maps rather than just classify. In our case there is just one image-level label and the low resolution, *e.g.* 5×5, of few-shot learning settings allows us to assume that the label applies to all locations due to large receptive field.

Dense classification improves the spatial distribution of class activations, as shown in Figure 3. By encouraging all spatial locations to be classified correctly, we are encouraging the embedding network to identify all parts of the object of interest rather than just the most discriminative details. Since each location on a feature map corresponds to a region in the image where only part of the object may be visible, our model behaves like *implicit data augmentation* of exhaustive shifts and crops over a dense grid with a single forward pass of each example in the network.

3.2. Implanting

From the learning on the training data (X, Y) of base classes C (stage 1) we only keep the embedding network ϕ_θ and we discard the classification layer. The assumption is that features learned on base classes are generic enough to be used for other classes, at least for the bottom layers [51]. However, given a new few-shot task on novel classes C' (stage 2), we argue that we can take advantage of the sup-

several works to enable soft-labeling [12] or to improve cosine similarity in the final layer [46, 31, 6, 32, 14].

[6] Given tensor $\mathbf{a} \in \mathbb{R}^{m \times n}$, denote by $\mathbf{a}^{(:,j)}$ the j-th m-dimensional slice along the second group of dimensions for $j \in [n]$.

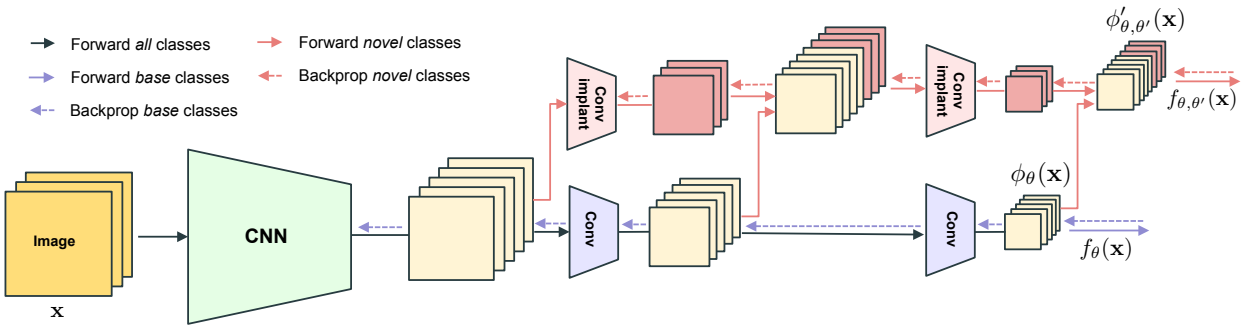


Figure 4. *Neural implants for CNNs.* The implants are convolutional filters operating in a new processing stream parallel to the base network. The input of an implant is the depth-wise concatenation of hidden states from both streams. When training neural implants, previously trained parameters are frozen. Purple and black arrows correspond to stage 1 flows; red and black to stage 2.

port data (X', Y') to find new features that are discriminative for the task at hand, at least in the top layers.

3.2.1 Architecture

We begin with the embedding network ϕ_θ, which we call *base network*. We widen this network by adding new convolution kernels in a number of its top convolutional layers. We call these new neurons *implants*. While learning the implants, we keep the base network parameters frozen, which preserves the representation of the base classes.

Let $\mathbf{a}_l$ denote the output activation of the convolutional layer l in the base network. The implant for this layer, if it exists, is a distinct convolutional layer with output activation $\mathbf{a}'_l$. Then the input of an implant at the next layer $l+1$ is the depth-wise concatenation $[\mathbf{a}_l, \mathbf{a}'_l]$ if $\mathbf{a}'_l$ exists, and just $\mathbf{a}_l$ otherwise. If θ'_l are the parameters of the l-th implant, then we denote by $\theta' := (\theta'_{l_0}, \dots, \theta'_L)$ the set of all new parameters, where l_0 is the first layer with an implant and L the network depth. The *widened* embedding network is denoted by $\phi_{\theta,\theta'}$.

As illustrated in Figure 4, we are creating a new stream of data in parallel to the base network. The implant stream is connected to the base stream at multiple top layers and leverages the previously learned features by learning additional connections for the new tasks.

Why implanting? In several few-shot learning works, in particular metric learning, it is common to focus on the top layer of the network and learn or generate a new classifier for the novel classes. The reason behind this choice underpins a major challenge in few-shot learning: deep neural networks are prone to overfitting. With implanting, we attempt to diminish this risk by adding a limited amount of new parameters, while preserving the previously trained ones intact. Useful visual representations and parameters learned from base classes can be quickly squashed during fine-tuning on the novel classes. With implants, we *freeze* them and train only the new neurons added to the network, maximizing the contribution of the knowledge acquired from base classes.

3.2.2 Training

To learn the implants only makes sense when a new task is given with support data (X', Y') over novel classes C' (stage 2). Here we use an approach similar to prototypical networks [42] in the sense that we generate a number of fictitious *subtasks* of the new task, the main difference being that we are now working on the novel classes.

We choose the simple approach of using each one of the given examples alone as a query in one subtask while all the rest are used as support examples. This involves no sampling and the process is deterministic. Because only one example is missing from the true support examples, each subtask approximates the true task very well.

In particular, for each $i \in N' := [n']$, we define a *query set* $Q_i := \{i\}$ and a *support set* $S_i := N' \setminus Q_i$. We compute class prototypes P_i on index set S_i according to (1), where we replace ϕ_θ by $\phi_{\theta,\theta'}$ and θ' are the implanted parameters. We define the *widened* network function $f_{\theta,\theta'}[P_i]$ on these prototypes by (2) with a similar replacement. We then freeze the base network parameters θ and train the implants θ' by minimizing a cost function like (4). Similarly to (4) and taking all subtasks into account, the overall cost function we are minimizing over θ' is given by

$$J(X', Y'; \theta, \theta') := \sum_{i=1}^{n'} \ell(f_{\theta,\theta'}[P_i](\mathbf{x}'_i), y'_i), \qquad (11)$$

where ℓ is cross-entropy (5).

In (11), activations are assumed flattened or globally pooled. Alternatively, we can *densely classify* them and apply the loss function to all spatial locations independently. Combining with (10), the cost function in this case is

$$J(X', Y'; \theta, \theta') := \sum_{i=1}^{n'} \sum_{k=1}^{r} \ell(f^{(k)}_{\theta,\theta'}[P_i](\mathbf{x}'_i), y'_i). \qquad (12)$$

Prototypes in (11) or (12) are recomputed at each iteration based on the current version of implants. Note that this training setup does not apply to the 1-shot scenario as it requires at least two *support* samples per class.

3.3. Inference on novel classes

Inference is the same whether the embedding network has been implanted or not. Here we adopt the prototypical network model too. What we have found to work best is to perform global pooling of the embeddings of the support examples and compute class prototypes $P := (\mathbf{p}_1, \dots, \mathbf{p}_{c'})$ by (1). Given a query $\mathbf{x} \in \mathcal{X}$, the standard prediction is then to assign it to the nearest prototype

$$\arg\max_{j \in C'} s(\phi_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j), \tag{13}$$

where s is cosine similarity [42]. Alternatively, we can *densely classify* the embedding $\phi_{\theta,\theta'}(\mathbf{x})$, soft-assigning independently the embedding $\phi^{(k)}_{\theta,\theta'}(\mathbf{x})$ of each spatial location, then average over all locations $k \in [r]$ according to

$$f_{\theta,\theta'}[P](\mathbf{x}) := \frac{1}{r} \sum_{k=1}^{r} \boldsymbol{\sigma}\left([s_\tau(\phi^{(k)}_{\theta,\theta'}(\mathbf{x}), \mathbf{p}_j)]_{j=1}^{c'} \right), \tag{14}$$

where s_τ is the scaled cosine similarity (7), and finally classify to $\arg\max_{j \in C'} f^j_{\theta,\theta'}[P](\mathbf{x})$.

4. Related work

Metric learning is common in few-shot learning. Multiple improvements of the standard softmax and cross-entropy loss are proposed by [49, 22, 53, 45, 9] to this end. Traditional methods like *siamese networks* are also considered [3, 40, 18] along with models that learn by comparing multiple samples at once [44, 50, 42]. Learning to generate new samples [9] is another direction. Our solution is related to *prototypical networks* [42] and *matching networks* [44] but we rather use a parametric classifier.

Meta-learning is the basis of a large portion of the few-shot learning literature. Recent approaches can be roughly classified as: *optimization-based methods*, that learn to initialize the parameters of a learner such that it becomes faster to fine-tune [5, 28, 29]; *memory-based methods* leveraging memory modules to store training samples or to encode adaptation algorithms [39, 35]; *data generation methods* that learn to generate new samples [47]; *parameter generating methods* that learn to generate the weights of a classifier [6, 33] or the parameters of a network with multiple layers [1, 7, 48, 8]. The motivation behind the latter is that it should be easier to generate new parameters rather than to fine-tune a large network or to train a new classifier from scratch. By generating a single linear layer at the end of the network [6, 32, 33], one neglects useful coarse visual information found in intermediate layers. We plug our neural implants at multiple depth levels, taking advantage of such features during fine-tuning and learning new ones.

Network	Pooling	1-shot	5-shot	10-shot
C128F	GAP	54.28 ±0.18	71.60 ±0.13	76.92 ±0.12
C128F	DC	49.84 ±0.18	69.64 ±0.15	74.61 ±0.13
ResNet-12	GAP	58.61 ±0.18	76.40 ±0.13	80.76 ±0.11
ResNet-12	DC	61.26 ±0.20	79.01 ±0.13	83.04 ±0.12

Table 1. *Average 5-way accuracy on novel classes of* mini*ImageNet, stage 1 only.* Pooling refers to stage 1 training. GAP: global average pooling; DC: dense classification. At testing, we use global max-pooling on queries for models trained with dense classification, and global average pooling otherwise.

Network adaptation is common when learning a new task or new domain. One solution is to learn to *mask* part of the network, keeping useful neurons and re-training/fine-tuning the remaining neurons on the new-task [25, 26]. Rusu *et al*. [38] rather *widen* the network by adding new neurons in parallel to the old ones at every layer. New neurons receive data from all hidden states, while previously generated weights are frozen when training for the new task. Our neural implants are related to [38] as we add new neurons in parallel and freeze the old ones. Unlike [38], we focus on low-data regimes, keeping the number of new implanted neurons small to diminish overfitting risks and train faster, and adding them only at top layers, taking advantage of generic visual features from bottom layers [51].

5. Experiments

We evaluate our method extensively on the *mini*ImageNet and FC100 datasets. We describe the experimental setup and report results below.

5.1. Experimental setup

Networks. In most experiments we use a ResNet-12 network [31] as our embedding network, composed of four residual blocks [11], each having three 3×3 convolutional layers with batch normalization [16] and swish-1 activation function [34]. Each block is followed by 2×2 max-pooling. The shortcut connections have a convolutional layer to adapt to the right number of channels. The first block has 64 channels, which is doubled at each subsequent block such that the output has depth 512. We also test dense classification on a lighter network C128F [6] composed of four convolutional layers, the first (last) two having 64 (128) channels, each followed by 2×2 max-pooling.

Datasets. We use ***mini*ImageNet [44]**, a subset of ImageNet ILSVRC-12 [37] of 60,000 images of resolution 84×84, uniformly distributed over 100 classes. We use the split proposed in [35]: $C = 64$ classes for training, 16 for validation and 20 for testing.

Stage 1 training		Support/query pooling at testing			
	Support →	GMP		GAP	
	Queries →	GMP	DC	GAP	DC
Global average pooling	Base classes	63.55 ±0.20	77.17 ±0.11	79.37 ±0.09	77.15 ±0.11
	Novel classes	72.25 ±0.13	70.71 ±0.14	76.40 ±0.13	73.28 ±0.14
	Both classes	37.74 ±0.07	38.65 ±0.05	56.25 ±0.10	54.80 ±0.09
Dense classification	Base classes	79.28 ±0.10	**80.67 ±0.10**	**80.61 ±0.10**	**80.70 ±0.10**
	Novel classes	**79.01 ±0.13**	77.93 ±0.13	78.55 ±0.13	**78.95 ±0.13**
	Both classes	42.45 ±0.07	57.98 ±0.10	67.53 ±0.10	**67.78 ±0.10**

Table 2. *Average 5-way 5-shot accuracy on base, novel and both classes of* mini*ImageNet with ResNet-12, stage 1 only.* GMP: global max-pooling; GAP: global average pooling; DC: dense classification. Bold: accuracies in the confidence interval of the best one.

Stage 2 training		Query pooling at testing		
Support	Queries	GAP	GMP	DC
GMP	GMP	79.03 ± 0.19	78.92 ± 0.19	79.04 ± 0.19
GMP	DC	79.06 ± 0.19	79.37 ± 0.18	79.15 ± 0.19
GAP	GAP	79.62 ± 0.19	74.57 ± 0.22	**79.77 ± 0.19**
GAP	DC	79.56 ± 0.19	74.58 ± 0.22	79.52 ± 0.19

Table 3. *Average 5-way 5-shot accuracy on novel classes of* mini*ImageNet with ResNet-12 and implanting in stage 2.* At testing, we use GAP for support examples. GMP: global max-pooling; GAP: global average pooling; DC: dense classification.

We also use **FC100**, a few-shot version of CIFAR-100 recently proposed by Oreshkin *et al*. [31]. Similarily to *mini*ImageNet, CIFAR-100 [19] has 100 classes of 600 images each, although the resolution is 32×32. The split is $C = 60$ classes for training, 20 for validation and 20 for testing. Given that all classes are grouped into 20 super-classes, this split does not separate super-classes: classes are more similar in each split and the semantic gap between base and novel classes is larger.

Evaluation protocol. The training set X comprises images of the base classes C. To generate the support set X' of a few-shot task on novel classes, we randomly sample C' classes from the validation or test set and from each class we sample k images. We report the *average accuracy* and the corresponding 95% *confidence interval* over a number of such tasks. More precisely, for all implanting experiments, we sample 5,000 few-shot tasks with 30 queries per class, while for all other experiments we sample 10,000 tasks. Using the same task sampling, we also consider few-shot tasks involving *base classes* C, following the benchmark of [6]. We sample a set of extra images from the base classes to form a test set for this evaluation, which is performed in two ways: independently of the novel classes C' and jointly on the union $C \cup C'$. In the latter case, base prototypes learned at stage 1 are concatenated with novel prototypes [6].

Implementation details. In stage 1, we train the embedding network for 8,000 (12,500) iterations with mini-batch size 200 (512) on *mini*ImageNet (FC100). On *mini*ImageNet, we use stochastic gradient descent with Nesterov momentum. On FC100, we rather use Adam optimizer [17]. We initialize the scale parameter at $\tau = 10$ (100) on *mini*ImageNet (FC100). For a given few-shot task in stage 2, the implants are learned over 50 epochs with AdamW optimizer [24] and scale fixed at $\tau = 10$.

5.2. Results

Networks. In Table 1 we compare ResNet-12 to C128F, with and without dense classification. We observe that dense classification improves classification accuracy on novel classes for ResNet-12, but it is detrimental for the small network. C128F is only 4 layers deep and the receptive field at the last layer is significantly smaller than the one of ResNet-12, which is 12 layers deep. It is thus likely that units from the last feature map correspond to non-object areas in the image. Regardless of the choice of using dense classification or not, ResNet-12 has a large performance gap over C128F. For the following experiments, we use exclusively ResNet-12 as our embedding network.

Dense classification. To evaluate stage 1, we skip stage 2 and directly perform testing. In Table 2 we evaluate 5-way 5-shot classification on *mini*ImageNet with global average pooling and dense classification at stage 1 training, while exploring different pooling strategies at inference. We also tried using global max-pooling at stage 1 training and got similar results as with global average pooling. Dense classification in stage 1 training outperforms global average pooling in all cases by a large margin. It also improves the ability of the network to integrate new classes without forgetting the base ones. Using dense classification at testing as well, the accuracy on both classes is 67.78%, outperforming the best result of 59.35% reported by [6]. At testing, dense classification of the queries with global average pooling of the support samples is the best overall choice. One exception is global max-pooling on both the support and query samples, which gives the highest accuracy for new classes but the difference is insignificant.

Implanting. In stage 2, we add implants of 16 channels to

Method	1-shot	5-shot	10-shot
GAP	58.61 ± 0.18	76.40 ± 0.13	80.76 ± 0.11
DC (ours)	$\mathbf{62.53 \pm 0.19}$	78.95 ± 0.13	82.66 ± 0.11
DC + WIDE	61.73 ± 0.19	78.25 ± 0.14	82.03 ± 0.12
DC + IMP (ours)	-	$\mathbf{79.77 \pm 0.19}$	$\mathbf{83.83 \pm 0.16}$
MAML [5]	48.70 ± 1.8	63.10 ± 0.9	-
PN [42]	49.42 ± 0.78	68.20 ± 0.66	-
Gidaris et al. [6]	55.45 ± 0.7	73.00 ± 0.6	-
PN [31]	56.50 ± 0.4	74.20 ± 0.2	78.60 ± 0.4
TADAM [31]	58.50	76.70	80.80

Table 4. *Average 5-way accuracy on novel classes of* mini-*ImageNet*. The top part is our solutions and baselines, all on ResNet-12. GAP: global average pooling (stage 1); DC: dense classification (stage 1); WIDE: last residual block widened by 16 channels (stage 1); IMP: implanting (stage 2). In stage 2, we use GAP on both support and queries. At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. PN: Prototypical Network [42]. MAML [5] and PN [42] use four-layer networks; while PN [31] and TADAM [31] use the same ResNet-12 as us. Gidaris et al. [6] use a Residual network of comparable complexity to ours.

Method	1-shot	5-shot	10-shot
GAP	41.02 ± 0.17	56.63 ± 0.16	61.65 ± 0.15
DC (ours)	$\mathbf{42.04 \pm 0.17}$	57.05 ± 0.16	61.91 ± 0.16
DC + IMP (ours)	-	$\mathbf{57.63 \pm 0.23}$	$\mathbf{62.91 \pm 0.22}$
PN [31]	37.80 ± 0.40	53.30 ± 0.50	58.70 ± 0.40
TADAM [31]	40.10 ± 0.40	56.10 ± 0.40	61.60 ± 0.50

Table 5. *Average 5-way accuracy on novel classes of FC100 with ResNet-12*. The top part is our solutions and baselines. GAP: global average pooling (stage 1); DC: dense classification (stage 1); IMP: implanting (stage 2). In stage 2, we use GAP on both support and queries. At testing, we use GAP on support examples and GAP or DC on queries, depending on the choice of stage 1. The bottom part results are as reported in the literature. All experiments use the same ResNet-12.

all convolutional layers of the last residual block of our embedding network pretrained in stage 1 on the base classes with dense classification. The implants are trained on the few examples of the novel classes and then used as an integral part of the widened embedding network $\phi_{\theta,\theta'}$ at testing. In Table 3, we evaluate different pooling strategies for support examples and queries in stage 2. Average pooling on both is the best choice, which we keep in the following.

Ablation study. In the top part of Table 4 we compare our best solutions with a number of baselines on 5-way *mini*ImageNet classification. One baseline is the embedding network trained with global average pooling in stage 1. Dense classification remains our best training option. In stage 2, the implants are able to further improve on the results of dense classification. To illustrate that our gain does not come just from having more parameters and greater feature dimensionality, another baseline is to compare it to widening the last residual block of the network by 16 channels in stage 1. It turns out that such widening does not bring any improvement on novel classes. Similar conclusions can be drawn from the top part of Table 5, showing corresponding results on FC100. The difference between different solutions is less striking here. This may be attributed to the lower resolution of CIFAR-100, allowing for less gain from either dense classification or implanting, since there may be less features to learn.

Comparison with the state-of-the-art. In the bottom part of Table 4 we compare our model with previous few-shot learning methods on the same 5-way *mini*ImageNet classification. All our solutions outperform by a large margin other methods on 1, 5 and 10-shot classification. Our implanted network sets a new state-of-the art for 5-way 5-shot classification of *mini*ImageNet. Note that prototypical network on ResNet-12 [31] is already giving very competitive performance. TADAM [31] builds on top of this baseline to achieve the previous state of the art. In this work we rather use a cosine classifier in stage 1. This setting is our baseline GAP and is already giving similar performance to TADAM [31]. Dense classification and implanting are both able to improve on this baseline. Our best results are at least 3% above TADAM [31] in all settings. Finally, in the bottom part of Table 5 we compare our model on 5-way FC100 classification against prototypical network [31] and TADAM [31]. Our model outperforms TADAM here too, though by a smaller margin.

6. Conclusion

In this work we contribute to few-shot learning by building upon a simplified process for learning on the base classes using a standard parametric classifier. We investigate for the first time in few-shot learning the activation maps and devise a new way of handling spatial information by a dense classification loss that is applied to each spatial location independently, improving the spatial distribution of the activation and the performance on new tasks. It is important that the performance benefit comes with deeper network architectures and high-dimensional embeddings. We further adapt the network for new tasks by implanting neurons with limited new parameters and without changing the original embedding. Overall, this yields a simple architecture that outperforms previous methods by a large margin and sets a new state of the art on standard benchmarks.

References

[1] L. Bertinetto, J. F. Henriques, J. Valmadre, P. Torr, and A. Vedaldi. Learning feed-forward one-shot learners. In *NIPS*, 2016. 1, 6

[2] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 1

[3] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005. 6

[4] L. Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Trans. PAMI*, 2006. 1

[5] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 1, 6, 8

[6] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018. 3, 4, 6, 7, 8

[7] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *ICLR*, 2017. 6

[8] C. Han, S. Shan, M. Kan, S. Wu, and X. Chen. Face recognition with contrastive convolution. In *ECCV*, 2018. 6

[9] B. Hariharan and R. B. Girshick. Low-shot visual recognition by shrinking and hallucinating features. *ICCV*, 2017. 6

[10] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *ICCV*, 2017. 1

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 6

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 4

[13] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 2001. 1

[14] E. Hoffer, I. Hubara, and D. Soudry. Fix your classifier: the marginal value of training the last weight layer. *ICLR*, 2018. 4

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 1

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 6

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. 7

[18] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICMLW*, 2015. 1, 6

[19] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 7

[20] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *CVPR*, 2018. 1

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016. 1

[22] W. Liu, Y. Wen, Z. Yu, and M. Yang. Large-margin softmax loss for convolutional neural networks. In *ICML*, 2016. 6

[23] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 4

[24] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017. 7

[25] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *ECCV*, 2018. 1, 6

[26] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *CVPR*, 2018. 1, 6

[27] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. Meta-learning with temporal convolutions. *arXiv preprint arXiv:1707.03141*, 2017. 3

[28] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. *ICLR*, 2018. 6

[29] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *CoRR, abs/1803.02999*, 2018. 6

[30] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 4

[31] B. N. Oreshkin, A. Lacoste, and P. Rodriguez. Tadam: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*, 2018. 3, 4, 6, 7, 8

[32] H. Qi, M. Brown, and D. G. Lowe. Low-shot learning with imprinted weights. In *CVPR*, 2018. 2, 3, 4, 6

[33] S. Qiao, C. Liu, W. Shen, and A. Yuille. Few-shot image recognition by predicting parameters from activations. *CVPR*, 2018. 3, 6

[34] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *ICLR*, 2018. 6

[35] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017. 6

[36] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. 1

[37] 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. *arXiv*, 2014. 6

[38] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 1, 6

[39] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, 2016. 1, 6

[40] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 6

[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[42] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017. 2, 3, 5, 6, 8

[43] S. Thrun. Lifelong learning algorithms. In *Learning to learn*. 1998. 1

[44] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *NIPS*, 2016. 1, 3, 6

[45] W. Wan, Y. Zhong, T. Li, and J. Chen. Rethinking feature distribution for loss functions in image classification. In *CVPR*, 2018. 6

[46] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: l 2 hypersphere embedding for face verification. In *ACM Multimedia*, 2017. 4

[47] Y.-X. Wang, R. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. 2018. 6

[48] Y.-X. Wang, D. Ramanan, and M. Hebert. Learning to model the tail. In *NIPS*, 2017. 1, 6

[49] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *ECCV*, 2016. 6

[50] F. S. Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018. 6

[51] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014. 4, 6

[52] F. Yu, V. Koltun, and T. A. Funkhouser. Dilated residual networks. In *CVPR*, 2017. 1

[53] Y. Zheng, D. K. Pal, and M. Savvides. Ring loss: Convex feature normalization for face recognition. In *CVPR*, 2018. 6

[54] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *CVPR*, 2016. 4