Meta-Learning with Differentiable Convex Optimization

Kwonjoon Lee[2]   Subhransu Maji[1,3]   Avinash Ravichandran[1]   Stefano Soatto[1,4]

[1]Amazon Web Services   [2]UC San Diego   [3]UMass Amherst   [4]UCLA

kwl042@ucsd.edu   {smmaji,ravinash,soattos}@amazon.com

Abstract

Many meta-learning approaches for few-shot learning rely on simple base learners such as nearest-neighbor classifiers. However, even in the few-shot regime, discriminatively trained linear predictors can offer better generalization. We propose to use these predictors as base learners to learn representations for few-shot learning and show they offer better tradeoffs between feature size and performance across a range of few-shot recognition benchmarks. Our objective is to learn feature embeddings that generalize well under a linear classification rule for novel categories. To efficiently solve the objective, we exploit two properties of linear classifiers: implicit differentiation of the optimality conditions of the convex problem and the dual formulation of the optimization problem. This allows us to use high-dimensional embeddings with improved generalization at a modest increase in computational overhead. Our approach, named MetaOptNet, achieves state-of-the-art performance on miniImageNet, tieredImageNet, CIFAR-FS, and FC100 few-shot learning benchmarks. Our code is available on-line[1].

1. Introduction

The ability to learn from a few examples is a hallmark of human intelligence, yet it remains a challenge for modern machine learning systems. This problem has received significant attention from the machine learning community recently where few-shot learning is cast as a meta-learning problem (*e.g.*, [22, 8, 33, 28]). The goal is to minimize generalization error across a distribution of tasks with few training examples. Typically, these approaches are composed of an *embedding model* that maps the input domain into a feature space and a *base learner* that maps the feature space to task variables. The meta-learning objective is to learn an embedding model such that the base learner generalizes well across tasks.

While many choices for base learners exist, nearest-neighbor classifiers and their variants (*e.g.*, [28, 33]) are

popular as the classification rule is simple and the approach scales well in the low-data regime. However, discriminatively trained linear classifiers often outperform nearest-neighbor classifiers (*e.g.*, [4, 16]) in the low-data regime as they can exploit the negative examples which are often more abundant to learn better class boundaries. Moreover, they can effectively use high dimensional feature embeddings as model capacity can be controlled by appropriate regularization such as weight sparsity or norm.

Hence, in this paper, we investigate linear classifiers as the base learner for a meta-learning based approach for few-shot learning. The approach is illustrated in Figure 1 where a linear support vector machine (SVM) is used to learn a classifier given a set of labeled training examples and the generalization error is computed on a novel set of examples from the same task. The key challenge is computational since the meta-learning objective of minimizing the generalization error across tasks requires training a linear classifier in the inner loop of optimization (see Section 3). However, the objective of linear models is convex and can be solved efficiently. We observe that two additional properties arising from the convex nature that allows efficient meta-learning: implicit differentiation of the optimization [2, 11] and the low-rank nature of the classifier in the few-shot setting. The first property allows the use of off-the-shelf convex optimizers to estimate the optima and implicitly differentiate the optimality or Karush-Kuhn-Tucker (KKT) conditions to train embedding model. The second property means that the number of optimization variables in the *dual formation* is far smaller than the feature dimension for few-shot learning.

To this end, we have incorporated a *differentiable quadratic programming* (QP) solver [1] which allows end-to-end learning of the embedding model with various linear classifiers, *e.g.*, multiclass support vector machines (SVMs) [5] or linear regression, for few-shot classification tasks. Making use of these properties, we show that our method is practical and offers substantial gains over nearest neighbor classifiers at a modest increase in computational costs (see Table 3). Our method achieves state-of-the-art performance on 5-way 1-shot and 5-shot classification for popu-

---

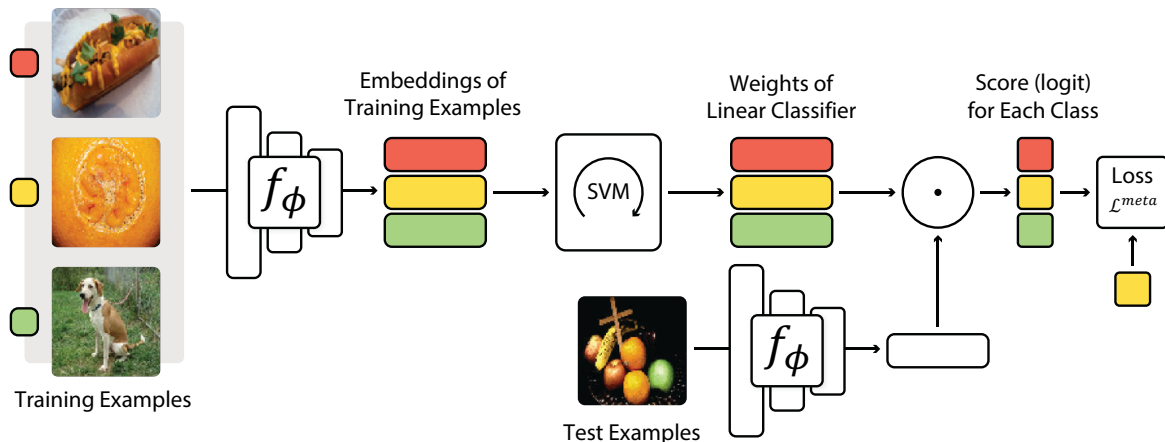[1]https://github.com/kjunelee/MetaOptNet

Figure 1. **Overview of our approach.** Schematic illustration of our method MetaOptNet on an 1-shot 3-way classification task. The meta-training objective is to learn the parameters ϕ of a feature embedding model f_ϕ that generalizes well across tasks when used with regularized linear classifiers (*e.g.*, SVMs). A task is a tuple of a few-shot training set and a test set (see Section 3 for details).

lar few-shot benchmarks including miniImageNet [33, 22], tieredImageNet [23], CIFAR-FS [3], and FC100 [20].

2. Related Work

Meta-learning studies what aspects of the learner (commonly referred to as *bias* or *prior*) effect generalization across a distribution of tasks [26, 31, 32]. Meta-learning approaches for few-shot learning can be broadly categorized these approaches into three groups. *Gradient-based methods* [22, 8] use gradient descent to adapt the embedding model parameters (*e.g.*, all layers of a deep network) given training examples. *Nearest-neighbor methods* [33, 28] learn a distance-based prediction rule over the embeddings. For example, prototypical networks [28] represent each class by the mean embedding of the examples, and the classification rule is based on the distance to the nearest class mean. Another example is matching networks [33] that learns a kernel density estimate of the class densities using the embeddings over training data (the model can also be interpreted as a form of *attention* over training examples). *Model-based methods* [18, 19] learn a parameterized predictor to estimate model parameters, *e.g.*, a recurrent network that predicts parameters analogous to a few steps of gradient descent in parameter space. While gradient-based methods are general, they are prone to overfitting as the embedding dimension grows [18, 25]. Nearest-neighbor approaches offer simplicity and scale well in the few-shot setting. However, nearest-neighbor methods have no mechanisms for feature selection and are not very robust to noisy features.

Our work is related to techniques for *backpropagation* though optimization procedures. Domke [6] presented a generic method based on *unrolling* gradient-descent for a fixed number of steps and automatic differentiation to compute gradients. However, the trace of the optimizer (*i.e.*,

the intermediate values) needs to be stored in order to compute the gradients which can be prohibitive for large problems. The storage overhead issue was considered in more detail by Maclaurin *et al.* [15] where they studied low precision representations of the optimization trace of deep networks. If the argmin of the optimization can be found analytically, such as in unconstrained quadratic minimization problems, then it is also possible to compute the gradients analytically. This has been applied for learning in low-level vision problems [30, 27]. A concurrent and closely related work [3] uses this idea to learn few-shot models using ridge-regression base learners which have closed-form solutions. We refer readers to Gould *et al.* [11] which provides an excellent survey of techniques for differentiating argmin and argmax problems.

Our approach advocates the use of linear classifiers which can be formulated as convex learning problems. In particular, the objective is a *quadratic program* (QP) which can be efficiently solved to obtain its *global optima* using gradient-based techniques. Moreover, the solution to convex problems can be characterized by their Karush-Kuhn-Tucker (KKT) conditions which allow us to *backpropagate* through the learner using the *implicit function theorem* [12]. Specifically, we use the formulation of Amos and Kolter [1] which provides efficient GPU routines for computing solutions to QPs and their gradients. While they applied this framework to learn representations for constraint satisfaction problems, it is also well-suited for few-shot learning as the problem sizes that arise are typically small.

While our experiments focus on linear classifiers with hinge loss and ℓ_2 regularization, our framework can be used with other loss functions and non-linear kernels. For example, the ridge regression learner used in [3] can be implemented within our framework allowing a direct comparison.

3. Meta-learning with Convex Base Learners

We first derive the meta-learning framework for few-shot learning following prior work (*e.g.*, [28, 22, 8]) and then discuss how convex base learners, such as linear SVMs, can be incorporated.

3.1. Problem formulation

Given the training set $\mathcal{D}^{train} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^T$, the goal of the base learner $\mathcal{A}$ is to estimate parameters θ of the predictor $y = f(\boldsymbol{x}; \theta)$ so that it generalizes well to the unseen test set $\mathcal{D}^{test} = \{(\boldsymbol{x}_t, y_t)\}_{t=1}^Q$. It is often assumed that the training and test set are sampled from the same distribution and the domain is mapped to a feature space using an embedding model f_ϕ parameterized by ϕ. For optimization-based learners, the parameters are obtained by minimizing the empirical loss over training data along with a regularization that encourages simpler models. This can be written as:

$$\theta = \mathcal{A}(\mathcal{D}^{train}; \phi) = \arg\min_\theta \mathcal{L}^{base}(\mathcal{D}^{train}; \theta, \phi) + \mathcal{R}(\theta) \tag{1}$$

where $\mathcal{L}^{base}$ is a loss function, such as the negative log-likelihood of labels, and $\mathcal{R}(\theta)$ is a regularization term. Regularization plays an important role in generalization when training data is limited.

Meta-learning approaches for few-shot learning aim to minimize the generalization error across a distribution of tasks sampled from a task distribution. Concretely, this can be thought of as learning over a collection of tasks: $\mathcal{T} = \{(\mathcal{D}_i^{train}, \mathcal{D}_i^{test})\}_{i=1}^I$, often referred to as a *meta-training* set. The tuple $(\mathcal{D}_i^{train}, \mathcal{D}_i^{test})$ describes a training and a test dataset, or a task. The objective is to learn an embedding model ϕ that minimizes generalization (or test) error across tasks given a base learner $\mathcal{A}$. Formally, the learning objective is:

$$\min_\phi \mathbb{E}_\mathcal{T} \left[\mathcal{L}^{meta}(\mathcal{D}^{test}; \theta, \phi), \text{where } \theta = \mathcal{A}(\mathcal{D}^{train}; \phi) \right]. \tag{2}$$

Figure 1 illustrates the training and testing for a single task. Once the embedding model f_ϕ is learned, its generalization is estimated on a set of *held-out* tasks (often referred to as a *meta-test* set) $\mathcal{S} = \{(\mathcal{D}_j^{train}, \mathcal{D}_j^{test})\}_{j=1}^J$ computed as:

$$\mathbb{E}_\mathcal{S} \left[\mathcal{L}^{meta}(\mathcal{D}^{test}; \theta, \phi), \text{where } \theta = \mathcal{A}(\mathcal{D}^{train}; \phi) \right]. \tag{3}$$

Following prior work [22, 8], we call the stages of estimating the expectation in Equation 2 and 3 as meta-training and meta-testing respectively. During meta-training, we keep an additional held-out *meta-validation* set to choose the hyper-parameters of the meta-learner and pick the best embedding model.

3.2. Episodic sampling of tasks

Standard few-shot learning benchmarks such as miniImageNet [22] evaluate models in K-way, N-shot classification tasks. Here K denotes the number of classes, and N denotes the number of training examples per class. Few-shot learning techniques are evaluated for small values of N, typically $N \in \{1, 5\}$. In practice, these datasets do not explicitly contain tuples $(\mathcal{D}_i^{train}, \mathcal{D}_i^{test})$, but each task for meta-learning is constructed "on the fly" during the meta-training stage, commonly described as an episode.

For example, in prior work [33, 22], a task (or episode) $\mathcal{T}_i = (\mathcal{D}_i^{train}, \mathcal{D}_i^{test})$ is sampled as follows. The overall set of categories is C^{train}. For each episode, categories C_i containing K categories from the C^{train} are first sampled (with replacement); then training (support) set $\mathcal{D}_i^{train} = \{(\boldsymbol{x}_n, y_n) \mid n = 1, \ldots, N \times K, y_n \in C_i\}$ consisting of N images per category is sampled; and finally, the test (query) set $\mathcal{D}_i^{test} = \{(\boldsymbol{x}_n, y_n) \mid n = 1, \ldots, Q \times K, y_n \in C_i\}$ consisting of Q images per category is sampled.

We emphasize that we need to sample without replacement, *i.e.*, $\mathcal{D}_i^{train} \cap \mathcal{D}_i^{test} = \varnothing$, to optimize the generalization error. In the same manner, meta-validation set and meta-test set are constructed on the fly from C^{val} and C^{test}, respectively. In order to measure the embedding model's generalization to *unseen categories*, C^{train}, C^{val}, and C^{test} are chosen to be mutually disjoint.

3.3. Convex base learners

The choice of the base learner $\mathcal{A}$ has a significant impact on Equation 2. The base learner that computes $\theta = \mathcal{A}(D^{train}; \phi)$ has to be efficient since the expectation has to be computed over a distribution of tasks. Moreover, to estimate parameters ϕ of the embedding model the gradients of the task test error $\mathcal{L}^{meta}(\mathcal{D}^{test}; \theta, \phi)$ with respect to ϕ have to be efficiently computed. This has motivated simple base learners such as nearest class mean [28] for which the parameters of the base learner θ are easy to compute and the objective is differentiable.

We consider base learners based on multi-class linear classifiers (*e.g.*, support vector machines (SVMs) [5, 34], logistic regression, and ridge regression) where the base-learner's objective is *convex*. For example, a K-class linear SVM can be written as $\theta = \{\boldsymbol{w}_k\}_{k=1}^K$. The Crammer and Singer [5] formulation of the multi-class SVM is:

$$\theta = \mathcal{A}(\mathcal{D}^{train}; \boldsymbol{\phi}) = \arg\min_{\{\boldsymbol{w}_k\}} \min_{\{\xi_i\}} \frac{1}{2} \sum_k ||\boldsymbol{w}_k||_2^2 + C \sum_n \xi_n$$

subject to

$$\boldsymbol{w}_{y_n} \cdot f_{\boldsymbol{\phi}}(\boldsymbol{x}_n) - \boldsymbol{w}_k \cdot f_{\boldsymbol{\phi}}(\boldsymbol{x}_n) \geq 1 - \delta_{y_n, k} - \xi_n, \ \forall n, k \tag{4}$$

where $\mathcal{D}^{train} = \{(\boldsymbol{x}_n, y_n)\}$, C is the regularization parameter and $\delta_{\cdot, \cdot}$ is the Kronecker delta function.

Gradients of the SVM objective. From Figure 1, we see that in order to make our system end-to-end trainable, we require that the solution of the SVM solver should be differentiable with respect to its input, *i.e.*, we should be able to compute $\{\frac{\partial \theta}{\partial f_\phi(\boldsymbol{x}_n)}\}_{n=1}^{N \times K}$. The objective of SVM is convex and has a unique optimum. This allows for the use of implicit function theorem (*e.g.*, [12, 7, 2]) on the optimality (KKT) conditions to obtain the necessary gradients. For the sake of completeness, we derive the form of the theorem for convex optimization problems as stated in [2]. Consider the following convex optimization problem:

$$
\begin{aligned}
\text{minimize} \quad & f_0(\theta, z) \\
\text{subject to} \quad & f(\theta, z) \preceq 0 \\
& h(\theta, z) = 0.
\end{aligned} \tag{5}
$$

where the vector $\theta \in \mathbb{R}^d$ is the optimization variable of the problem, the vector $z \in \mathbb{R}^e$ is the input parameter of the optimization problem, which is $\{f_\phi(\boldsymbol{x}_n)\}$ in our case. We can optimize the objective by solving for the saddle point $(\tilde{\theta}, \tilde{\lambda}, \tilde{\nu})$ of the following Lagrangian:

$$
L(\theta, \lambda, \nu, z) = f_0(\theta, z) + \lambda^T f(\theta, z) + \nu^T h(\theta, z). \tag{6}
$$

In other words, we can obtain the optimum of the objective function by solving $g(\tilde{\theta}, \tilde{\lambda}, \tilde{\nu}, z) = 0$ where

$$
g(\theta, \lambda, \nu, z) = \begin{bmatrix} \nabla_\theta L(\theta, \lambda, \nu, z) \\ \mathbf{diag}(\lambda) f(\theta, z) \\ h(\theta, z) \end{bmatrix}. \tag{7}
$$

Given a function $f(x) : \mathbb{R}^n \to \mathbb{R}^m$, denote $D_x f(x)$ as its Jacobian $\in \mathbb{R}^{m \times n}$.

Theorem 1 *(From Barratt [2]) Suppose $g(\tilde{\theta}, \tilde{\lambda}, \tilde{\nu}, z) = 0$. Then, when all derivatives exist,*

$$
D_z \tilde{\theta} = -D_\theta g(\tilde{\theta}, \tilde{\lambda}, \tilde{\nu}, z)^{-1} D_z g(\tilde{\theta}, \tilde{\lambda}, \tilde{\nu}, z). \tag{8}
$$

This result is obtained by applying the implicit function theorem to the KKT conditions. Thus, once we compute the optimal solution $\tilde{\theta}$, we can obtain a closed-form expression for the gradient of $\tilde{\theta}$ with respect to the input data. This obviates the need for backpropagating through the entire optimization trajectory since the solution does not depend on the trajectory or initialization due to its uniqueness. This also saves memory, an advantage that convex problems have over generic optimization problems.

Time complexity. The forward pass (*i.e.*, computation of Equation 4) using our approach requires the solution to the QP solver whose complexity scales as $O(d^3)$ where d is the number of optimization variables. This time is dominated by factorizing the KKT matrix required for primal-dual interior point method. Backward pass requires the solution to Equation 8 in Theorem 1, whose complexity is

$O(d^2)$ given the factorization already computed in the forward pass. Both forward pass and backward pass can be expensive when the dimension of embedding f_ϕ is large.

Dual formulation. The dual formulation of the objective in Equation 4 allows us to address the poor dependence on the embedding dimension and can be written as follows. Let

$$
\boldsymbol{w}_k(\boldsymbol{\alpha}^k) = \sum_n \alpha_n^k f_\phi(\boldsymbol{x}_n) \quad \forall k. \tag{9}
$$

We can instead optimize in the dual space:

$$
\max_{\{\boldsymbol{\alpha}^k\}} \left[-\frac{1}{2} \sum_k ||\boldsymbol{w}_k(\boldsymbol{\alpha}^k)||_2^2 + \sum_n \alpha_n^{y_n} \right]
$$

subject to

$$
\begin{aligned}
& \alpha_n^{y_n} \leq C, \quad \alpha_n^k \leq 0 \quad \forall k \neq y_n, \\
& \sum_k \alpha_n^k = 0 \quad \forall n.
\end{aligned} \tag{10}
$$

This results in a quadratic program (QP) over the dual variables $\{\boldsymbol{\alpha}^k\}_{k=1}^K$. We note that the size of the optimization variable is the number of training examples times the number of classes. This is often *much smaller* than the size of the feature dimension for few-shot learning. We solve the dual QP of Equation 10 using [1] which implements a differentiable GPU-based QP solver. In practice (as seen in Table 3) the time taken by the QP solver is comparable to the time taken to compute features using the ResNet-12 architectures so the overall speed per iteration is not significantly different from those based on simple base learners such as nearest class prototype (mean) used in Prototypical Networks [28].

Concurrent to our work, Bertinetto *et al.* [3] employed ridge regression as the base learner which has a closed-form solution. Although ridge regression may not be best suited for classification problems, their work showed that training models by minimizing squared error with respect to one-hot labels works well in practice. The resulting optimization for ridge regression is also a QP and can be implemented within our framework as:

$$
\max_{\{\boldsymbol{\alpha}^k\}} \left[-\frac{1}{2} \sum_k ||\boldsymbol{w}_k(\boldsymbol{\alpha}^k)||_2^2 - \frac{\lambda}{2} \sum_k ||\boldsymbol{\alpha}^k||_2^2 + \sum_n \alpha_n^{y_n} \right] \tag{11}
$$

where $\boldsymbol{w}_k$ is defined as Equation 9. A comparison of linear SVM and ridge regression in Section 4 shows a slight advantage of the linear SVM formation.

3.4. Meta-learning objective

To measure the performance of the model we evaluate the negative log-likelihood of the test data sampled from

the same task. Hence, we can re-express the meta-learning objective of Equation 2 as:

$$\mathcal{L}^{meta}(\mathcal{D}^{test}; \theta, \phi, \gamma) =$$
$$\sum_{(\boldsymbol{x},y) \in \mathcal{D}^{test}} [-\gamma \boldsymbol{w}_y \cdot f_\phi(\boldsymbol{x}) + \log \sum_k \exp(\gamma \boldsymbol{w}_k \cdot f_\phi(\boldsymbol{x}))] \tag{12}$$

where $\theta = \mathcal{A}(\mathcal{D}^{train}; \phi) = \{\boldsymbol{w}_k\}_{k=1}^{K}$ and γ is a learnable scale parameter. Prior work in few-shot learning [20, 3, 10] suggest that adjusting the prediction score by a learnable scale parameter γ provides better performance under nearest class mean and ridge regression base learners.

We empirically find that inserting γ is beneficial for the meta-learning with SVM base learner as well. While other choices of test loss, such as hinge loss, are possible, log-likelihood worked the best in our experiments.

4. Experiments

We first describe the network architecture and optimization details used in our experiments (Section 4.1). We then present results on standard few-shot classification benchmarks including derivatives of ImageNet (Section 4.2) and CIFAR (Section 4.3), followed by a detailed analysis of the impact of various base learners on accuracy and speed using the same embedding network and training setup (Section 4.4-4.6).

4.1. Implementation details

Meta-learning setup. We use a ResNet-12 network following [20, 18] in our experiments. Let Rk denote a residual block that consists of three {3×3 convolution with k filters, batch normalization, Leaky ReLU(0.1)}; let MP denote a 2×2 max pooling. We use DropBlock regularization [9], a form of structured Dropout. Let DB(k, b) denote a DropBlock layer with keep_rate=k and block_size=b. The network architecture for ImageNet derivatives is: R64-MP-DB(0.9,1)-R160-MP-DB(0.9,1)-R320-MP-DB(0.9,5)-R640-MP-DB(0.9,5), while the network architecture used for CIFAR derivatives is: R64-MP-DB(0.9,1)-R160-MP-DB(0.9,1)-R320-MP-DB(0.9,2)-R640-MP-DB(0.9,2). We do not apply a global average pooling after the last residual block.

As an optimizer, we use SGD with Nesterov momentum of 0.9 and weight decay of 0.0005. Each mini-batch consists of 8 episodes. The model was meta-trained for 60 epochs, with each epoch consisting of 1000 episodes. The learning rate was initially set to 0.1, and then changed to 0.006, 0.0012, and 0.00024 at epochs 20, 40 and 50, respectively, following the practice of [10].

During meta-training, we adopt horizontal flip, random crop, and color (brightness, contrast, and saturation) jitter data augmentation as in [10, 21]. For experiments on mini-ImageNet with ResNet-12, we use label smoothing with $\epsilon = 0.1$. Unlike [28] where they used higher way classification for meta-training than meta-testing, we use a 5-way classification in both stages following recent works [10, 20]. Each class contains 6 test (query) samples during meta-training and 15 test samples during meta-testing. Our meta-trained model was chosen based on 5-way 5-shot test accuracy on the meta-validation set.

Meta-training shot. For prototypical networks, we match the meta-training shot to meta-testing shot following the usual practice [28, 10]. For SVM and ridge regression, we observe that keeping meta-training shot higher than meta-testing shot leads to better test accuracies as shown in Figure 2. Hence, during meta-training, we set training shot to 15 for miniImageNet with ResNet-12; 5 for miniImageNet with 4-layer CNN (in Table 3); 10 for tieredImageNet; 5 for CIFAR-FS; and 15 for FC100.

Base-learner setup. For linear classifier training, we use the quadratic programming (QP) solver OptNet [1]. Regularization parameter C of SVM was set to 0.1. Regularization parameter λ of ridge regression was set to 50.0. For the nearest class mean (prototypical networks), we use squared Euclidean distance normalized with respect to the feature dimension.

Early stopping. Although we can run the optimizer until convergence, in practice we found that running the QP solver for a fixed number of iterations (just three) works well in practice. Early stopping acts an additional regularizer and even leads to a slightly better performance.

4.2. Experiments on ImageNet derivatives

The **miniImageNet** dataset [33] is a standard benchmark for few-shot image classification benchmark, consisting of 100 randomly chosen classes from ILSVRC-2012 [24]. These classes are randomly split into 64, 16 and 20 classes for meta-training, meta-validation, and meta-testing respectively. Each class contains 600 images of size 84×84. Since the class splits were not released in the original publication [33], we use the commonly-used split proposed in [22].

The **tieredImageNet** benchmark [23] is a larger subset of ILSVRC-2012 [24], composed of 608 classes grouped into 34 high-level categories. These are divided into 20 categories for meta-training, 6 categories for meta-validation, and 8 categories for meta-testing. This corresponds to 351, 97 and 160 classes for meta-training, meta-validation, and meta-testing respectively. This dataset aims to minimize the semantic similarity between the splits. All images are of size 84×84.

Results. Table 1 summarizes the results on the 5-way mini-ImageNet and tieredImageNet. Our method achieves state-of-the-art performance on 5-way miniImageNet and tiered-ImageNet benchmarks. Note that LEO [25] make use of encoder and relation network in addition to the WRN-28-10 backbone network to produce sample-dependent initializa-

Table 1. **Comparison to prior work on miniImageNet and tieredImageNet.** Average few-shot classification accuracies (%) with 95% confidence intervals on miniImageNet and tieredImageNet meta-test splits. a-b-c-d denotes a 4-layer convolutional network with a, b, c, and d filters in each layer. *Results from [22]. †Used the union of meta-training set and meta-validation set to meta-train the meta-learner. "RR" stands for ridge regression.

model	backbone	miniImageNet 5-way		tieredImageNet 5-way	
		1-shot	5-shot	1-shot	5-shot
Meta-Learning LSTM* [22]	64-64-64-64	43.44 ± 0.77	60.60 ± 0.71	-	-
Matching Networks* [33]	64-64-64-64	43.56 ± 0.84	55.31 ± 0.73	-	-
MAML [8]	32-32-32-32	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 1.75
Prototypical Networks*† [28]	64-64-64-64	49.42 ± 0.78	68.20 ± 0.66	53.31 ± 0.89	72.69 ± 0.74
Relation Networks* [29]	64-96-128-256	50.44 ± 0.82	65.32 ± 0.70	54.48 ± 0.93	71.32 ± 0.78
R2D2 [3]	96-192-384-512	51.2 ± 0.6	68.8 ± 0.1	-	-
Transductive Prop Nets [14]	64-64-64-64	55.51 ± 0.86	69.86 ± 0.65	59.91 ± 0.94	73.30 ± 0.75
SNAIL [18]	ResNet-12	55.71 ± 0.99	68.88 ± 0.92	-	-
Dynamic Few-shot [10]	64-64-128-128	56.20 ± 0.86	73.00 ± 0.64	-	-
AdaResNet [19]	ResNet-12	56.88 ± 0.62	71.94 ± 0.57	-	-
TADAM [20]	ResNet-12	58.50 ± 0.30	76.70 ± 0.30	-	-
Activation to Parameter† [21]	WRN-28-10	59.60 ± 0.41	73.74 ± 0.19	-	-
LEO† [25]	WRN-28-10	61.76 ± 0.08	77.59 ± 0.12	66.33 ± 0.05	81.44 ± 0.09
MetaOptNet-RR (ours)	ResNet-12	61.41 ± 0.61	77.88 ± 0.46	$\mathbf{65.36 \pm 0.71}$	$\mathbf{81.34 \pm 0.52}$
MetaOptNet-SVM (ours)	ResNet-12	62.64 ± 0.61	78.63 ± 0.46	$\mathbf{65.99 \pm 0.72}$	$\mathbf{81.56 \pm 0.53}$
MetaOptNet-SVM-trainval (ours)†	ResNet-12	$\mathbf{64.09 \pm 0.62}$	$\mathbf{80.00 \pm 0.45}$	65.81 ± 0.74	$\mathbf{81.75 \pm 0.53}$

tion of gradient descent. TADAM [20] employs a task embedding network (TEN) block for each convolutional layer – which predicts element-wise scale and shift vectors.

We also note that [25, 21] pretrain the WRN-28-10 feature extractor [36] to jointly classify all 64 classes in miniImageNet meta-training set; then freeze the network during the meta-training. [20] make use of a similar strategy of using standard classification: they co-train the feature embedding on few-shot classification task (5-way) and standard classification task (64-way). In contrast, our system is meta-trained end-to-end, explicitly training the feature extractor to work well on few-shot learning tasks with regularized linear classifiers. This strategy allows us to clearly see the effect of meta-learning. Our method is arguably simpler and achieves strong performance.

4.3. Experiments on CIFAR derivatives

The **CIFAR-FS** dataset [3] is a recently proposed few-shot image classification benchmark, consisting of all 100 classes from CIFAR-100 [13]. The classes are randomly split into 64, 16 and 20 for meta-training, meta-validation, and meta-testing respectively. Each class contains 600 images of size 32×32.

The **FC100** dataset [20] is another dataset derived from CIFAR-100 [13], containing 100 classes which are grouped into 20 superclasses. These classes are partitioned into 60 classes from 12 superclasses for meta-training, 20 classes from 4 superclasses for meta-validation, and 20 classes from 4 superclasses for meta-testing. The goal is to minimize semantic overlap between classes similar to the goal
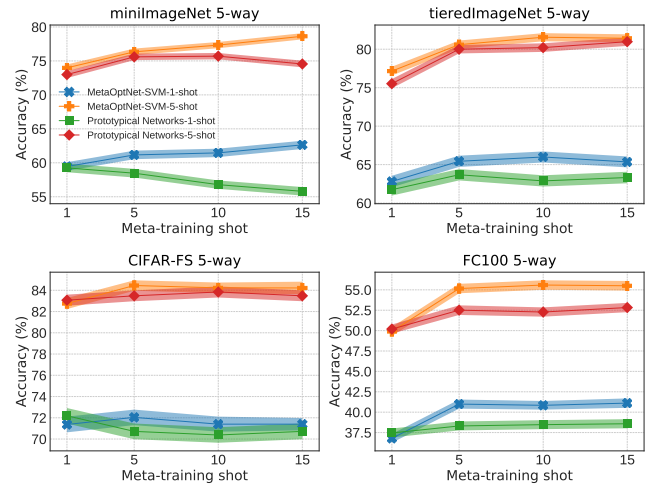

Figure 2. **Test accuracies (%) on miniImageNet meta-test set with varying meta-training shot**. The error bar denotes 95 % confidence interval. Ridge regression base learner (MetaOptNet-RR) converges in 1 iteration; SVM base learner (MetaOptNet-SVM) was run for 3 iterations.

of tieredImageNet. Each class contains 600 images of size 32×32.

Results. Table 2 summarizes the results on the 5-way classification tasks where our method MetaOptNet-SVM achieves the state-of-the-art performance. On the harder FC100 dataset, the gap between various base learners is more significant, which highlights the advantage of complex base learners in the few-shot learning setting.

Table 2. **Comparison to prior work on CIFAR-FS and FC100.** Average few-shot classification accuracies (%) with 95% confidence intervals on CIFAR-FS and FC100. a-b-c-d denotes a 4-layer convolutional network with a, b, c, and d filters in each layer. *CIFAR-FS results from [3]. †FC100 result from [20]. ¶Used the union of meta-training set and meta-validation set to meta-train the meta-learner. "RR" stands for ridge regression.

		CIFAR-FS 5-way		FC100 5-way	
model	backbone	1-shot	5-shot	1-shot	5-shot
MAML* [8]	32-32-32-32	58.9 ± 1.9	71.5 ± 1.0	-	-
Prototypical Networks*† [28]	64-64-64-64	55.5 ± 0.7	72.0 ± 0.6	35.3 ± 0.6	48.6 ± 0.6
Relation Networks* [29]	64-96-128-256	55.0 ± 1.0	69.3 ± 0.8	-	-
R2D2 [3]	96-192-384-512	65.3 ± 0.2	79.4 ± 0.1	-	-
TADAM [20]	ResNet-12	-	-	40.1 ± 0.4	56.1 ± 0.4
ProtoNets (our backbone) [28]	ResNet-12	$\mathbf{72.2 \pm 0.7}$	83.5 ± 0.5	37.5 ± 0.6	52.5 ± 0.6
MetaOptNet-RR (ours)	ResNet-12	$\mathbf{72.6 \pm 0.7}$	84.3 ± 0.5	40.5 ± 0.6	55.3 ± 0.6
MetaOptNet-SVM (ours)	ResNet-12	$\mathbf{72.0 \pm 0.7}$	84.2 ± 0.5	41.1 ± 0.6	55.5 ± 0.6
MetaOptNet-SVM-trainval (ours)¶	ResNet-12	$\mathbf{72.8 \pm 0.7}$	$\mathbf{85.0 \pm 0.5}$	$\mathbf{47.2 \pm 0.6}$	$\mathbf{62.5 \pm 0.6}$

Table 3. **Effect of the base learner and embedding network architecture.** Average few-shot classification accuracy (%) and forward inference time (ms) per episode on miniImageNet and tieredImageNet with varying base learner and backbone architecture. The former group of results used the standard 4-layer convolutional network with 64 filters per layer used in [33, 28], whereas the latter used a 12-layer ResNet without the global average pooling. "RR" stands for ridge regression.

	miniImageNet 5-way				tieredImageNet 5-way			
	1-shot		5-shot		1-shot		5-shot	
model	acc. (%)	time (ms)	acc. (%)	time (ms)	acc. (%)	time (ms)	acc. (%)	time (ms)
4-layer conv (feature dimension=1600)								
Prototypical Networks [17, 28]	$53.47_{\pm0.63}$	$6_{\pm0.01}$	$70.68_{\pm0.49}$	$7_{\pm0.02}$	$54.28_{\pm0.67}$	$6_{\pm0.03}$	$71.42_{\pm0.61}$	$7_{\pm0.02}$
MetaOptNet-RR (ours)	$53.23_{\pm0.59}$	$20_{\pm0.03}$	$69.51_{\pm0.48}$	$27_{\pm0.05}$	$54.63_{\pm0.67}$	$21_{\pm0.05}$	$72.11_{\pm0.59}$	$28_{\pm0.06}$
MetaOptNet-SVM (ours)	$52.87_{\pm0.57}$	$28_{\pm0.02}$	$68.76_{\pm0.48}$	$37_{\pm0.05}$	$54.71_{\pm0.67}$	$28_{\pm0.07}$	$71.79_{\pm0.59}$	$38_{\pm0.08}$
ResNet-12 (feature dimension=16000)								
Prototypical Networks [17, 28]	$59.25_{\pm0.64}$	$60_{\pm17}$	$75.60_{\pm0.48}$	$66_{\pm17}$	$61.74_{\pm0.77}$	$61_{\pm17}$	$80.00_{\pm0.55}$	$66_{\pm18}$
MetaOptNet-RR (ours)	$61.41_{\pm0.61}$	$68_{\pm17}$	$\mathbf{77.88_{\pm0.46}}$	$75_{\pm17}$	$\mathbf{65.36_{\pm0.71}}$	$69_{\pm17}$	$81.34_{\pm0.52}$	$77_{\pm17}$
MetaOptNet-SVM (ours)	$\mathbf{62.64_{\pm0.61}}$	$78_{\pm17}$	$\mathbf{78.63_{\pm0.46}}$	$89_{\pm17}$	$\mathbf{65.99_{\pm0.72}}$	$78_{\pm17}$	$\mathbf{81.56_{\pm0.53}}$	$90_{\pm17}$

4.4. Comparisons between base learners

Table 3 shows the results where we vary the base learner for two different embedding architectures. When we use a standard 4-layer convolutional network where the feature dimension is low (1600), we do not observe a substantial benefit of adopting discriminative classifiers for few-shot learning. Indeed, nearest class mean classifier [17] is proven to work well under a low-dimensional feature as shown in Prototypical Networks [28]. However, when the embedding dimensional is much higher (16000), SVMs yield better few-shot accuracy than other base learners. Thus, regularized linear classifiers provide robustness when high-dimensional features are available.

The added benefits come at a modest increase in com-putational cost. For ResNet-12, compared to nearest class mean classifier, the additional overhead is around 13% for the ridge regression base learner and around 30-50% for the SVM base learner. As seen in from Figure 2, the performance of our model on both 1-shot and 5-shot regimes generally increases with increasing meta-training shot. This makes the approach more practical as we can meta-train the embedding once with a high shot for all meta-testing shots.

As noted in the FC100 experiment, SVM base learner seems to be beneficial when the semantic overlap between test and train is smaller. We hypothesize that the class embeddings are more significantly more compact for training data than test data (*e.g.*, see [35]); hence flexibility in the base learner allows robustness to noisy embeddings and improves generalization.
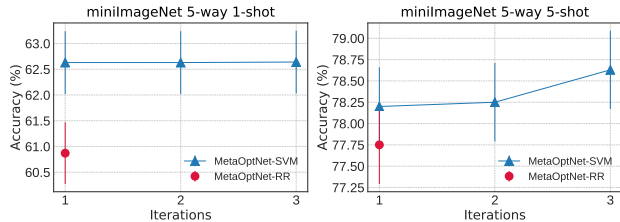
Figure 3. **Test accuracies (%) on miniImageNet meta-test set with varying iterations of QP solver.** The error bar denotes 95% confidence interval. "RR" stands for ridge regression.

4.5. Reducing meta-overfitting

Augmenting meta-training set. Despite sampling tasks, at the end of meta-training MetaOptNet-SVM with ResNet-12 achieves nearly 100% test accuracy on all the meta-training datasets except the tieredImageNet. To alleviate overfitting, similarly to [25, 21], we use the union of the meta-training and meta-validation sets to meta-train the embedding, keeping the hyperparameters, such as the number of epochs, identical to the previous setting. In particular, we terminate the meta-training after 21 epochs for mini-ImageNet, 52 epochs for tieredImageNet, 21 epochs for CIFAR-FS, and 21 epochs for FC100. Tables 1 and 2 show the results with the augmented meta-training sets, denoted as MetaOptNet-SVM-trainval. On minImageNet, CIFAR-FS, and FC100 datasets, we observe improvements in test accuracies. On tieredImageNet dataset, the difference is negligible. We suspect that this is because our system has not yet entered the regime of overfitting (In fact, we observe ~94% test accuracy on tieredImageNet meta-training set). Our results suggest that meta-learning embedding with more meta-training *"classes"* helps reduce overfitting to the meta-training set.

Various regularization techniques. Table 4 shows the effect of regularization methods on MetaOptNet-SVM with ResNet-12. We note that early works on few-shot learning [28, 8] did not employ any of these techniques. We observe that without the use of regularization, the performance of ResNet-12 reduces to the one of the 4-layer convolutional network with 64 filters per layer shown in Table 3. This shows the importance of regularization for meta-learners. We expect that performances of few-shot learning systems would be further improved by introducing novel regularization methods.

4.6. Efficiency of dual optimization

To see whether the dual optimization is indeed effective and efficient, we measure accuracies on meta-test set with varying iteration of the QP solver. Each iteration of QP solver [1] involves computing updates for primal and dual variables via LU decomposition of KKT matrix. The results are shown in Figure 3. The QP solver reaches the optima of ridge regression objective in just one iteration. Alternatively

Data Aug.	Weight Decay	Drop Block	Label Smt.	Larger Data	1-shot	5-shot
					51.13	70.88
✓					55.80	75.76
	✓				56.65	73.72
✓	✓				60.33	76.61
✓	✓	✓			61.11	77.40
✓	✓	✓	✓		62.64	78.63
✓	✓	✓	✓	✓	64.09	80.00

Table 4. **Ablation study.** Various regularization techniques improves test accuracy regularization techniques improves test accuracy (%) on 5-way miniImageNet benchmark. We use MetaOptNet-SVM with ResNet-12 for results. 'Data Aug.', 'Label Smt.', and 'Larger Data' stand for data augmentation, label smoothing on the meta-learning objective, and merged dataset of meta-training split and meta-test split, respectively.

one can use its closed-form solution as used in [3]. Also, we observe that for 1-shot tasks, the QP SVM solver reaches optimal accuracies in 1 iteration, although we observed that the KKT conditions are not exactly satisfied yet. For 5-shot tasks, even if we run QP SVM solver for 1 iteration, we achieve better accuracies than other base learners. When the iteration of SVM solver is limited to 1 iteration, 1 episode takes 69 ± 17 ms for an 1-shot task, and 80 ± 17 ms for a 5-shot task, which is on par with the computational cost of the ridge regression solver (Table 3). These experiments show that solving dual objectives for SVM and ridge regression is very effective under few-shot settings.

5. Conclusion

In this paper, we presented a meta-learning approach with convex base learners for few-shot learning. The dual formulation and KKT conditions can be exploited to enable computational and memory efficient meta-learning that is especially well-suited for few-shot learning problems. Linear classifiers offer better generalization than nearest-neighbor classifiers at a modest increase in computational costs (as seen in Table 3). Our experiments suggest that regularized linear models allow significantly higher embedding dimensions with reduced overfitting. For future work, we aim to explore other convex base-learners such as kernel SVMs. This would allow the ability to incrementally increase model capacity as more training data becomes available for a task.

References

[1] Brandon Amos and J. Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *ICML*, 2017. 1, 2, 4, 5, 8

[2] Shane Barratt. On the Differentiability of the Solution to Convex Optimization Problems. arXiv:1804.05098, 2018. 1, 4

[3] Luca Bertinetto, João F. Henriques, Philip H. S. Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR*, 2019. 2, 4, 5, 6, 7, 8

[4] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *ICML*, 2008. 1

[5] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, Mar. 2002. 1, 3

[6] Justin Domke. Generic methods for optimization-based modeling. In *AISTATS*, 2012. 2

[7] Asen L. Dontchev and R. Tyrrell Rockafellar. Implicit functions and solution mappings. *Springer Monogr. Math.*, 2009. 4

[8] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017. 1, 2, 3, 6, 7, 8

[9] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Dropblock: A regularization method for convolutional networks. In *NeurIPS*, 2018. 5

[10] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018. 5, 6

[11] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016. 1, 2

[12] Steven G. Krantz and Harold R. Parks. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2012. 2, 4

[13] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). 6

[14] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, and Yi Yang. Transductive propagation network for few-shot learning. In *ICLR*, 2019. 6

[15] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *ICML*, 2015. 2

[16] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *ICCV*, 2011. 1

[17] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriella Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(11):2624–2637, Nov. 2013. 7

[18] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018. 2, 5, 6

[19] Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *ICML*, 2018. 2, 6

[20] Boris N. Oreshkin, Pau Rodríguez, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018. 2, 5, 6, 7

[21] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L. Yuille. Few-shot image recognition by predicting parameters from activations. In *CVPR*, 2018. 5, 6, 8

[22] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017. 1, 2, 3, 5, 6

[23] Mengye Ren, Sachin Ravi, Eleni Triantafillou, Jake Snell, Kevin Swersky, Josh B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018. 2, 5

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, Dec. 2015. 5

[25] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019. 2, 5, 6, 8

[26] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987. 2

[27] Uwe Schmidt and Stefan Roth. Shrinkage fields for effective image restoration. In *CVPR*, 2014. 2

[28] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, 2017. 1, 2, 3, 4, 5, 6, 7, 8

[29] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018. 6, 7

[30] Marshall F. Tappen, Ce Liu, Edward H. Adelson, and William T. Freeman. Learning gaussian conditional random fields for low-level vision. In *CVPR*, 2007. 2

[31] Sebastian Thrun. *Lifelong Learning Algorithms*, pages 181–209. Springer US, Boston, MA, 1998. 2

[32] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, Jun 2002. 2

[33] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NIPS*, 2016. 1, 2, 3, 5, 6, 7

[34] Jason Weston and Chris Watkins. Support Vector Machines for Multiclass Pattern Recognition. In *European Symposium On Artificial Neural Networks*, 1999. 3

[35] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, 2014. 7

[36] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016. 6