Meta-Transfer Learning for Few-Shot Learning

Qianru Sun[1,3] Yaoyao Liu[2*] Tat-Seng Chua[1] Bernt Schiele[3]

[1]National University of Singapore [2]Tianjin University
[3]Max Planck Institute for Informatics, Saarland Informatics Campus
`{qsun, schiele}@mpi-inf.mpg.de`
`yaoyaoliu@outlook.com {dcssq, dcscts}@nus.edu.sg`

Abstract

Meta-learning has been proposed as a framework to address the challenging few-shot learning setting. The key idea is to leverage a large number of similar few-shot tasks in order to learn how to adapt a base-learner to a new task for which only a few labeled samples are available. As deep neural networks (DNNs) tend to overfit using a few samples only, meta-learning typically uses shallow neural networks (SNNs), thus limiting its effectiveness. In this paper we propose a novel few-shot learning method called ***meta-transfer learning (MTL)*** *which learns to adapt a* ***deep NN*** *for* ***few shot learning tasks****. Specifically,* meta *refers to training multiple tasks, and* transfer *is achieved by learning scaling and shifting functions of DNN weights for each task. In addition, we introduce the* ***hard task (HT) meta-batch*** *scheme as an effective learning curriculum for MTL. We conduct experiments using (5-class, 1-shot) and (5-class, 5-shot) recognition tasks on two challenging few-shot learning benchmarks: miniImageNet and Fewshot-CIFAR100. Extensive comparisons to related works validate that our* ***meta-transfer learning*** *approach trained with the proposed* ***HT meta-batch*** *scheme achieves top performance. An ablation study also shows that both components contribute to fast convergence and high accuracy*[1].

1. Introduction

While deep learning systems have achieved great performance when sufficient amounts of labeled data is available [50, 14, 40], there has been growing interest in reducing the required amount of data. Few-shot learning tasks have been defined for this purpose. The aim is to learn new concepts from few labeled examples only, e.g. 1-shot learning [21]. While humans tend to be highly effective in this

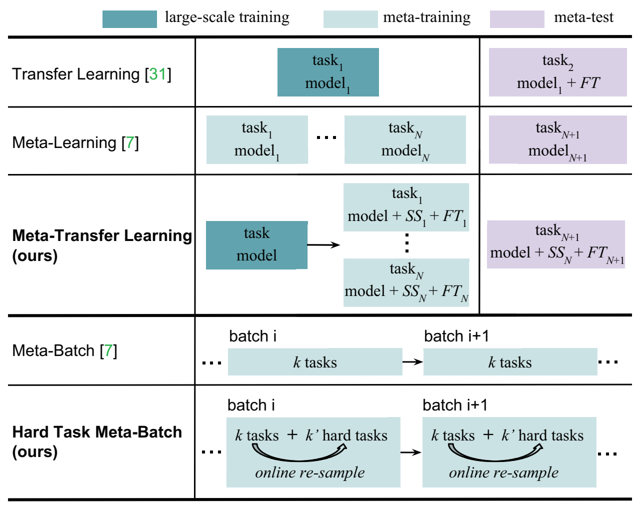
large-scale training
meta-training
meta-test
Transfer Learning [31]
task$_1$ model$_1$
task$_2$ model$_1$ + FT
Meta-Learning [7]
task$_1$ model$_1$
task$_N$ model$_N$
task$_{N+1}$ model$_{N+1}$
Meta-Transfer Learning (ours)
task model
task$_1$ model + SS_1 + FT_1
task$_N$ model + SS_N + FT_N
task$_{N+1}$ model + SS_N + FT_{N+1}
Meta-Batch [7]
batch i
batch i+1
k tasks
Hard Task Meta-Batch (ours)
k tasks + k' hard tasks
online re-sample

Figure 1. Meta-transfer learning (MTL) is our meta-learning paradigm and hard task (HT) meta-batch is our training strategy. The upper three rows show the differences between MTL and related methods, transfer-learning [31] and meta-learning [7]. The bottom rows compare HT meta-batch with the conventional meta-batch [7]. *FT* stands for fine-tuning a classifier. *SS* represents the *Scaling* and *Shifting* operations in our MTL method.

context, often grasping the essential connection between new concepts and their own knowledge and experience, it remains challenging for machine learning approaches. E.g., on the CIFAR-100 dataset, a state-of-the-art method [30] achieves only 40.1% accuracy for 1-shot image classification, compared to 75.7% for the all-class fully supervised case [5].

Few-shot learning methods can be roughly categorized into two classes: data augmentation methods and task based meta-learning. Data augmentation is a classic technique to increase the amount of available data and thus also useful for few-shot learning [18]. Several methods propose to learn a data generator *e.g.* conditioned on Gaussian noise [25, 38, 46]. However, the data generation models often under-perform when trained on few-shot data [1]. An

*This work was done during his internship at NUS.
[1]Code and supplementary materials will be made public soon.

alternative is to merge data from multiple tasks which, however, is often not effective due to variances of the data across tasks [46].

In contrast to data-augmentation methods, meta-learning is a task-level learning method [2, 29, 44]. Meta-learning aims to accumulate experience from learning multiple tasks [7, 34, 42, 27, 11], while base-learning focuses on modeling the data distribution of a single task. A state-of-the-art representative of this, namely Model-Agnostic Meta-Learning (MAML), learns to search for the optimal initialization state to fast adapt a base-learner to a new task [7]. Its task-agnostic property makes it possible to generalize to few-shot supervised learning as well as unsupervised reinforcement learning [11, 8]. However, in our view, there are two main limitations of this type of approaches limiting their effectiveness: i) these methods usually require a large number of similar tasks for meta-training which is costly; and ii) each task is typically modeled by a low-complexity base learner (such as a shallow neural network) to avoid model overfitting, thus being unable to use deeper and more powerful architectures. For example, for the miniImageNet dataset [45], MAML uses a *shallow* CNN with only 4 CONV layers and its optimal performance was obtained learning on $240k$ tasks ($60k$ iterations in total and each meta-batch contains 4 tasks).

In this paper, we propose a novel meta-learning method called **meta-transfer learning (MTL)** leveraging the advantages of both transfer and meta learning (see conceptual comparison of related methods in Figure 1). In a nutshell, MTL is a novel learning method that helps deep neural nets converge faster while reducing the probability to overfit when using few labeled training data only. In particular, "transfer" means that DNN weights trained on large-scale data can be used in other tasks by two light-weight neuron operations: *Scaling* and *Shifting* (*SS*), i.e. $\alpha X + \beta$. "Meta" means that the parameters of these operations can be viewed as hyper-parameters trained on few-shot learning tasks [27, 22]. Large-scale trained DNN weights offer a good initialization, enabling fast convergence of meta-transfer learning with fewer tasks, e.g. only $8k$ tasks for miniImageNet [45], 30 times fewer than MAML [7]. Light-weight operations on DNN neurons have less parameters to learn, e.g. less than $\frac{2}{49}$ if considering neurons of size 7×7 ($\frac{1}{49}$ for α and $< \frac{1}{49}$ for β), reducing the chance of overfitting. In addition, these operations keep those trained DNN weights unchanged, and thus avoid the problem of "catastrophic forgetting" which means forgetting general patterns when adapting to a specific task [23, 24].

The second main contribution of this paper is an effective meta-training curriculum. Curriculum learning [3] and hard negative mining [41] both suggest that faster convergence and stronger performance can be achieved by better arrangement of training data. Inspired by these ideas, we design our **hard task (HT) meta-batch** strategy to offer a challenging but effective learning curriculum. As shown in the bottom rows of Figure 1, a conventional meta-batch contains a number of random tasks [7], but our HT meta-batch online re-samples harder ones according to past failure tasks with lowest validation accuracy.

Our overall contribution is thus three-fold: i) we propose a novel **MTL** method that learns to transfer large-scale pre-trained DNN weights for solving few-shot learning tasks; ii) we propose a novel **HT meta-batch** learning strategy that forces meta-transfer to "grow faster and stronger through hardship"; and iii) we conduct extensive experiments on two few-shot learning benchmarks, namely miniImageNet [45] and Fewshot-CIFAR100 (FC100) [30], and achieve the state-of-the-art performance.

2. Related work

Few-shot learning Research literature on few-shot learning exhibits great diversity. In this section, we focus on methods using the supervised meta-learning paradigm [10, 44, 7] most relevant to ours and compared to in the experiments. We can divide these methods into three categories. 1) *Metric learning* methods [45, 42, 43] learn a similarity space in which learning is particularly efficient for few-shot examples. 2) *Memory network* methods [27, 36, 30, 26] learn to store "experience" when learning seen tasks and then generalize that to unseen tasks. 3) *Gradient descent* based methods [7, 34, 20, 11, 52] have a specific *meta-learner* that learns to adapt a specific *base-learner* (to few-shot examples) through different tasks. E.g. MAML [7] uses a meta-learner that learns to effectively initialize a base-learner for a new learning task. Meta-learner optimization is done by gradient descent using the validation loss of the base-learner. Our method is closely related. An important difference is that our MTL approach leverages transfer learning and benefits from referencing neuron knowledge in pre-trained deep nets. Although MAML can start from a pre-trained network, its element-wise fine-tuning makes it hard to learn deep nets without overfitting (validated in our experiments).

Transfer learning *What* and *how* to transfer are key issues to be addressed in transfer learning, as different methods are applied to different source-target domains and bridge different transfer knowledge [31, 49, 47, 51]. For deep models, a powerful transfer method is adapting a pre-trained model for a new task, often called *fine-tuning* (*FT*). Models pre-trained on large-scale datasets have proven to generalize better than randomly initialized ones [6]. Another popular transfer method is taking pre-trained networks as backbone and adding high-level functions, e.g. for object detection [15] and image segmentation [13, 4]. Our meta-transfer learning leverages the idea of transferring pre-trained weights and aims to meta-learn how to effectively

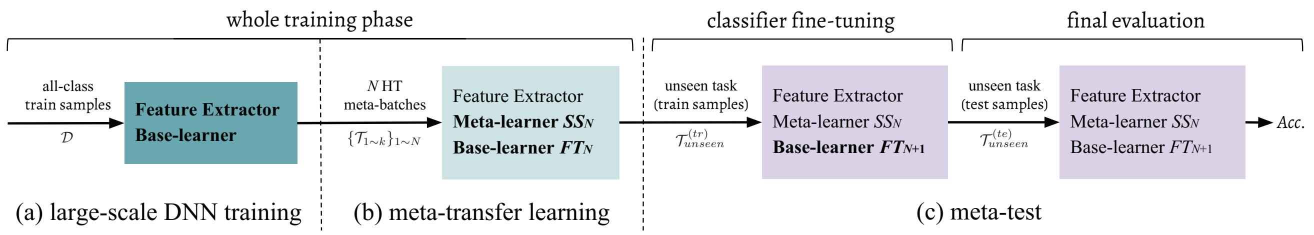

Figure 2. The pipeline of our proposed few-shot learning method, including three phases: (a) DNN training on large-scale data, i.e. using all training datapoints (Section 4.1); (b) Meta-transfer learning (MTL) that learns the parameters of *Scaling* and *Shifting* (*SS*), based on the pre-trained feature extractor (Section 4.2). Learning is scheduled by the proposed HT meta-batch (Section 4.3); and (c) meta-test is done for an unseen task which consists of a base-learner (classifier) *Fine-Tuning* (*FT*) stage and a final evaluation stage, described in the last paragraph in Section 3. Input data are along with arrows. Modules with names in bold get updated at corresponding phases.

transfer. In this paper, large-scale trained DNN weights are *what* to transfer, and the operations of *Scaling* and *Shifting* indicate *how* to transfer.

Some few-shot learning methods have been proposed to transfer pre-trained weights for fast adaptation [17, 26, 33, 39]. Typically, weights are fine-tuned for each task, while we learn a meta-transfer learner through all tasks, which is different in terms of the underlying learning paradigm.

Curriculum learning & Hard sample mining Curriculum learning was proposed by Bengio *et al*. [3] and is popular for multi-task learning [32, 37, 48, 12]. They showed that instead of observing samples at random it is better to organize samples in a meaningful way so that fast convergence, effective learning and better generalization can be achieved. Pentina *et al*. [32] use adaptive SVM classifiers to evaluate task difficulty for later organization. Differently, our MTL method does task evaluation online at the phase of episode test, without needing any auxiliary model.

Hard sample mining was proposed by Shrivastava *et al*. [41] for object detection. It treats image proposals overlapped with ground truth as hard negative samples. Training on more confusing data enables the model to achieve higher robustness and better performance. Inspired by this, we sample harder tasks online and make our MTL learner "grow faster and stronger through more hardness". In our experiments, we show that this can be generalized to enhance other meta-learning methods, e.g. MAML [7].

3. Preliminary

This section introduces and defines the meta-learning notations and the problem setup, following related work [45, 34, 7, 30].

Meta-learning consists of two phases: meta-train and meta-test. A meta-training example is a classification task $\mathcal{T}$ sampled from a distribution $p(\mathcal{T})$. $\mathcal{T}$ is called episode, including a training split $\mathcal{T}^{(tr)}$ to optimize the base-learner, and a test split $\mathcal{T}^{(te)}$ to optimize the meta-learner. In particular, meta-training aims to learn from a number of episodes $\{\mathcal{T}\}$ sampled from $p(\mathcal{T})$. An unseen task $\mathcal{T}_{unseen}$ in meta-test will start from that experience of the meta-learner and adapt the base-learner. The final evaluation is done by testing a set of unseen datapoints $\mathcal{T}^{(te)}_{unseen}$.

Meta-training phase. This phase aims to learn a meta-learner from multiple episodes. In each episode, meta-training has a two-stage optimization. Stage-1 is called base-learning, where the cross-entropy loss is used to optimize the parameters of the base-learner. Stage-2 contains a feed-forward test on episode test datapoints. The test loss is used to optimize the parameters of the meta-learner. Specifically, given an episode $\mathcal{T} \in p(\mathcal{T})$, the base-learner $\theta_\mathcal{T}$ is learned from episode training data $\mathcal{T}^{(tr)}$ and its corresponding loss $\mathcal{L}_\mathcal{T}(\theta_\mathcal{T}, \mathcal{T}^{(tr)})$. After optimizing this loss, the base-learner has parameters $\tilde{\theta}_\mathcal{T}$. Then, the meta-learner is updated using test loss $\mathcal{L}_\mathcal{T}(\tilde{\theta}_\mathcal{T}, \mathcal{T}^{(te)})$. After meta-training on all episodes, the meta-learner is optimized by test losses $\{\mathcal{L}_\mathcal{T}(\tilde{\theta}_\mathcal{T}, \mathcal{T}^{(te)})\}_{\mathcal{T}\in p(\mathcal{T})}$. Therefore, the number of meta-learner updates equals to the number of episodes.

Meta-test phase. This phase aims to test the performance of the trained meta-learner for fast adaptation to unseen task. Given $\mathcal{T}_{unseen}$, the meta-learner $\tilde{\theta}_\mathcal{T}$ teaches the base-learner $\theta_{\mathcal{T}_{unseen}}$ to adapt to the objective of $\mathcal{T}_{unseen}$ by some means, e.g. through initialization [7]. Then, the test result on $\mathcal{T}^{(te)}_{unseen}$ is used to evaluate the meta-learning approach. If there are multiple unseen tasks $\{\mathcal{T}_{unseen}\}$, the average result on $\{\mathcal{T}^{(te)}_{unseen}\}$ will be the final evaluation.

4. Methodology

As shown in Figure 2, our method consists of three phases. First, we train a DNN on large-scale data, e.g. on miniImageNet (64-class, 600-shot) [45], and then fix the low-level layers as Feature Extractor (Section 4.1). Second, in the meta-transfer learning phase, MTL learns the *Scaling* and *Shifting* (*SS*) parameters for the Feature Extractor neurons, enabling fast adaptation to few-shot tasks (Section 4.2). For improved overall learning, we use our HT meta-batch strategy (Section 4.3). The training steps are detailed in Algorithm 1 in Section 4.4. Finally, the typical meta-test phase is performed, as introduced in Section 3.

4.1. DNN training on large-scale data

This phase is similar to the classic pre-training stage as, e.g., pre-training on Imagenet for object recognition [35]. Here, we do not consider data/domain adaptation from other datasets, and pre-train on readily available data of few-shot learning benchmarks, allowing for fair comparison with other few-shot learning methods. Specifically, for a particular few-shot dataset, we merge all-class data $\mathcal{D}$ for pre-training. For instance, for miniImageNet [45], there are totally 64 classes in the training split $\mathcal{D}$ and each class contains 600 samples, which we use to pre-train a 64-class classifier.

We first randomly initialize a feature extractor Θ (e.g. CONV layers in ResNets [14]) and a classifier θ (e.g. the last FC layer in ResNets [14]), and then optimize them by gradient descent as follows,

$$[\Theta;\theta] =: [\Theta;\theta] - \alpha\nabla\mathcal{L}_{\mathcal{D}}([\Theta;\theta]) \tag{1}$$

where $\mathcal{L}$ denotes the following empirical loss,

$$\mathcal{L}_{\mathcal{D}}([\Theta;\theta]) = \frac{1}{|\mathcal{D}|}\sum_{(x,y)\in\mathcal{D}} l(f_{[\Theta;\theta]}(x),y), \tag{2}$$

e.g. cross-entropy loss, and α denotes the learning rate. In this phase, the feature extractor Θ is learned. It will be frozen in the following meta-training and meta-test phases, as shown in Figure 2. The learned classifier θ will be discarded, because subsequent few-shot tasks contain different classification objectives, e.g. 5-class instead of 64-class classification for miniImageNet [45].

4.2. Meta-transfer learning (MTL)

As shown in Figure 2(b), our proposed meta-transfer learning (MTL) method optimizes the meta operations *Scaling* and *Shifting* (*SS*) through HT meta-batch training (Section 4.3). Figure 3 visualizes the difference of updating through *SS* and *FT*. *SS* operations, denoted as Φ_{S_1} and Φ_{S_2}, do not change the frozen neuron weights of Θ during learning, while *FT* updates the complete Θ.

In the following, we detail the *SS* operations. Given a task $\mathcal{T}$, the loss of $\mathcal{T}^{(tr)}$ is used to optimize the current base-learner (classifier) θ' by gradient descent:

$$\theta' \leftarrow \theta - \beta\nabla_\theta\mathcal{L}_{\mathcal{T}^{(tr)}}([\Theta;\theta;\Phi_{S_{\{1,2\}}}]), \tag{3}$$

which is different to Eq. 1, as we do not update Θ. Note that here θ is different to the one from the previous phase, the large-scale classifier θ in Eq. 1. The new θ concerns only a few of classes, e.g. 5 classes in miniImageNet [45], to classify in a novel few-shot setting. θ' corresponds to a temporal classifier only working in the current task, initialized by the θ optimized for the previous task (see Eq. 5).

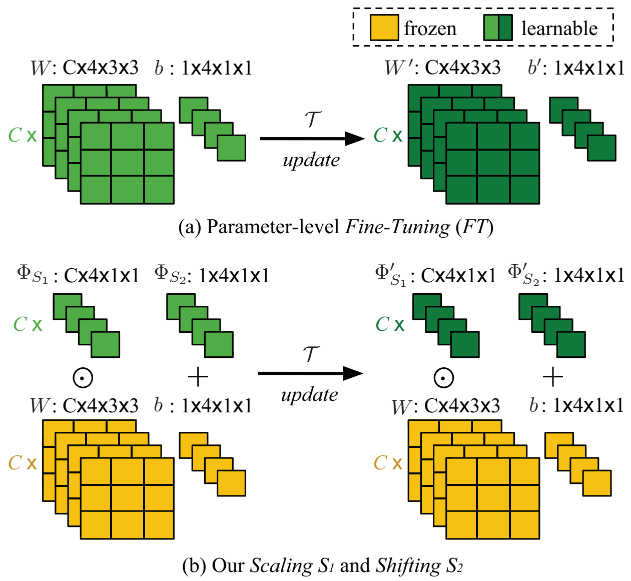
frozen
learnable
W: Cx4x3x3 b : 1x4x1x1
W': Cx4x3x3 b': 1x4x1x1
C x
$\mathcal{T}$
update
(a) Parameter-level *Fine-Tuning* (*FT*)
Φ_{S_1} : Cx4x1x1 Φ_{S_2}: 1x4x1x1
Φ'_{S_1} : Cx4x1x1 Φ'_{S_2}: 1x4x1x1
$\odot$
$+$
W: Cx4x3x3 b : 1x4x1x1
(b) Our *Scaling S₁* and *Shifting S₂*

Figure 3. (a) Parameter-level *Fine-Tuning* (*FT*) is a conventional meta-training operation, e.g. in MAML [7]. Its update works for all neuron parameters, W and b. (b) Our neuron-level *Scaling* and *Shifting* (*SS*) operations in MTL. They reduce the number of learning parameters and avoid overfitting problems. In addition, they keep large-scale trained parameters (in yellow) frozen, preventing "catastrophic fogetting" [23, 24].

Φ_{S_1} is initialized by ones and Φ_{S_1} by zeros. Then, they are optimized by the test loss of $\mathcal{T}^{(te)}$ as follows,

$$\Phi_{S_i} =: \Phi_{S_i} - \gamma\nabla_{\Phi_{S_i}}\mathcal{L}_{\mathcal{T}^{(te)}}([\Theta;\theta';\Phi_{S_{\{1,2\}}}]). \tag{4}$$

In this step, θ is updated with the same learning rate γ as in Eq. 4,

$$\theta =: \theta - \gamma\nabla_\theta\mathcal{L}_{\mathcal{T}^{(te)}}([\Theta;\theta';\Phi_{S_{\{1,2\}}}]). \tag{5}$$

Re-linking to Eq. 3, we note that the above θ' comes from the last epoch of base-learning on $\mathcal{T}^{(tr)}$.

Next, we describe how we apply $\Phi_{S_{\{1,2\}}}$ to the frozen neurons as shown in Figure 3(b). Given the trained Θ, for its l-th layer containing K neurons, we have K pairs of parameters, respectively as weight and bias, denoted as $\{(W_{i,k}, b_{i,k})\}$. Note that the neuron location l,k will be omitted for readibility. Based on MTL, we learn K pairs of scalars $\{\Phi_{S_{\{1,2\}}}\}$. Assuming X is input, we apply $\{\Phi_{S_{\{1,2\}}}\}$ to (W,b) as

$$SS(X;W,b;\Phi_{S_{\{1,2\}}}) = (W\odot\Phi_{S_1})X + (b+\Phi_{S_2}), \tag{6}$$

where $\odot$ denotes the element-wise multiplication.

Taking Figure 3(b) as an example of a single 3×3 filter, after *SS* operations, this filter is scaled by Φ_{S_1} then the feature maps after convolutions are shifted by Φ_{S_2} in addition to the original bias b. Detailed steps of *SS* are given in Algorithm 2 in Section 4.4.

Figure 3(a) shows a typical parameter-level *Fine-Tuning* (*FT*) operation, which is in the meta optimization phase of our related work MAML [7]. It is obvious that *FT* updates the complete values of W and b, and has a large number of parameters, and our *SS* reduces this number to below $\frac{2}{9}$ in the example of the figure.

In summary, *SS* can benefit MTL in three aspects. 1) It starts from a strong initialization based on a large-scale trained DNN, yielding fast convergence for MTL. 2) It does not change DNN weights, thereby avoiding the problem of "catastrophic forgetting" [23, 24] when learning specific tasks in MTL. 3) It is light-weight, reducing the chance of overfitting of MTL in few-shot scenarios.

4.3. Hard task (HT) meta-batch

In this section, we introduce a method to schedule hard tasks in meta-training batches. The conventional meta-batch is composed of randomly sampled tasks, where the randomness implies random difficulties [7]. In our meta-training pipeline, we intentionally pick up failure cases in each task and re-compose their data to be harder tasks for adverse re-training. We aim to force our meta-learner to "grow up through hardness".

Pipeline. Given a (M-class, N-shot) task $\mathcal{T}$, a meta-batch $\{\mathcal{T}_{1\sim k}\}$ contains two splits, $\mathcal{T}^{(tr)}$ and $\mathcal{T}^{(te)}$, for base-learning and test, respectively. As shown in Algorithm 2 line 2-5, base-learner is optimized by the loss of $\mathcal{T}^{(tr)}$ (in multiple epochs). *SS* parameters are then optimized by the loss of $\mathcal{T}^{(te)}$ once. During the loss computation on $\mathcal{T}^{(te)}$, we can also get the recognition accuracy for M classes. Then, we choose the lowest accuracy Acc_m to determine the most difficult class-m (also called failure class) in the current task.

After obtaining all failure classes $\{m\}$ from $\{\mathcal{T}_{1\sim k}\}$ in current meta-batch, we re-sample tasks from the data indexed by $\{m\}$. Specifically, we assume $p(\mathcal{T}|\{m\})$ is the task distribution, we sample a "harder" task $\mathcal{T}^{hard} \in p(\mathcal{T}|\{m\})$. Two important details are given below.

Choosing hard class-m. We choose the failure class-m from each task by ranking the class-level accuracies instead of fixing a threshold. In a dynamic online setting as ours it is more sensible to choose the hardest cases based on ranking rather than fixing a threshold ahead of time.

Two methods of hard tasking using $\{m\}$. Chosen $\{m\}$, we can re-sample tasks $\mathcal{T}^{hard}$ by (1) directly using the samples of class-m in the current task $\mathcal{T}$, or (2) indirectly using the label of class-m to sample new samples of that class. In fact, setting (2) considers to include more data variance of class-m and it works better than setting (1) in general.

4.4. Algorithm

Algorithm 1 summarizes the training process of two main stages: large-scale DNN training (line 1-5) and meta-transfer learning (line 6-22). HT meta-batch re-sampling and continuous training phases are shown in lines 16-20, for which the failure classes are returned by Algorithm 2, see line 14.

Algorithm 1: MTL with HT meta-batch strategy

```
Input: Task distribution $p(\mathcal{T})$ and corresponding
       dataset $\mathcal{D}$, learning rates $\alpha$, $\beta$ and $\gamma$
Output: Feature extractor $\Theta$, base learner $\theta$, Scaling
        and Shifting parameters $\Phi_{S_{\{1,2\}}}$
1  Randomly initialize $\Theta$ and $\theta$;
2  for samples in $\mathcal{D}$ do
3      Evaluate $\mathcal{L}_{\mathcal{D}}([\Theta;\theta])$ by Eq. 2;
4      Optimize $\Theta$ and $\theta$ by Eq. 1;
5  end
6  Initialize $\Phi_{S_1}$ by ones, initialize $\Phi_{S_2}$ by zeros;
7  Reset $\theta$ for few-shot tasks;
8  Randomly initialize $\theta$;
9  for meta-batches do
10     Randomly sample tasks $\{\mathcal{T}\} \subseteq p(\mathcal{T})$;
11     while not done do
12         Sample task $\mathcal{T}_i \in \{\mathcal{T}\}$;
13         Optimize $\Phi_{S_{\{1,2\}}}$ and $\theta$ with $\mathcal{T}_i$ by
            Algorithm 2;
14         Get the returned class-$m$ then add it to a hard
            class set $\{m\}$;
15     end
16     Sample hard tasks $\{\mathcal{T}^{hard}\} \subseteq p(\mathcal{T}|\{m\})$;
17     while not done do
18         Sample task $\mathcal{T}_j^{hard} \in \{\mathcal{T}^{hard}\}$ ;
19         Optimize $\Phi_{S_{\{1,2\}}}$ and $\theta$ with $\mathcal{T}_j^{hard}$ by
            Algorithm 2 ;
20     end
21     Empty $\{m\}$.
22 end
```

Algorithm 2 presents an entire learning epoch on a single task. Base-learning in the episode training split are given in lines 2-5. The meta-level update by the test loss is shown on line 6. In lines 7-11, the recognition rates of all test classes are computed and returned to Algorithm 1 (line 14) for hard task sampling.

5. Experiments

We evaluate the proposed **MTL** and **HT meta-batch** in terms of few-shot recognition accuracy and model convergence speed. Below we describe the datasets we evaluate on and detailed settings, followed by an ablation study and a comparison to state-of-the-art methods.

Algorithm 2: Detail learning steps within a task $\mathcal{T}$

```
Input: Task $\mathcal{T}$, learning rates $\beta$ and $\gamma$, feature
       extractor $\Theta$, base learner $\theta$, Scaling and
       Shifting parameters $\Phi_{S_{\{1,2\}}}$
Output: Base learner $\theta$, Scaling and Shifting
        parameters $\Phi_{S_{\{1,2\}}}$, the worst classified
        class-$m$ in $\mathcal{T}$
1  Sample training datapoints $\mathcal{T}^{(tr)}$ and test datapoints
     $\mathcal{T}^{(te)}$ from $\mathcal{T}$ ;
2  for samples in $\mathcal{T}^{(tr)}$ do
3      Evaluate $\mathcal{L}_{\mathcal{T}^{(tr)}}$;
4      Optimize $\theta'$ by Eq. 3;
5  end
6  Optimize $\Phi_{S_{\{1,2\}}}$ and $\theta$ by Eq. 4 and Eq. 5 (using
     $\mathcal{T}^{(te)}$) ;
7  while not done do
8      Sample class-$k$ in $\mathcal{T}^{(te)}$;
9      Compute $Acc_k$ for $\mathcal{T}^{(te)}$;
10 end
11 Return class-$m$ with the lowest accuracy $Acc_m$.
```

5.1. Datasets and implementation details

We conduct few-shot learning experiments on two benchmarks, miniImageNet [45] and Fewshot-CIFAR100 (FC100) [30]. The former is widely used in related works [7, 34, 11, 9, 28], and the latter is a newly proposed one which is more challenging in terms of low image resolution and strict training-test splits [30].

miniImageNet. It was proposed by Vinyals *et al.* [45] for few-shot learning evaluation. Its complexity is high due to the use of ImageNet images, but requires less resources and infrastructure than running on the full ImageNet dataset [35]. In total, there are 100 classes with 600 samples of 84×84 color images per class. These 100 classes are divided into 64, 16, and 20 classes respectively for sampling tasks for meta-training, meta-validation and meta-test, following related works [7, 34, 11, 9, 28].

Fewshot-CIFAR100 (FC100). It is based on the popular object classification dataset CIFAR100 [19]. The splits were proposed by [30], see details in the supplementary. It offers a more challenging scenario with lower image resolution and more challenging meta-training/test splits (separated according to object super-classes) than miniImageNet. It contains 100 object classes and each class has 600 samples of 32×32 color images. The 100 classes belong to 20 super-classes. Meta-training data are from 60 classes belonging to 12 super-classes. Meta-validation and meta-test sets contain 20 classes belonging to 4 super-classes, respectively. These splits according to super-classes minimize the information overlap between training and test tasks.

The following settings are used for both datasets. We train a large-scale DNN with all training datapoints (Section 4.1) and stop this training after $10k$ iterations. We use the same task sampling method as related works [7, 34]. Specifically, 1) we consider the 5-class classification and 2) we sample 5-class, 1-shot (5-shot or 10-shot) episodes to contain 1 (5 or 10) samples for train episode, and 15 (uniform) samples for episode test. Note that in the state-of-the-art work [30], 32 and 64 samples are respectively used in 5-shot and 10-shot settings for episode test. In total, we sample $8k$ tasks for meta-training (same for w/ or w/o HT meta-batch), and respectively sample 600 random tasks for meta-validation and meta-test. We present other details, such as learning rate and dropout hyperparameters, in the supplementary.

Network architecture. We present the details for the Feature Extractor Θ, MTL meta-learner with *Scaling* Φ_{S_1} and *Shifting* Φ_{S_2}, and MTL base-learner (classifier) θ.

The architecture of Θ have two options, ResNet-12 and 4CONV, commonly used in related works [7, 45, 34, 28, 26, 30]. **4CONV** consists of 4 layers with 3×3 convolutions and 32 filters, followed by batch normalization (BN) [16], a ReLU nonlinearity, and 2×2 max-pooling. **ResNet-12** is more popular in recent works [30, 26, 9, 28]. It contains 4 residual blocks and each block has 3 CONV layers with 3×3 kernels. At the end of each residual block, a 2×2 max-pooling layer is applied. The number of filters starts from 64 and is doubled every next block. Following 4 blocks, there is a mean-pooling layer to compress the output feature maps to a feature embedding. **The difference** between using 4CONV and using ResNet-12 in our methods is that ResNet-12 MTL sees the large-scale data training, but 4CONV MTL is learned from scratch because of its poor performance for large-scale data training (see results in the supplementary). Therefore, we emphasize the experiments of using ResNet-12 MTL for its superior performance.

The architectures of Φ_{S_1} **and** Φ_{S_2} are generated according to the architecture of Θ, as introduced in Section 4.2. That is when using ResNet-12 in MTL, Φ_{S_1} and Φ_{S_2} also have 12 layers, respectively.

The architecture of θ is a FC layer. We empirically find that a single FC layer is faster to train and more effective for classification than multiple layers. We give the multi-layer results in the supplementary.

5.2. Ablation study setting

In order to show the effectiveness of our MTL, we design six ablative settings: two baselines without meta-learning but more classic learning, and four MTL methods each time disabling one of the following components: ResNet-12 (ResNet-12 pre-trained on large-scale data), *SS* Θ and HT meta-batch. Note that the alternative meta-learning operation to *SS* is *Fine-Tuning* (*FT*) [7], see Figure 3. Abla-

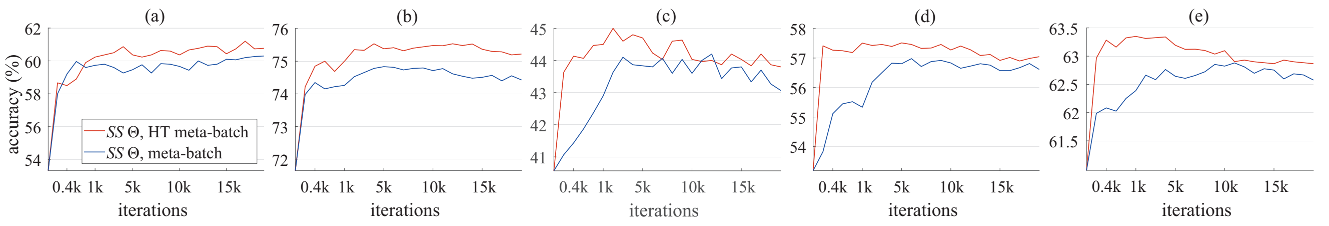

Figure 4. (a)(b) show the results of 1-shot, 5-shot on miniImageNet; (c)(d)(e) show the results of 1-shot, 5-shot and 10-shot on FC100. Note that each meta-training iteration contains 2 tasks.

tive methods are given in the bottom rows of Table 1 and Table 2. Some bullet names are explained as follows.

No meta-learning, *update* $[\Theta;\theta]$. There is no meta-training phase. During test phase, each task has its whole model $[\Theta;\theta]$ updated on $\mathcal{T}^{(tr)}$, and then tests on $\mathcal{T}^{(te)}$.

No meta-learning, *update* θ. This is similar to the above method. During test phase, each task has its classifier layer θ updated. Θ works as a frozen feature extractor.

***FT* Θ, HT meta-batch, 4CONV.** In terms of the model, this is exactly the same as MAML [7]. For the learning strategy, it uses our proposed HT meta-batch. This setting aims to validate the generalization ability of HT meta-batch.

5.3. Results and Analysis

Table 1 and Table 2 present the overall results on miniImageNet and FC100 datasets. Extensive comparisons are done with ablative methods and the state-of-the-arts. Figure 4 shows the performance gap between *with* and *without* HT meta-batch in terms of recognition accuracy and converging speed.

Result overview on miniImageNet. In Table 1, we can see that the proposed MTL with *SS* Θ, HT meta-batch and ResNet-12(pre) achieves the best few-shot classification performance with 61.2% for (5-class, 1-shot). Besides, it tackles the (5-class, 5-shot) tasks with an accuracy of 75.5% that is comparable to the state-of-the-art results, *i.e.*, 76.7%, reported by TADAM [30] whose model used 72 additional FC layers in the ResNet-12 arch. In terms of the network arch, it is obvious that models using ResNet-12 (pre) outperforms those using 4CONV by large margins, *e.g.*, 4CONV models have the best 1-shot result with 50.44% [43] which is 10.8% lower than our best.

Result overview on FC100. In Table 2, we give the results of TADAM using their reported numbers in the paper [30]. We used the public code of MAML [7] to get its results for this new dataset. Comparing these methods, we can see that MTL consistently outperforms MAML by large margins, *i.e.*, around 7% in all tasks; and surpasses TADAM by a relatively larger number of 5% for 1-shot, and with 1.5% and 1.8% respectively for 5-shot and 10-shot tasks.

MTL *vs.* No meta-learning. MTL consistently achieves better performance than methods of **No meta-learning**, *e.g.*, the largest margins are 11.2% for 1-shot and 9.8% for 5-shot on miniImageNet. This validates the effectiveness of the meta-learning paradigm for tackling few-shot learning problems. Between two **No meta-learning** methods, we can see that updating both feature extractor Θ and classifier θ is inferior to updating Θ only, *e.g.*, around 5% reduction on miniImageNet 1-shot. This supports our motivation to learn θ but not $[\Theta;\theta]$ during base-learning (we present more detailed comparisons for this choice in the supplementary). One reason is that in few-shot settings, that there are too many parameters to optimize with too little data.

Performance effects of MTL components. MTL with full components, *SS* Θ, HT meta-batch and ResNet-12(pre), achieves the best performances for all few-shot settings on both datasets, comparing with other ablative methods in bottom rows of Table 1 and Table 2. Using ablative methods, the largest margin between two adjacent rows is 9.2% for 1-shot and 7.5% for 5-shot on miniImageNet, *i.e.*, between using 4CONV and using ResNet-12(pre). We can conclude that our large-scale network training on deep CNN significantly boost the few-shot learning performance. This is an important gain brought by the transfer learning idea in our MTL approach. It is interesting to note that this gain on FC100 is not as large as for miniImageNet: only 1.7%, 1.0% and 4.0%. The possible reason is that FC100 tasks for meta-train and meta-test are clearly split according to super-classes. The data domain gap is larger than for miniImageNet, which makes transfer learning more difficult.

For another fair comparison with MAML [7], we have a setting denoted as "*FT* Θ, meta-batch, ResNet-12(pre)" which is actually MAML with ResNet-12(pre). We can see that our best MTL results are consistently higher: *e.g.*, 2.9% and 3.9% on miniImageNet.

SS surpasses *FT* by an average improvement of 2% on both miniImageNet and FC100. When including HT meta-batch, we obtain additional 1.1% and 1.7% improvements for the datasets, respectively. HT meta-batch makes a larger contribution to the more challenging dataset – FC100. In another aspect, the strong generalization ability of HT meta-batch is shown by comparing "*FT* Θ, HT meta-batch, 4CONV" with MAML [7], for which we can see that a consistent gain of 1% is achieved by using HT meta-batch.

Few-shot learning method		Feature extractor	1-shot	5-shot
Data augmentation	Adv. ResNet, [25]	WRN-40 (pre)	55.2	69.6
	Delta-encoder, [38]	VGG-16 (pre)	58.7	73.6
Metric learning	Matching Nets, [45]	4 CONV	43.44 ± 0.77	55.31 ± 0.73
	ProtoNets, [42]	4 CONV	49.42 ± 0.78	68.20 ± 0.66
	CompareNets, [43]	4 CONV	50.44 ± 0.82	65.32 ± 0.70
Memory network	Meta Networks, [27]	5 CONV	49.21 ± 0.96	–
	SNAIL, [26]	ResNet-12 (pre)$^\diamond$	55.71 ± 0.99	68.88 ± 0.92
	TADAM, [30]	ResNet-12 (pre)†	58.5 ± 0.3	$\mathbf{76.7 \pm 0.3}$
Gradient descent	MAML, [7]	4 CONV	48.70 ± 1.75	63.11 ± 0.92
	Meta-LSTM, [34]	4 CONV	43.56 ± 0.84	60.60 ± 0.71
	Hierarchical Bayes, [11]	4 CONV	49.40 ± 1.83	–
	Bilevel Programming, [9]	ResNet-12$^\diamond$	50.54 ± 0.85	64.53 ± 0.68
	MetaGAN, [52]	ResNet-12	52.71 ± 0.64	68.63 ± 0.67
	adaResNet, [28]	ResNet-12‡	56.88 ± 0.62	71.94 ± 0.57
No meta-learning	*update* $[\Theta; \theta]$	ResNet-12 (pre)	45.3 ± 1.8	64.6 ± 0.9
	update θ	ResNet-12 (pre)	50.0 ± 1.8	66.7 ± 0.9
MTL (Ours)	*FT* Θ, HT meta-batch	4 CONV	49.1 ± 1.9	64.1 ± 0.9
	FT Θ, meta-batch	ResNet-12 (pre)	58.3 ± 1.9	71.6 ± 0.9
	FT Θ, HT meta-batch	ResNet-12 (pre)	59.1 ± 1.9	73.1 ± 0.9
	SS Θ, meta-batch	ResNet-12 (pre)	60.2 ± 1.8	74.3 ± 0.9
	SS Θ, HT meta-batch	ResNet-12 (pre)	$\mathbf{61.2 \pm 1.8}$	$\mathbf{75.5 \pm 0.8}$

$^\diamond$Additional 2 convolutional layers ‡Additional 1 convolutional layer †Additional 72 fully connected layers

Table 1. The 5-way, 1-shot and 5-shot classification accuracy (%) on miniImageNet dataset. "pre" means pre-trained for a single classification task using all training datapoints.

Few-shot learning method		Feature extractor	1-shot	5-shot	10-shot
Gradient descent	MAML, [7]‡	4 CONV	38.1 ± 1.7	50.4 ± 1.0	56.2 ± 0.8
Memory network	TADAM, [30]	ResNet-12 (pre)†	40.1 ± 0.4	56.1 ± 0.4	61.6 ± 0.5
No meta-learning	*update* $[\Theta; \theta]$	ResNet-12 (pre)	38.4 ± 1.8	52.6 ± 1.0	58.6 ± 0.8
	update θ	ResNet-12 (pre)	39.3 ± 1.8	51.8 ± 1.0	61.0 ± 0.8
MTL (Ours)	*FT* Θ, HT meta-batch	4 CONV	39.9 ± 1.8	51.7 ± 0.9	57.2 ± 0.8
	FT Θ, meta-batch	ResNet-12 (pre)	41.6 ± 1.9	54.4 ± 0.9	61.2 ± 0.8
	FT Θ, HT meta-batch	ResNet-12 (pre)	41.8 ± 1.9	55.1 ± 0.9	61.9 ± 0.8
	SS Θ, meta-batch	ResNet-12 (pre)	43.6 ± 1.8	55.4 ± 0.9	62.9 ± 0.8
	SS Θ, HT meta-batch	ResNet-12 (pre)	$\mathbf{45.1 \pm 1.8}$	$\mathbf{57.6 \pm 0.9}$	$\mathbf{63.4 \pm 0.8}$

†Additional 72 fully connected layers ‡Our implementation using the public code of MAML.

Table 2. The 5-way with 1-shot, 5-shot and 10-shot classification accuracy (%) on Fewshot-CIFAR100 (FC100) dataset. "pre" means pre-trained for a single classification task using all training datapoints.

Speed of convergence of MTL. MAML [7] used $240k$ tasks to achieve the best performance on miniImageNet. Impressively, our MTL methods used only $8k$ tasks, see Figure 4(a)(b) (note that each iteration contains 2 tasks). This advantage is more obvious for FC100 on which MTL methods need at most $2k$ tasks, Figure 4(c)(d)(e). We attest this to two reasons. First, MTL starts from the pre-trained ResNet-12. And second, *SS* (in MTL) needs to learn only $< \frac{2}{9}$ parameters of the number of *FT* (in MAML) when using ResNet-12.

Speed of convergence of HT meta-batch. Figure 4 shows 1) MTL with HT meta-batch consistently achieves higher performances than MTL with the conventional meta-batch [7], in terms of the recognition accuracy in all settings; and 2) it is impressive that MTL with HT meta-batch achieves top performs early, after *e.g.* about $2k$ iterations for 1-shot, $1k$ for 5-shot and $1k$ for 10-shot, on the more challenging dataset – FC100.

6. Conclusions

In this paper, we show that our novel MTL trained with HT meta-batch learning curriculum achieves the top performance for tackling few-shot learning problems. The key operations of MTL on pre-trained DNN neurons proved highly efficient for adapting learning experience to the unseen task. The superiority was particularly achieved in the extreme 1-shot cases on two challenging benchmarks – miniImageNet and FC100. In terms of learning scheme, HT meta-batch showed consistently good performance for all baselines and ablative models. On the more challenging FC100 benchmark, it showed to be particularly helpful for boosting convergence speed. This design is independent from any specific model and could be generalized well whenever the hardness of task is easy to evaluate in online iterations.

Acknowledgments

This research is part of NExT research which is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative. It is partially supported by German Research Foundation (DFG CRC 1223).

References

[1] S. Bartunov and D. P. Vetrov. Few-shot generative modelling with generative matching networks. In *AISTATS*, pages 670–678, 2018. 1

[2] S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992. 2

[3] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, pages 41–48, 2009. 2, 3

[4] L. 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 Trans. Pattern Anal. Mach. Intell.*, 40(4):834–848, 2018. 2

[5] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016. 1

[6] D. Erhan, Y. Bengio, A. C. Courville, P. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010. 2

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017. 1, 2, 3, 4, 5, 6, 7, 8

[8] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. In *NIPS*, 2018. 2

[9] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, pages 1563–1572, 2018. 6, 8

[10] H. E. Geoffrey and P. C. David. Using fast weights to deblur old memories. In *CogSci*, pages 177–186, 1987. 2

[11] E. Grant, C. Finn, S. Levine, T. Darrell, and T. L. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *ICLR*, 2018. 2, 6, 8

[12] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks. In *ICML*, pages 1311–1320, 2017. 3

[13] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, 2017. 2

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 4

[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy. Speed/accuracy trade-offs for modern convolutional object detectors. In *CVPR*, pages 3296–3297, 2017. 2

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 6

[17] R. Keshari, M. Vatsa, R. Singh, and A. Noore. Learning structure and strength of CNN filters for small sample size training. In *CVPR*, 2018. 3

[18] A. Khoreva, R. Benenson, E. Ilg, T. Brox, and B. Schiele. Lucid data dreaming for object tracking. *arXiv*, 1703.09554, 2017. 1

[19] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 2009. 6

[20] Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, pages 2933–2942, 2018. 2

[21] F. Li, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(4):594–611, 2006. 1

[22] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few shot learning. In *ICML*, 2018. 2

[23] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *NIPS*, pages 6470–6479, 2017. 2, 4, 5

[24] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, pages 3–17, 1989. 2, 4, 5

[25] A. Mehrotra and A. Dukkipati. Generative adversarial residual pairwise networks for one shot learning. *arXiv*, 1703.08033, 2017. 1, 8

[26] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. Snail: A simple neural attentive meta-learner. In *ICLR*, 2018. 2, 3, 6, 8

[27] T. Munkhdalai and H. Yu. Meta networks. In *ICML*, 2017. 2, 8

[28] T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler. Rapid adaptation with conditionally shifted neurons. In *ICML*, pages 3661–3670, 2018. 6, 8

[29] D. K. Naik and R. Mammone. Meta-neural networks that learn by learning. In *IJCNN*, pages 437–442, 1992. 2

[30] B. N. Oreshkin, P. Rodríguez, and A. Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *NIPS*, 2018. 1, 2, 3, 6, 7, 8

[31] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Trans. Neural Networks*, 22(2):199–210, 2011. 1, 2

[32] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *CVPR*, pages 5492–5500, 2015. 3

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

[34] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 2, 3, 6, 8

[35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 4, 6

[36] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016. 2

[37] N. Sarafianos, T. Giannakopoulos, C. Nikou, and I. A. Kakadiaris. Curriculum learning for multi-task classification of visual attributes. In *ICCV Workshops*, pages 2608–2615, 2017. 3

[38] E. Schwartz, L. Karlinsky, J. Shtok, S. Harary, M. Marder, R. S. Feris, A. Kumar, R. Giryes, and A. M. Bronstein. Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *NIPS*, 2018. 1, 8

[39] T. R. Scott, K. Ridgeway, and M. C. Mozer. Adapted deep embeddings: A synthesis of methods for k-shot inductive transfer learning. In *NIPS*, 2018. 3

[40] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, 2017. 1

[41] A. Shrivastava, A. Gupta, and R. B. Girshick. Training region-based object detectors with online hard example mining. In *CVPR*, pages 761–769, 2016. 2, 3

[42] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4080–4090, 2017. 2, 8

[43] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018. 2, 7, 8

[44] S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998. 2

[45] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *NIPS*, pages 3630–3638, 2016. 2, 3, 4, 6, 8

[46] Y. Wang, R. B. Girshick, M. Hebert, and B. Hariharan. Low-shot learning from imaginary data. In *CVPR*, 2018. 1, 2

[47] Y. Wei, Y. Zhang, J. Huang, and Q. Yang. Transfer learning via learning to transfer. In *ICML*, volume 80, pages 5085–5094, 2018. 2

[48] D. Weinshall, G. Cohen, and D. Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *ICML*, pages 5235–5243, 2018. 3

[49] J. Yang, R. Yan, and A. G. Hauptmann. Adapting SVM classifiers to data with shifted distributions. In *ICDM Workshops*, pages 69–76, 2007. 2

[50] L. Yann, B. Yoshua, and H. Geoffrey. Deep learning. *Nature*, 521(7553):436, 2015. 1

[51] A. R. Zamir, A. Sax, W. B. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. 2018. 2

[52] R. Zhang, T. Che, Z. Grahahramani, Y. Bengio, and Y. Song. Metagan: An adversarial approach to few-shot learning. In *NIPS*, 2018. 2, 8