LCC: Learning to Customize and Combine Neural Networks for Few-Shot Learning

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

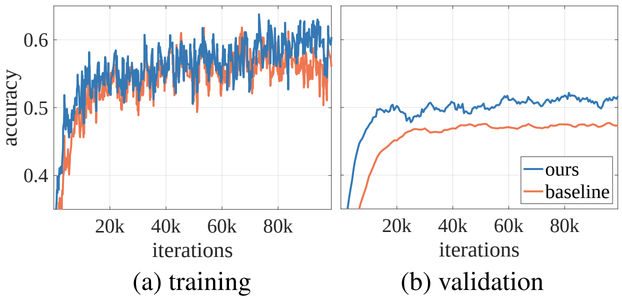
[1]Tianjin University[†] [2]National University of Singapore
[3]Max Planck Institute for Informatics, Saarland Informatics Campus
{liuyaoyao, liuanan, ytsu}@tju.edu.cn
{qsun, schiele}@mpi-inf.mpg.de dcscts@nus.edu.sg

Abstract

Meta-learning has been shown to be an effective strategy for few-shot learning. The key idea is to leverage a large number of similar few-shot tasks in order to meta-learn how to best initiate a (single) base-learner for novel few-shot tasks. While meta-learning how to initialize a base-learner has shown promising results, it is well-known that hyperparameter settings such as the learning rate and the weighting of the regularization term are important to achieve best performance. We thus propose to also meta-learn these hyperparameters and in fact learn a time- and layer-varying scheme for learning a base-learner on novel tasks. Additionally, we propose to learn not only a single base-learner but an ensemble of several base-learners to obtain more robust results. While ensembles of learners have shown to improve performance in various settings, this is challenging for few-shot learning tasks due to the limited number of training samples. Therefore, our approach also aims to meta-learn how to effectively combine several base-learners. We conduct extensive experiments and report top performance for five-class few-shot recognition tasks on two challenging benchmarks: miniImageNet and Fewshot-CIFAR100 (FC100)[1].

1. Introduction

Few-shot learning aims to learn new concepts from a handful of training examples, e.g. from 1 or 5 training images [30, 11, 54]. This ability is well-handled by humans, while in contrast, it remains challenging for machine learning models that typically require a significant amount of training data for good performance [26]. For instance on the CIFAR-100 dataset, a classification model trained in the fully supervised mode achieves 76% accuracy for the 100-class setting [9], while the best-performing 1-shot model achieves only 45% in average for the much simpler 5-class setting [54]. On the other hand, in many real-world applications we are lacking significant amounts of training data, as e.g. in the medical domain. It is thus desirable to improve machine learning models to handle few-shot settings.



(a) training (b) validation

Figure 1. An example of 1-shot result on miniImageNet [59]. X-axis is meta iteration. A point on the curve corresponds to one task and the curves are smoothed with a rate of 0.8. Our approach with multiple base-learners achieves clearly better generalization performance, compared to the baseline model MAML [11].

The nature of few-shot learning with very scarce training data makes is difficult to train powerful machine learning models for new concepts. Meta-learning methods aim to tackle this problem by transferring experience from similar few-shot learning tasks [7]. There are different meta strategies, among which the optimization-based methods are particularly promising for today's neural networks [11, 12, 17, 13, 29, 63, 54, 2]. These methods follow a unified training process that contains two loops. The inner-loop learns a base-learner for an individual task, and the outer-loop then

[*]Equal contribution.
[†]Yaoyao Liu did this work during his internship at NUS.
[1]Code and supplementary materials will be released soon.

uses the validation performance of the learned base-learner to optimize the meta-learner. In previous work [11, 12, 2], the task of the meta-learner is to effectively initialize the base-learner.

In this work we are addressing two shortcomings of previous work. First, the learning process of a base-learner for few-shot tasks is quite unstable [2], and often results in low performance. An intuitive solution is to train an ensemble of models and use the combined prediction which should be more robust [6, 41, 24]. However, it is not obvious how to obtain and combine an ensemble of base-learners given the fact that only few training samples are available. Rather than learning multiple independent base-learners, we propose to use the sequence of base-learners while training a single base-learner as the ensemble and also learn how to weigh them for best performance automatically. Second, it is well known that the value of various hyperparameters are critical for best performance which is particularly important in few-shot learning settings. We thus propose to also meta-learn two important hyperparameters, namely learning rate and regularization weight. We call the resulting novel meta-learning approach **LCC**. **LCC** explicitly **L**earns to **C**ustomize multiple base-learners as well as learns to **C**ombine their prediction results. Our "multiple base-learners" are *different models* since each one of them results from a specific training epoch and is trained with a specific set of hyperparameter values. LCC sets these hyperparameters to be fine-grained, e.g. layer-wise learning rates, in order to enable more efficient model exploration. During test, LCC combines multiple base-learners' predictions using soft weights in order to produce more robust results. Overall, the used hyperparameters and soft weights are also meta-learning targets of LCC. For meta-training we leverage meta gradient descent methods that have been shown effective [11, 54, 2, 12, 45].

Importantly, fast model adaptation is an objective of meta-learning. In the adaptation process, the most active adapting behaviors actually happen in the early epochs, and then converging to and even overfitting to training data in later epochs. Related works use a single base-learner (usually from the last epoch), so their meta-learners learn only partial adaptation experience [11, 54, 12]. By contrast, our LCC leverages an ensemble modeling strategy that adapts base-learners at different training epochs with optimized hyperparameters. Its meta-learner thus obtains the optimized combinational experience. Figure 1 presents that our approach improves the generalization ability substantially over the baseline approach that uses a single base-learner with standard hyperparameters [11].

Our overall contribution is thus three-fold. (1) We propose the novel meta-learning approach LCC that learns to combine an ensemble of base-learners for few-shot learning. LCC both learns how to combine an ensemble of base-learners and learns how to learn these models automatically with fine-grained hyperparameters, e.g. layerwise learning rates and regularization weights. (2) Extensive experiments on two challenging few-shot benchmarks, miniImageNet [59] and Fewshot-CIFAR100 (FC100) [40]. (3) Indepth analysis of the learning process of LCC. We report several interesting observations for automatic adaption. For example, the learning rate of the later-epoch base-learner is often slightly higher, which is opposite to the common schedule, i.e. monotonically decreasing the learning rate, of large-scale network training [18, 55].

2. Related works

Few-shot learning & meta-learning. Research literature on few-shot learning paradigms exhibits a high diversity from using data augmentation techniques [60, 62] over sharing feature representation [3, 61] to meta-learning [16, 58]. In this paper, we focus on the meta-learning paradigm that leverages few-shot learning experiences from similar tasks, based on the episodic formulation (see Section 3.1). Related work can be roughly divided into three categories: (1) metric learning methods [59, 49, 57] aim to learn a similarity space, in which the learning should be efficient for few-shot examples; (2) memory network methods [37, 46, 40, 35] aim to learn training "experience" from seen tasks and then aim to generalize to the learning of unseen ones; and (3) gradient descent based methods [11, 12, 2, 43, 29, 17, 63, 54] usually employing a meta-learner that learns to fast adapt a NN base-learner to a new task within a few iterations. State-of-the-art models are MAML [11] and its recent improved version MAML++ [2]. Their meta-learners learn to effectively initialize the parameters of a NN base-learner for a new task. Our approach is closely related to MAML related methods [11, 2]. An important difference is that we learn how to customize and how to combine an ensemble of base-learners for robust model prediction, while MAML [11] and MAML++ [2] use a single base-learner.

Hyperparameter optimization. Building a model for a new task is a process of exploration-exploitation. Exploring suitable architectures and hyperparameters are important before training. Traditional methods are model-free, e.g. grid search. Bergstra and Bengio [5] advocated using random search over grid search. Li *et al.* [31] improved random search by adaptively allocating resources to promising configurations. Jaderberg *et al.* [23] scheduled a population of networks in parallel, and periodically replace the weights of under-performing networks by better ones. These methods require multiple full training trials and are thus costly. Model-based hyperparameter optimization methods are adaptive but sophisticated, e.g. using random forests [20], Gaussian processes [50] and input warped Gaussian processes [52] or scalable Bayesian optimization [51]. In our approach, we meta-learn hyperparameters by a simple and elegant gradient descent method, without additional manual labor. Related methods using gradient descent mostly work for single network training [4, 10, 33, 32, 13, 34]. While, we aim to learn a sequence of hyperparameters for multiple base-learners.

Ensemble modeling. It is a strategy that aims to improve machine learning performance using multiple algorithms, and has proved to effectively reduce problems related to overfitting [27, 53]. Mitchell *et al.* [36] provided a theoretical explanation for it. Boosting is one classical way to build an ensemble by training new models with emphasizing hard samples, e.g. AdaBoost [14] and Gradient Tree Boosting [15]. Stacking combines multiple models by learning a combiner like a logistic regression model. It applies to both supervised learning tasks [6, 41, 24] and unsupervised learning [48]. Bootstrap aggregating (bagging) builds an ensemble using models generated in parallel to reduce the variance [6], e.g. Random Forests [19]. In few-shot settings, it is hard to train plenty of different models in parallel. Our approach makes use of the ensemble of training epochs to obtain different models. Ensembling models in a temporal way [28] and utilizing features extracted by an ensemble of attribute models [56] are also related works. Comparing to them, our difference lies in that our multiple models are customized with optimized hyperparameters and combined with learned weights, automatically.

3. Preliminary

This section first introduces the unified episodic formulation of few-shot learning, following related works [59, 43, 11, 40, 54, 45]. Then, we briefly introduce the meta gradient decent of meta-learner based on a single base model, which is commonly used in related works [11, 12, 54, 2].

3.1. Episodic formulation

The episodic formulation was proposed for few-shot learning first in [59]. The problem definition of few-shot learning is different from traditional image classification, in three aspects: (1) the main phases are not train and test but meta-train and meta-test, each of which includes training and testing; (2) the samples in meta-train and meta-test are not datapoints but episodes, i.e. few-shot classification tasks; and (3) the objective is not classifying unseen datapoints but to fast adapt the meta-learned experience or knowledge to the learning of a new few-shot classification task.

Given a dataset $\mathcal{D}$ for meta-train, we first sample few-shot episodes (tasks) $\{\mathcal{T}\}$ from a task distribution $p(\mathcal{T})$ such that each episode $\mathcal{T}$ contains few samples of few classes, e.g. 5 classes and 1 shot per class. Each episode $\mathcal{T}$ includes a training split $\mathcal{T}^{(tr)}$ to optimize a specific base-learning network, and a test split $\mathcal{T}^{(te)}$ to compute a generalization loss used to optimize a global meta-learner. For meta-test, given an unseen dataset $\mathcal{D}_{un}$, we sample a test task $\mathcal{T}_{un}$ to have the same-size training/test splits. "Unseen" means there is no overlap of image classes between meta-test and meta-train tasks. We first initiate a new model with meta-learned network parameters (ours with additional hyperparameters), then train this model on the training split $\mathcal{T}_{un}^{(tr)}$. We finally evaluate the performance on the test split $\mathcal{T}_{un}^{(te)}$. If we have multiple unseen tasks for meta-test, we report average accuracy as the final result.

3.2. Meta gradient descent

Meta gradient descent is a classical way of outer-loop optimization [58, 47, 39]. MAML [11] first applied this to supervised meta-learning and reinforcement learning. It optimizes meta parameters θ (meta-learner) that are used to initialize a specific model Θ (base-learner) for fast adaption to a new task [11]. It trains a single base-learner for prediction in each episode.

Given an episode $\mathcal{T} = \{\mathcal{T}^{(tr)}, \mathcal{T}^{(te)}\}$, we initialize the base-learner parameters Θ as $\Theta_0 \leftarrow \theta$, then adapt it by using gradient descent using the loss on the training datapoints $\mathcal{T}^{(tr)}$,

$$\Theta_{m+1} \leftarrow \Theta_m - \alpha \nabla_{\Theta_m} \mathcal{L}_\lambda(\mathcal{T}^{(tr)}, \Theta_m), \tag{1}$$

where $\mathcal{L}_\lambda$ is the penalty with a fixed hyperparameter λ, α is a fixed learning rate and m the epoch number. Each base-training contains M epochs. After M epochs, a validation loss of $\mathcal{T}^{(te)}$ is computed based on Θ_M. The corresponding gradient on θ is called meta gradient, and it unrolls through the entire base adaptation procedure from Θ_M to Θ_0 (*i.e.* the θ itself). The update of θ is thus to apply a meta gradient descent computation as follows,

$$\theta =: \theta - \beta \nabla_\theta \mathcal{L}_\lambda(\mathcal{T}^{(te)}, \Theta_M). \tag{2}$$

where β is the meta-learning rate. This meta gradient update involves a gradient through a gradient. It requires an additional backward pass through the base-learner to compute Hessian-vector products [11], and this is supported by standard libraries such as TensorFlow [1]. In the following, we show how to leverage meta gradient descent within our approach.

4. Learning to Customize and Combine (LCC)

As shown in Figure 2, our LCC both learns a sequence of base-learners and learns to combine their prediction scores during test for best performance. Hyperparameters are learned by meta gradients automatically.

4.1. Initiate the sequence of base-learners

We use the sequence of base-learners obtained from training a single base-learner as the ensemble. We thus

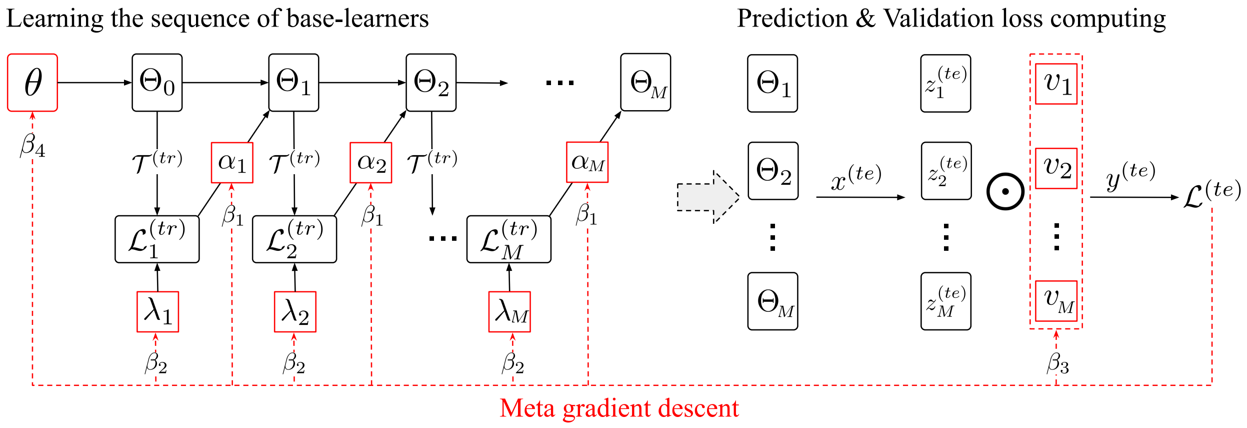


Figure 2. The overall computing flow of our LCC, on one training task. LCC learns to combine the sequence of base-learners, with network weights denoted as $\Theta_{1\sim M}$, while training a single base-learning network as the ensemble. For prediction, it uses the weighted sum of the scores predicted by those base-learners. Finally, it uses the validation loss $\mathcal{L}^{(te)}$ for meta gradient back-propagation, in order to optimize the key hyperparameters and combination weights, namely $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$ and $\mathbf{v}$. Note that initialization parameters θ are also optimized.

can formulate the initiation of these base-learners in a sequential manner. Our "initiation" here includes the initialization of neural network parameters, i.e., weights and bias (the initialization of the 1-st base-learner is the same as for MAML [11]), as well as the configuration of specific hyperparameters, for the sequence of base-learners.

Given an episode $\mathcal{T} = \{\mathcal{T}^{(tr)}, \mathcal{T}^{(te)}\}$, let Θ_m corresponds to the parameters of the base-learner working at epoch m (w.r.t. the m-th base-learner or BL-m), with $m \in \{1, ..., M\}$. First, we initiate BL-1 with the initialization parameters θ (network weights and bias), as well as with specific hyperparameters, i.e. learning rates α_m and regularization weights λ_m. We then adapt BL-1 using gradient descent on the training split $\mathcal{T}^{(tr)}$, and its updated weights and bias are then used to initialize the parameters of BL-2. We formulate the general process as follows,

$$\Theta_0 \leftarrow \theta, \tag{3}$$

$$\Theta_m \leftarrow \Theta_{m-1} - \alpha_m \nabla_\Theta \mathcal{L}_m^{(tr)}, \tag{4}$$

where α_m denotes the learning rate specified for BL-m, and $\mathcal{L}_m^{(tr)}$ is the training loss. Note that Θ_0 is introduced to make the notation consistent. If we use $F(x; \Theta_m)$ to initialize function BL-m mapping the inputs to the prediction scores, the training loss of $\mathcal{T}^{(tr)} = \{x_j^{(tr)}, y_j^{(tr)}\}_{j=1}^{N_1}$ can be unfolded as,

$$\begin{aligned}\mathcal{L}_m^{(tr)} = \frac{1}{N_1} \sum_{j=1}^{N_1} L_{ce}\big(F(x_j^{(tr)}, \Theta_{m-1}), y_j^{(tr)}\big) \\ + \lambda_m \left\| \Theta_{m-1} \right\|_2^2,\end{aligned} \tag{5}$$

where L_{ce} is the softmax cross entropy loss, and $\|\Theta_m\|_2$ is the regularization of network weights and λ_m is the regularization weight specified for BL-m. The meta optimization on hyperparameters α_m and λ_m is given in Section 4.2.

4.2. Learn to customize base-learners

As introduced in Section 4.1, the specific learning rate α_m and regularization weight λ_m are used to configure the m-th base-learner. It is well known that fine-grained hyperparameters, e.g. layerwise learning rates, are more efficient, but exponentially expensive to set by hand [21, 5]. Our LCC does not have this problem and can learn fine-grained hyperparameters without additional labour. Therefore, we use layerwise learning rates and regularization weights as $\boldsymbol{\alpha}_m = \{\alpha_{m,k}\}_{k=1}^K$ and $\boldsymbol{\lambda}_m = \{\lambda_{m,k}\}_{k=1}^K$, where K is the layer number. When plugging $\boldsymbol{\alpha}_m$ and $\boldsymbol{\lambda}_m$ into Eq. 4, we get all base-learners with fine-grained customization.

Our LCC automatically optimizes $\boldsymbol{\alpha}_m$ and $\boldsymbol{\lambda}_m$ by meta gradient descent. First, it computes the validation loss on the test split $\mathcal{T}^{(te)} = \{x_j^{(te)}, y_j^{(te)}\}_{j=1}^{N_2}$ as,

$$\mathcal{L}^{(te)} = \frac{1}{N_2} \sum_{j=1}^{N_2} L_{ce}\big(\hat{y}_j^{(te)}, y_j^{(te)}\big), \tag{6}$$

which is based on the sequence of base-learners. $\hat{y}$ denotes the combination of their predictions, and its detailed computation is given in Section 4.3.

Then, it uses $\mathcal{L}^{(te)}$ to compute meta gradients of $\boldsymbol{\alpha}$ or $\boldsymbol{\lambda}$, which unrolls the entire adaptation process on the sequence of base-learners back to the initiation step. Thus, the sequence of involved hyperparameters $\boldsymbol{\alpha} = \{\boldsymbol{\alpha}_m\}_{m=1}^M$ or $\boldsymbol{\lambda} = \{\boldsymbol{\lambda}_m\}_{m=1}^M$ can be updated as,

$$\boldsymbol{\alpha} =: \boldsymbol{\alpha} - \beta_1 \nabla_{\boldsymbol{\alpha}} \mathcal{L}^{(te)}, \tag{7}$$

$$\boldsymbol{\lambda} =: \boldsymbol{\lambda} - \beta_2 \nabla_{\boldsymbol{\lambda}} \mathcal{L}^{(te)}, \tag{8}$$

where β_1 and β_2 are meta-learning rates determining the update stepsize of hyperparameter values.

The meta updates in Eq. 7 and Eq. 8 involve the backward pass through BL-M to BL-1. Derivatives are back-propagated through the unfolded inner loop (of every base-learner) which contains all convolutional and fully-connected layers. The corresponding layerwise learning rates and regularization weights thus all get updated.

4.3. Learn to combine base-learners

As introduced in Sections 4.1 and 4.2, our LCC optimizes the parameters as well as hyperparameters for the sequence of base-learners. For prediction, it uses the weighted sum of the sequence of prediction scores (from all base-learners). It optimizes the combination weights by meta gradient descent.

First, we formulate the prediction scores z of a single base-learner as:

$$z = F(x; \Theta). \tag{9}$$

For multiple base-learners, we define the combination weights as $\mathbf{v} = \{v_m\}_{m=1}^{M}$, and thus compute the combination as follows,

$$z = \sum_{m=1}^{M} v_m F(x; \Theta_m). \tag{10}$$

Similar to the meta updates on $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$, we update $\mathbf{v}$ as,

$$\mathbf{v} =: \mathbf{v} - \beta_3 \nabla_{\mathbf{v}} \mathcal{L}^{(te)}, \tag{11}$$

where β_3 is the stepsize of this update. $\mathcal{L}^{(te)}$ is the validation loss as follows,

$$\mathcal{L}^{(te)} = \frac{1}{N_2} \sum_{j=1}^{N_2} L_{ce}\Big(\sum_{m=1}^{M} v_m F\big(x_j^{(te)}; \Theta_m\big), y_j^{(te)} \Big), \tag{12}$$

which uses the weighted sum of all model predictions, and is also the expanded version of Eq. 6.

4.4. Overall optimization and algorithm

When including the initialization parameters θ [11] (for the initialization of 1st base-learner), we have the overall formulation of meta-parameterization as:

$$[\theta; \boldsymbol{\alpha}; \boldsymbol{\lambda}; \mathbf{v}] =: [\theta; \boldsymbol{\alpha}; \boldsymbol{\lambda}; \mathbf{v}] - \boldsymbol{\beta} \odot \nabla \mathcal{L}_{meta}, \tag{13}$$

where $\boldsymbol{\beta} = \{\beta_c\}_{c=1}^{4}$ and β_4 is the stepsize for updating θ. For the computation of $\mathcal{L}_{meta}$, we apply the meta-batch strategy in the iteration of episode training, following [11]. At each iteration, we sample a batch of P episodes $\{\mathcal{T}_i\}_{i=1}^{P}$ and then compute the average validation loss as,

$$\mathcal{L}_{meta} = \frac{1}{P} \sum_{i=1}^{P} \mathcal{L}\big(\mathcal{T}_i^{(te)}; \boldsymbol{\alpha}, \boldsymbol{\lambda}, \mathbf{v}\big). \tag{14}$$

Algorithm 1 summarizes the meta-train (line 2-10) and meta-test (line 12-17) procedures of our LCC approach. For completeness, the base-learning updates in a single episode are given in Algorithm 2.

Algorithm 1: Learn to customize and combine (LCC)

```
Input: Meta-train episode distribution $p_{tr}(\mathcal{T})$,
       Meta-test episode distribution $p_{te}(\mathcal{T})$, and
       meta-train stepsizes $\boldsymbol{\beta}$.
Output: The average accuracy of meta-test.
1  % Meta-train phase:
2  Randomly initialize $\theta$;
3  for all meta iterations do
4      Sample a batch of meta-train episodes
         $\{\mathcal{T}_i\} \in p_{tr}(\mathcal{T})$;
5      for $\mathcal{T}_i$ in $\{\mathcal{T}_i\}$ do
6          Train the sequence of base-learners on $\mathcal{T}_i$ by
             Algorithm 2;
7      end
8      Evaluate $\mathcal{L}_{meta}$ with Eq. 14 ;
9      Optimize $\theta$, $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$ and $\mathbf{v}$ with Eq. 13 using $\boldsymbol{\beta}$.
10 end
11 % Meta-test phase:
12 Sample meta-test episodes $\{\mathcal{T}_i\} \in p_{te}(\mathcal{T})$;
13 for $\mathcal{T}_i$ in $\{\mathcal{T}_i\}$ do
14     Train the sequence of base-learners on $\mathcal{T}_i$ and
         obtain the prediction scores $z_i$ by Algorithm 2;
15     Compute episode test accuracy $Acc_i$;
16 end
17 Return the average accuracy of $\{Acc_i\}$.
```

Algorithm 2: Update the sequence of base-learners

```
Input: An episode $\mathcal{T}$, initialization parameters $\theta$,
       learning rates $\boldsymbol{\alpha}$, regularization weights $\boldsymbol{\lambda}$,
       combination weights $\mathbf{v}$.
Output: Prediction scores $z$, and episode test loss
        $\mathcal{L}^{(te)}$.
Initialize $\Theta_0 = \theta$;
for $m$ in $\{1, ..., M\}$ do
    Evaluate $\mathcal{L}_m^{(tr)}$ using $\boldsymbol{\lambda}_m$ with Eq. 5;
    Get $\Theta_m$ using $\boldsymbol{\alpha}_m$ with Eq. 4;
end
Compute $z$ using $\mathbf{v}$ with Eq. 10;
Evaluate $\mathcal{L}^{(te)}$ with Eq. 12.
```

5. Experiments

We evaluate and analyze the proposed **LCC** approach in terms of its overall performance and the effects from its two components, i.e. using multiple base-learners and meta-learning hyperparameters. We first describe the datasets and detailed settings, then compare the results to state-of-the-art methods and conduct an ablation study.

5.1. Datasets and implementation details

We conduct few-shot learning experiments on two benchmarks: miniImageNet [59] and Fewshot-CIFAR100 (FC100) [40]. The former one is widely used in related works [11, 43, 17, 13, 38, 54], and the later one is more challenging due to lower image resolution and harder training-test splits [40, 54].

miniImageNet was proposed by Vinyals *et al.* [59] for evaluation of few-shot learning. It is complex because of using ImageNet images, but requires fewer resources and infrastructure than running models on full ImageNet [44]. There are 100 classes with 600 samples of 84×84 color images per class. Classes are divided into 64, 16, and 20 classes respectively for sampling tasks for meta-training, meta-validation and meta-test, following related works [11, 43, 17, 13, 38].

Fewshot-CIFAR100 (FC100) is based on the popular object classification dataset CIFAR100 [25]. The splits were proposed by [40], see details in the supplementary. It offers a more challenging scenario with lower image resolution and more challenging meta-train/meta-test splits (separated according to the super-classes of objects) than miniImageNet. It contains 100 object classes and each class has 600 samples of 32×32 color images per class. The 100 classes belong to 20 super-classes. Meta-train data are from 60 classes belonging to 12 super-classes. Meta-validation and meta-test data are from the other two 20 classes belonging to 4 super-classes, respectively. These splits according to super-classes minimize the information overlap between meta-train and meta-test (meta-validation) tasks.

The following settings are shared for both datasets. We use the same task sampling used in related works [11, 43, 12, 2]. Specifically, we consider the 5-class classification and sample 5-class, 1-shot (5-shot or 10-shot) episodes to contain 1 (5 or 10) samples as episode train data, and 15 (a uniform number) samples as episode test data. In total, we sample $240k$ tasks for meta-training, and respectively sample 600 random tasks for meta-validation and meta-test.

The base architecture is **4CONV**, which is commonly used in related works [43, 11, 49, 57, 17, 2]. **4CONV** consists of 4 layers with 3×3 convolutions and 32 filters, followed by batch normalization (BN) [22], a ReLU nonlinearity, a 2×2 max-pooling layer, and a fully-connected layer.

The configuration of meta-learners. The network initialization parameters θ have the same architecture as the base-learner, except that the BN, non-linear and max-pooling layer are removed. The architectures of $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ depend on both the number and architecture of the base-learner. In our default setting, 5 base-learners with 4CONV architecture are learned in the ensemble, so $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ consist of 50 (for weights and biases) and 25 (only for weights) different variables, respectively. The architecture of the combination weights $\mathbf{v}$ is related to the number of base-learners in the ensemble, so it has 5 variables.

The initialization of meta-learners. θ is initialized randomly, which is the same as MAML [11]. All weights of $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ are initialized with 0.01 and 0.001 respectively. All the weights of $\mathbf{v}$ are initialized with the reciprocal of the base-learner number, i.e. $[0.2, 0.2, 0.2, 0.2, 0.2]$.

The hyperparameters of meta-learners. The meta iteration number is set to 60k and 50k for MAML and MAML++ respectively. The meta batch size is 4, and the meta learning rate for the initialization parameters θ is 0.001 (β_4). All the above settings exactly follow [11] and [2]. For the new added meta-learners, the meta learning rates are set to $\beta_1 = 0.0001$, $\beta_2 = 0.00001$, and $\beta_3 = 0.0001$ for $\boldsymbol{\alpha}$, $\boldsymbol{\lambda}$, and $\mathbf{v}$ respectively.

The most related methods. MAML [11] is commonly used as baseline, and MAML++ [2] is the most recently published state-of-the-art method also using 4CONV as base architecture. MAML++ introduced six training tips which contribute to stable and efficient meta-training process. Our approach is called **LCC**. If we use the training tips of MAML++, we obtain an improved version called **LCC++**. Note that LCC++ and MAML++ have the overlap of learning layerwise learning rates. For this part, we use our implementation as we can set flexible stepsizes for the meta update. Therefore, LCC++ actually uses the other five training tips of MAML++.

5.2. Results and analyses

We conduct extensive few-shot learning experiments. In Table 1 and Table 2, we present our results compared to the state-of-the-art, respectively on the miniImageNet and FC100 datasets. In Table 3, we provide an ablation study for several components of our approach, on miniImageNet. In Figure 4, we show the specific changes on the recognition accuracies in different ablative settings. In Figure 3, we particularly plot the weight changes of multiple base-learners during meta-learning in (a), and show its boost performance compared to baseline settings in (b), as "multiple base-learners" is one of our main contributions. For the other contribution of "meta-learning hyperparameter", we plot extensive curves in Figure 5 and Figure 6.

Overview on miniImageNet. In Table 1, we can see that our LCC++ achieves the best performance in both 1-shot (54.6%) and 5-shot (71.1%) settings, compared to the methods with the same 4CONV architecture. Only methods [40, 54, 45, 42] that use deeper neural networks with expensive pre-training as an important pre-processing step do obtain higher performance. Similarly, we expect further gains of our approach using similar pre-training strategies.

Overview on FC100. In Table 2, we present the results of TADAM [40] and MTL [54] using their reported numbers. We note that the numbers of MAML are from [54], and those of MAML++ are our results using the public code.

Method	Arch.	1-shot	5-shot
TADAM [40]‡	ResNet12	58.5 ± 0.3	76.7 ± 0.3
MTL [54]‡	ResNet12	61.2 ± 1.8	75.5 ± 0.8
LEO [45]‡	WRN-28	61.76 ± 0.08	77.59 ± 0.12
PFA [42]‡	WRN-28	59.60 ± 0.41	73.74 ± 0.19
MatchingNets [59]	4CONV	43.44 ± 0.77	55.31 ± 0.73
ProtoNets [49]	4CONV	49.42 ± 0.78	68.20 ± 0.66
Meta-LSTM [43]	4CONV	43.56 ± 0.84	60.60 ± 0.71
Bilevel [13]	4CONV	50.54 ± 0.85	64.53 ± 0.68
CompareNets [57]	4CONV	50.44 ± 0.82	65.32 ± 0.70
LLAMA [17]	4CONV	49.40 ± 1.83	–
Baseline++ [8]	4CONV	48.24 ± 0.75	66.43 ± 0.63
MAML [11]	4CONV	48.70 ± 1.75	63.11 ± 0.92
MAML++ [2]	4CONV	52.15 ± 0.26	68.32 ± 0.44
LCC (Ours)	4CONV	54.0 ± 1.8	65.8 ± 0.9
LCC++ (Ours)	4CONV	**54.6** ± 0.4	**71.1** ± 0.4

‡Pre-trained on many-shot classification task

Table 1. Few-shot classification accuracy (%) on miniImageNet.

When comparing methods using the same base learning architecture 4CNOV, that is LCC vs MAML and LCC++ vs. MAML++, we can see that our approach LCC (LCC++) obtains better performance. For example, LCC++ achieves 1.0%, 2.3%, and 2.1% improvement on 1-shot, 5-shot, and 10-shot respectively over MAML++. Quite interestingly, on this more challenging dataset, our approach (4CONV) achieves comparable results to TADAM which uses a pre-trained and deeper network (ResNet12).

Multiple base-learners with learnable weights $\mathbf{v}$. In Table 3, we can see that with fixed α and λ, our approach using multiple base-learners with learnable weighting scheme (No.6) performs better than a single base-learner (No.2) as well as multiple base-learners with fixed average weights (No.1). Please note that No.2 essentially corresponds to the setting of MAML, but the results here are slightly lower than reported in Table 1 (48.70%, 63.11%). This is because the original MAML ran meta-train with 5 epochs and ran meta-test with 10 epochs. Here, we report the results of using meta-test with 5 epochs (for fair comparison with our approach which also uses 5 epochs). The last row of Table 3 shows the "oracle" results by assuming the "Optimal" values of $\mathbf{v}$ have been learned by LCC and are fixed during training. They are clearly higher than the results of any arbitrary $\mathbf{v}$ (No.1 or No.2), especially in the 1-shot setting.

Method	1-shot	5-shot	10-shot
TADAM [40]‡	40.1 ± 0.4	56.1 ± 0.4	61.5 ± 0.5
MTL [54]‡	45.1 ± 1.8	57.6 ± 0.9	63.4 ± 0.8
MAML [11]◇	38.1 ± 1.7	50.4 ± 1.0	56.2 ± 0.8
MAML++ [2]†	38.7 ± 0.4	52.9 ± 0.4	58.8 ± 0.4
LCC (Ours)	**40.6** ± 1.8	52.7 ± 0.9	56.9 ± 0.8
LCC++ (Ours)	39.7 ± 0.4	**55.2** ± 0.4	**60.9** ± 0.4

◇Reported in [54]

†Our implementation using the public code

‡Pre-trained on many-shot classification task

Table 2. Few-shot classification accuracy (%) on FC100.

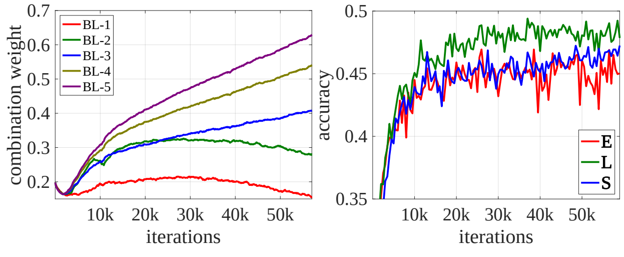
No.	Meta-learned			Accuracy	
	$\boldsymbol{\alpha}$	$\boldsymbol{\lambda}$	$\mathbf{v}$	1-shot	5-shot
1			**E**	47.0 ± 1.8	62.0 ± 0.9
2			**S**	48.0 ± 1.8	62.4 ± 0.9
3	✓		**S**	49.7 ± 1.8	64.4 ± 0.9
4		✓	**S**	49.0 ± 1.8	63.4 ± 0.9
5	✓	✓	**S**	49.0 ± 1.8	65.0 ± 0.9
6			**L**	49.7 ± 1.8	65.4 ± 0.9
7	✓		**L**	52.9 ± 1.8	65.6 ± 0.9
8		✓	**L**	48.6 ± 1.8	64.7 ± 0.9
LCC(Ours)	✓	✓	**L**	**54.0** ± 1.8	**65.8** ± 0.9
"oracle" $\mathbf{v}$			**O**	52.4 ± 1.8	64.7 ± 0.9

Table 3. Ablation results (%) on miniImageNet. **L** denotes that $\mathbf{v}$ is Learnable; **S** means using a Single base-learner, i.e. $\mathbf{v}$ is fixed as $[0, 0, 0, 0, 1]$; and **E** denotes an ablation case with fixed Equal weights $\mathbf{v} = [0.2, 0.2, 0.2, 0.2, 0.2]$. The last row shows the "oracle" by assuming the "Optimal" (denoted as **O**) values of $\mathbf{v}$ have been learned by LCC(Ours) and are fixed during training.

For No.1, 2, 6, the validation accuracies during meta-train are shown in Figure 3(b). No.1 gives BL-1 with weights fixed to 0.2 which causes stronger fluctuations in later iterations (red curve). By contrast, our method automatically adjusts this weight to close to 0 when other learners become mature, see Figure 3(a). With automatic tuning, our approach performs the best during the entire meta-train.

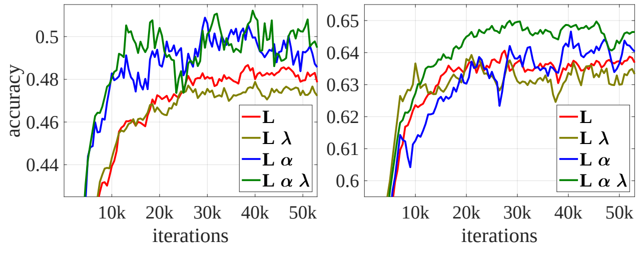
From Figure 3(a) we can observe that the weights of the 5 base-learners are initialized as 0.2 and then adapted over time. Intuitively, an increase relates to the fact that a base-learners become more mature in later iterations. Interestingly, base-learners working at later epochs gain relatively higher weights but the base-learner working at the initial epoch (the BL-1) tends to be disabled when the meta-train process converges after around $30k$ iterations.

Meta-learning hyperparameters $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$. The fine-grained hyperparameters, i.e. layerwise learning rates $\boldsymbol{\alpha}$ and regularization weights $\boldsymbol{\lambda}$, can be automatically learned by our LCC approach. In Table 3, we have two blocks to present the ablative results of using single base-learner (**S**) and using multiple base-learners with learnable combination weights (**L**) in the miniImageNet 1-shot and 5-shot settings. Particularly in Figure 4, we demonstrate the validation curves (the curve smooth rate is 0.7) of the whole meta-train processes. It is clearly shown that our approach with



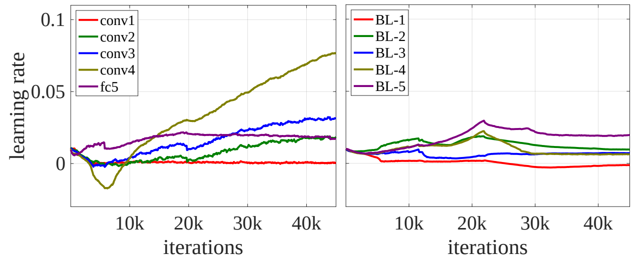
(a) (b)

Figure 3. Meta-learned $\mathbf{v}$ value and meta validation accuracy on miniImageNet 1-shot. (a) The changes of $\mathbf{v}$ values during the meta-training. (b) The meta validation accuracy comparison for different settings of $\mathbf{v}$. **E**, **S**, and **L** are from Table 3 and curves in (b) correspond to No.1, No.2 and No.6, respectively.



(a) (b)

Figure 4. Meta validation results of our ablation study on miniImageNet 1-shot (a) and 5-shot (b). From top to down, they correspond to the No.6, No.7, No.8 and LCC(Ours) in Table 3.

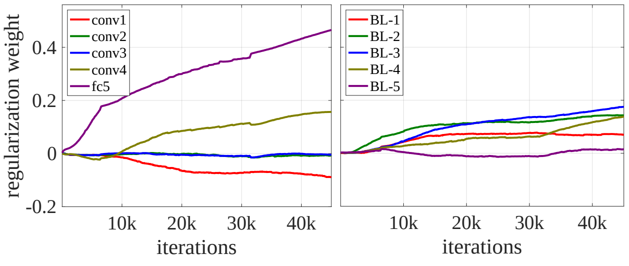


(a) (b)

Figure 5. Learning rates $\boldsymbol{\alpha}$ in the meta-learning procedure. (a) Values of each layer averaged over 5 base-learners; (b) Using a single learning rate for each base-learner (i.e., α is a learnable scale but not layerwise).

meta-learned hyperparameters achieves top performance.

About $\boldsymbol{\alpha}$. In Table 3, comparing No.3 to No.2 and No.7 to No.6, we can conclude that meta-learning layerwise $\boldsymbol{\alpha}$ consistently improves the model performance, e.g. it gains 3.2% for the case of using multiple base-learners in 1-shot setting. We can observe the change of layerwise $\boldsymbol{\alpha}$ in Figure 5(a), and the change of a single learning rate for each base-learner in Figure 5(b). Note that (a) shows the results of using multiple base-learners, for which the curve of a specific layer is obtained by averaging over the exact layers of all base-learners. In this "averaged base-learner", we can observe that higher-level conv layers learn to increase their learning rates. It is quite amazing that conv4 explores a much bigger learning step, around 70 times higher than its initial value of 0.01. While in (b), when each base-learner has a single learning rate to learn, the global change of this rate is in a small range, e.g. the biggest jump in BL-5(purple) is from 0.01 to around 0.03.



(a) (b)

Figure 6. Regularization weights $\boldsymbol{\lambda}$ in the whole meta-learning procedure. (a) Values of each layer averaged over 5 base-learners; (b) Values of each base-learner averaged over 5 layers.

It also shows in (b) that base-learners working at later epochs tend to get higher learning rates. This is opposite to the common schedule, i.e. monotonically decreasing the learning rate, of traditional large-scale network training [18]. In our few-shot case, this increasing phenomenon can be interpreted as: our LCC learns to update more on the base-learners which have both maturer patterns and higher combination weights (i.e. v values), and in turn gets greater feedback from them for meta optimization.

About $\boldsymbol{\lambda}$. In Table 3, comparing No.4 to No.3 and No.8 to No.7, we can see that meta-learning $\boldsymbol{\lambda}$ does not help as much as meta-learning $\boldsymbol{\alpha}$. While, meta-learning them together (i.e. **L** $\boldsymbol{\alpha}$ $\boldsymbol{\lambda}$) makes consistent improvements. Figure 4 gives more detailed results. The superiority of learning $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$ is significant in the 5-shot case (b).

In Figure 6, we present the curves of meta-learned $\boldsymbol{\lambda}$. In (a), the $\boldsymbol{\lambda}$ value of an individual layer is the average of those of 5 base-learners. We can see that high-level layers learn to have higher $\boldsymbol{\lambda}$ values. We believe this is a collaborating behavior with the simultaneously meta-learned $\boldsymbol{\alpha}$ which also gets increased in higher-level layers (Figure 5(a)). An intuitive interpretation is that heavily penalizing peaky weights is needed when the weights are updated with large steps. Another interesting point in Figure 6(a) is the values of $\boldsymbol{\lambda}$ of conv1 become negative after $10k$ iterations. This can be explained that the gradient vanishing problem probably happens during the training with very scarce samples. Our LCC learns to use negative $\boldsymbol{\lambda}$ to penalize such vanishing. Figure 6(b) shows that LCC can learn to adapt the values of $\boldsymbol{\lambda}$ for multiple base-learners. For convenient visualization, layerwise $\boldsymbol{\lambda}$ values of each base-learner are averaged.

6. Conclusions

We propose a novel LCC approach that learns to customize a sequence of base-learns and learns to combine their prediction results. It addresses shortcomings of previous meta-learning approaches by meta-learning hyperparameters both layer-wise as well as over time and allows to use an ensemble of base-learners. Following the meta-learning paradigm, the method allows to achieve top performance in comparison to related work. The design of our approach is independent from a specific base-learning model, i.e. base-learner architecture, and can be generalized also to pre-trained and deeper networks.

Acknowledgments

This research is part of NExT++ research supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC@SG Funding Initiative. It is also partially supported by German Research Foundation (DFG CRC 1223), and National Natural Science Foundation of China (61772359).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 1603.04467, 2016.

[2] A. Antoniou, H. Edwards, and A. Storkey. How to train your maml. In *ICLR*, 2019.

[3] E. Bart and S. Ullman. Cross-generalization: Learning novel classes from a single example by feature replacement. In *CVPR*, 2005.

[4] Y. Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[6] L. Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

[7] R. Caruana. Learning many related tasks at the same time with backpropagation. In *NIPS*, 1994.

[8] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. Wang, and J.-B. Huang. A closer look at few-shot classification. In *ICLR*, 2019.

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

[10] J. Domke. Generic methods for optimization-based modeling. In *AISTATS*, 2012.

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

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

[13] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *ICML*, 2018.

[14] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[15] J. H. Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

[16] H. E. Geoffrey and P. C. David. Using fast weights to deblur old memories. In *CogSci*, 1987.

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

[18] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li. Bag of tricks for image classification with convolutional neural networks. *arXiv*, 2018.

[19] T. K. Ho. Random decision forests. In *ICDAR*, 1995.

[20] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION*, 2011.

[21] F. Hutter, J. Lücke, and L. Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI*, 29(4):329–337, 2015.

[22] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[23] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu. Population based training of neural networks. *arXiv*, 1711.09846, 2017.

[24] C. Ju, A. Bibaut, and M. J. van der Laan. The relative performance of ensemble methods with deep convolutional neural networks for image classification. *arXiv*, 1704.01664, 2017.

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

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[27] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[28] S. Laine and T. Aila. Temporal ensembling for semi-supervised learning. In *ICLR*, 2017.

[29] Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *ICML*, 2018.

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

[31] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18:185:1–185:52, 2017.

[32] J. Luketina, T. Raiko, M. Berglund, and K. Greff. Scalable gradient-based tuning of continuous regularization hyperparameters. In *ICML*, 2016.

[33] D. Maclaurin, D. K. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015.

[34] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. In *ICLR*, 2019.

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

[36] T. Mitchell. Machine learning, mcgraw-hill higher education. *New York*, 1997.

[37] T. Munkhdalai and H. Yu. Meta networks. In *ICML*, 2017.

[38] T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler. Rapid adaptation with conditionally shifted neurons. In *ICML*, 2018.

[39] D. K. Naik and R. Mammone. Meta-neural networks that learn by learning. In *IJCNN*, 1992.

[40] B. N. Oreshkin, P. Rodríguez, and A. Lacoste. TADAM: task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018.

[41] M. Ozay and F. T. Y. Vural. A new fuzzy stacked generalization technique and analysis of its performance. *arXiv*, 1204.0171, 2012.

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

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

[44] 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.

[45] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019.

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

[47] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[48] P. Smyth and D. Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83, 1999.

[49] J. Snell, K. Swersky, and R. S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017.

[50] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

[51] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable bayesian optimization using deep neural networks. In *ICML*, 2015.

[52] J. Snoek, K. Swersky, R. S. Zemel, and R. P. Adams. Input warping for bayesian optimization of non-stationary functions. In *ICML*, 2014.

[53] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In *NIPS*, 1995.

[54] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, 2019.

[55] Q. Sun, L. Ma, S. Joon Oh, L. Van Gool, B. Schiele, and M. Fritz. Natural and effective obfuscation by head inpainting. In *CVPR*, 2018.

[56] Q. Sun, B. Schiele, and M. Fritz. A domain based approach to social relation recognition. In *CVPR*, 2017.

[57] 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.

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

[59] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *NIPS*, 2016.

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

[61] Y.-X. Wang and M. Hebert. Learning from small sample sets by combining unsupervised meta-training with cnns. In *NIPS*, 2016.

[62] Y. Xian, S. Sharma, B. Schiele, and Z. Akata. f-VAEGAN-D2: A feature generating framework for any-shot learning. In *CVPR*, 2019.

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