Machine Vision and Applications (2023) 34:53
https://doi.org/10.1007/s00138-023-01403-4

ORIGINAL PAPER

The general framework for few-shot learning by kernel HyperNetworks

Marcin Sendera[1,4] · Marcin Przewięźlikowski[1,4,5] · Jan Miksa[1] · Mateusz Rajski[1] · Konrad Karanowski[2] · Maciej Zięba[2,3] · Jacek Tabor[1] · Przemysław Spurek[1]

Received: 17 March 2023 / Revised: 17 April 2023 / Accepted: 18 April 2023

Abstract

Few-shot models aim at making predictions using a minimal number of labeled examples from a given task. The main challenge in this area is the *one-shot* setting, where only one element represents each class. We propose the general framework for few-shot learning via kernel HyperNetworks—the fusion of kernels and hypernetwork paradigm. Firstly, we introduce the classical realization of this framework, dubbed HyperShot. Compared to reference approaches that apply a gradient-based adjustment of the parameters, our models aim to switch the classification module parameters depending on the task's embedding. In practice, we utilize a hypernetwork, which takes the aggregated information from support data and returns the classifier's parameters handcrafted for the considered problem. Moreover, we introduce the kernel-based representation of the support examples delivered to hypernetwork to create the parameters of the classification module. Consequently, we rely on relations between the support examples' embeddings instead of the backbone models' direct feature values. Thanks to this approach, our model can adapt to highly different tasks. While such a method obtains very good results, it is limited by typical problems such as poorly quantified uncertainty due to limited data size. We further show that incorporating Bayesian neural networks into our general framework, an approach we call BayesHyperShot, solves this issue.

Keywords Few-shot learning · Meta-learning · HyperNetworks · Kernel methods · Bayesian neural networks

1 Introduction

Current artificial intelligence techniques cannot rapidly generalize from a few examples. This common inability stems from the fact that most deep neural networks must be trained on large-scale data. In contrast, humans can learn new tasks quickly by utilizing what they learned in the past. Few-shot learning models try to fill this gap by *learning how to learn* from a limited number of examples. Few-shot learning is the problem of making predictions based on a few labeled examples. The goal of few-shot learning is not to recognize a fixed set of labels but to quickly adapt to new tasks with a small amount of training data. After training, the model can classify new data using only a few training examples.

Marcin Sendera and Marcin Przewięźlikowski have contributed equally to this work.

✉ Marcin Sendera
marcin.sendera@doctoral.uj.edu.pl

Marcin Przewięźlikowski
marcin.przewiezlikowski@doctoral.uj.edu.pl

Jan Miksa
jan.miksa@student.uj.edu.pl

Mateusz Rajski
mateusz.rajski@student.uj.edu.pl

Konrad Karanowski
254533@student.pwr.edu.pl

Maciej Zięba
maciej.zieba@pwr.edu.pl

Jacek Tabor
jacek.tabor@uj.edu.pl

Przemysław Spurek
przemyslaw.spurek@uj.edu.pl

[1] Faculty of Mathematics and Computer Science, Jagiellonian University, Lojasiewicza 6, Kraków, Poland

[2] Department of Artificial Intelligence, Wroclaw University of Science and Technology, Wyb. Wyspianskiego 27, Wrocław, Poland

[3] Tooploox, Wrocław, Poland

[4] Doctoral School of Exact and Natural Sciences, Jagiellonian University, Lojasiewicza 11, Kraków, Poland

[5] IDEAS NCBR, Chmielna 69, Warsaw, Poland

Published online: 20 May 2023

Two novel few-shot learning techniques have recently emerged. The first is based on the kernel methods and Gaussian processes [1–3]. The universal deep kernel has enough data to generalize well to unseen tasks without overfitting. The second technique makes use of the Hypernetworks [4–9], which allow to aggregate information from the support set and produce dedicated network weights for new tasks.

The above approaches give promising results but also have some limitations. Kernel-based methods are not flexible enough, since they use Gaussian processes on top of the models. Moreover, it is not trivial to use Gaussian processes for classification tasks. On the other hand, Hypernetworks must aggregate information from the support set, and it is hard to model the relation between classes as opposed to classical feature extraction.

This paper introduces a general framework that combines the Hypernetworks paradigm with kernel methods to realize a new strategy that mimics the human way of learning. First, we examine the entire support set and extract the information in order to distinguish objects of each class. Then, based on the relations between their features, we create the decision rules.

Kernel methods realize the first part of the process. For each of the few-shot tasks, we extract the features from the support set through the backbone architecture and calculate kernel values between them. Then we use a Hypernetwork architecture [3, 4]—a neural network that takes kernel representation and produces decision rules in the form of a classifier. In our framework, the Hypernetwork aggregates the information from the support set and produces weights or adaptation parameters of the target model dedicated to the specific task, classifying the query set.

The models we propose inherit the flexibility from Hypernetworks and the ability to learn the relation between objects from kernel-based methods.

We consider two alternative approaches to the realization of our framework—the classical one, which we dubbed HyperShot, and the Bayesian version called BayesHyperShot. Firstly, we started with the classical approach.

We perform an extensive experimental study of HyperShot by benchmarking it on various one-shot and few-shot image classification tasks. We find that HyperShot demonstrates high accuracy in all tasks, performing comparably or better than the other recently proposed methods. Moreover, HyperShot shows a strong ability to generalize, as evidenced by its performance on cross-domain classification tasks.

Unfortunately HyperShot, similar to other few-shot algorithms, suffers from limited data size, which may result in drawbacks such as poorly quantified uncertainty. To solve this problem, we extend our method by incorporating Bayesian neural network and Hypernetwork paradigms, which results in a Bayesian version of our general framework —BayesHyperShot. In practice, hypernetwork produces parameters of probability distribution on weights. In this paper, we consider Gaussian prior, but our framework can be used for modeling even more complex priors.

The contributions of this work are fourfold:

- In this paper, we propose a general framework that realizes the *learn how to learn* paradigm by modeling learning rules which are not based on gradient optimization and can produce completely different decision strategies.
- We propose a new approach to solve the few-shot learning problem by aggregating information from the support set by kernel methods and directly producing weights from the neural network dedicated to the query set.
- We propose HyperShot model, which combines the Hypernetworks paradigm with kernel methods classically, to produce the weights dedicated for each task.
- We show that our model can be generalized to Bayesian version BayesHyperShot, which allows for solving problems with poorly quantified uncertainty.

2 Hypernetworks for few-shot learning–a general framework

In this section, we present our general framework for utilizing the Hypernetworks and kernel-based methods for few-shot learning. In the beginning, we give a quick recap of the necessary background. Then, we introduce our framework by presenting the HyperShot. Finally, we show how to incorporate the Bayesian approach in this general framework and present the BayesHyperShot.

2.1 Background

Few-shot learning The terminology describing the few-shot learning setup is dispersive due to the colliding definitions used in the literature. For a unified taxonomy, we refer the reader to [10, 11]. Here, we use the nomenclature derived from the meta-learning literature, which is the most prevalent at the time of writing. Let:

$$\mathcal{S} = \{(\mathbf{x}_l, \mathbf{y}_l)\}_{l=1}^{L} \tag{1}$$

be a support set containing input–output pairs, with L examples with the equal class distribution. In the *one-shot* scenario, each class is represented by a single example, and $L = K$, where K is the number of the considered classes in the given task. For *few-shot* scenarios, each class usually has from 2 to 5 representatives in the support set $\mathcal{S}$. Let:

$$\mathcal{Q} = \{(\mathbf{x}_m, \mathbf{y}_m)\}_{m=1}^{M} \tag{2}$$

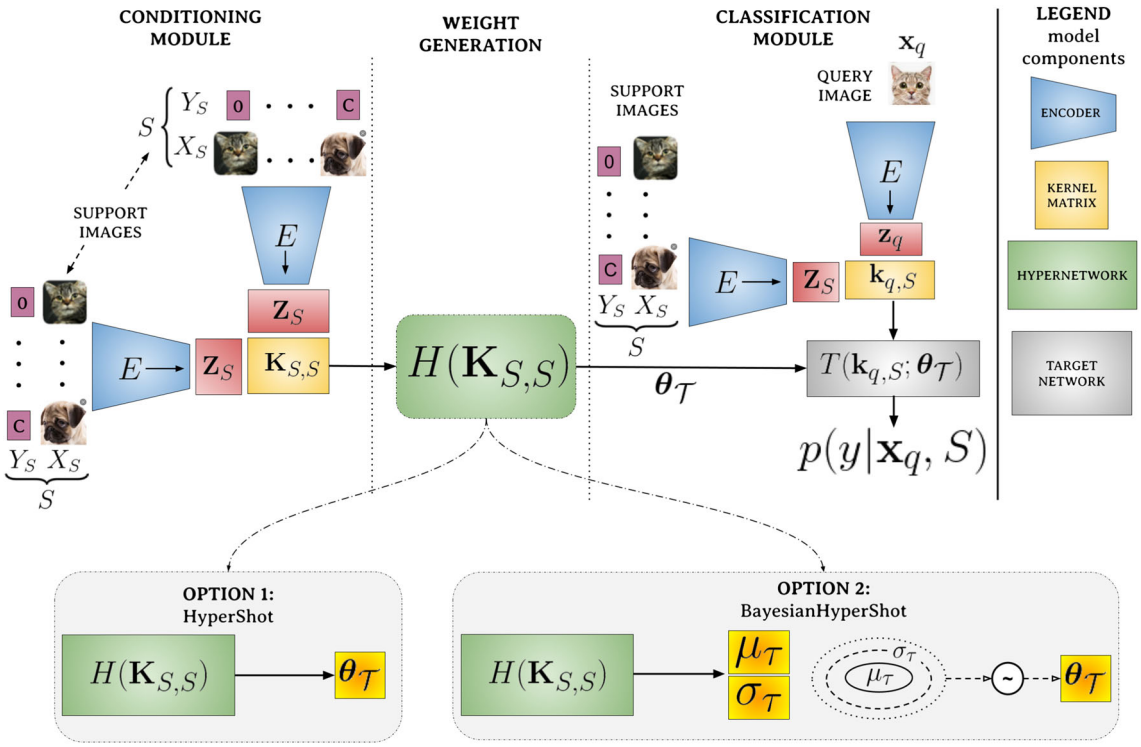

Fig. 1 General architecture of our framework. First, the examples from a support set are sorted according to the corresponding class labels and transformed by encoding network $E(\cdot)$ to obtain the matrix of ordered embeddings of the support examples, $\mathbf{Z}_S$. The low-dimensional representations stored in $\mathbf{Z}_S$ are further used to compute kernel matrix $\mathbf{K}_{S,S}$. The values of the kernel matrix are passed to the hypernetwork $H(\cdot)$ that creates the parameters $\boldsymbol{\theta}_\tau$ for the target classification module $T(\cdot)$. We identify two options for generating $\boldsymbol{\theta}_\tau$: (**i**) HyperShot—hypernetwork H generates $\boldsymbol{\theta}_\tau$ straightforwardly; (**ii**) BayesHyperShot—H generates parameters μ_τ and σ_τ of a Gaussian distribution, from which $\boldsymbol{\theta}_\tau$ can then be sampled. The query image $\mathbf{x}$ is processed by encoder $E(\cdot)$, and the vector of kernel values $\mathbf{k}_{\mathbf{x},S}$ is calculated between query embedding $\mathbf{z}_\mathbf{x}$ and the corresponding representations of support examples, $\mathbf{Z}_S$. The kernel vector $\mathbf{k}_{\mathbf{x},S}$ is further passed to target model $T(\cdot)$ to obtain the probability distribution for the considered classes

be a query set (sometimes referred to in the literature as a target set), with M examples, where M is typically one order of magnitude greater than K. For ease of notation, the support and query sets are grouped in a task $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$. During the training stage, the models for few-shot applications are fed by randomly selected examples from training set $\mathcal{D} = \{\mathcal{T}_n\}_{n=1}^N$, defined as a collection of such tasks.

During the inference stage, we consider task $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{X}_*\}$, where $\mathcal{S}_*$ is a support set with the known class values for a given task, and $\mathcal{X}_*$ is a set of query (unlabeled) inputs. The goal is to predict the class labels for query inputs $\mathbf{x} \in \mathcal{X}_*$, assuming support set $\mathcal{S}_*$ and using the model trained on $\mathcal{D}$.

Hypernetwork In the canonical work [4], hypernetworks are defined as neural models that generate weights for a separate target network solving a specific task. The authors aim to reduce the number of trainable parameters by designing a hypernetwork with a smaller number of parameters than the target network. Making an analogy between hypernetworks and generative models, the authors of [12] use this mechanism to generate a diverse set of target networks approximating the same function.

2.2 HyperShot overview

We introduce HyperShot—a model that utilizes hypernetworks for few-shot problems. The main idea of the proposed approach is to predict the values of the parameters for a classification network that makes predictions on the query images given the information extracted from support examples for a given task. Thanks to this approach, we can switch the classifier's parameters between completely different tasks based on the support set. The information about the current task is extracted from the support set using a parameterized kernel function that operates on embedding space. Thanks to this approach, we use relations among the support examples instead of taking the direct values of the embedding values as an input to the hypernetwork. Consequently, this approach is robust to the embedding values for new tasks far from the feature regions observed during training. The classification of the query image is also performed using the kernel values calculated with respect to the support set.

The architecture of HyperShot is provided in Fig. 1. We aim to predict the class distribution $p(\mathbf{y} \mid S, \mathbf{x})$, given a query image $\mathbf{x}$ and set of support examples $S = \{(\mathbf{x}_l, y_l)\}_{l=1}^K$.

First, all images from the support set are grouped by their corresponding class values. Next, each of the images $\mathbf{x}_l$ from the support set is transformed using encoding network $E(\cdot)$, which creates low-dimensional representations of the images, $E(\mathbf{x}_l) = \mathbf{z}_l$. The constructed embeddings are sorted according to class labels and stored in the matrix $\mathbf{Z}_S = [\mathbf{z}_{\pi(1)}, \ldots, \mathbf{z}_{\pi(K)}]^{\mathrm{T}}$, where $\pi(\cdot)$ is the bijective function, that satisfies $y_{\pi(l)} \leq y_{\pi(k)}$ for $l \leq k$.

In the next step, we calculate the kernel matrix $\mathbf{K}_{S,S}$, for vector pairs stored in rows of $\mathbf{Z}_S$. To achieve this, we use the parametrized kernel function $k(\cdot, \cdot)$, and calculate $k_{i,j}$ element of matrix $\mathbf{K}_{S,S}$ in the following way:

$$k_{i,j} = k(\mathbf{z}_{\pi(i)}, \mathbf{z}_{\pi(j)}). \tag{3}$$

The kernel matrix $\mathbf{K}_{S,S}$ represents the extracted information about the relations between support examples for a given task. The matrix $\mathbf{K}_{S,S}$ is further reshaped to the vector format and delivered to the input of the hypernetwork $H(\cdot)$. The role of the hypernetwork is to provide the parameters $\boldsymbol{\theta}_T$ of target model $T(\cdot)$ responsible for the classification of the query object. Thanks to that approach, we can switch between the parameters for entirely different tasks without moving via the gradient-controlled trajectory, like in some reference approaches like MAML.

We base the architecture of the Hypernetwork on a multilayer perceptron (MLP). Specifically, the HyperNetwork first processes the kernel matrix $\mathbf{K}_{S,S}$ through an MLP (dubbed the "neck"), whose output is then processed by "heads"—separate MLPs dedicated to producing the weights of each target network layer—see Fig. 4 for a visualization. We summarize the parameters of Hypernetworks and target networks used in our experiments in Table 8.

The query image $\mathbf{x}$ is classified in the following manner. First, the input image is transformed to low-dimensional feature representation $z_{\mathbf{x}}$ by encoder $E(\mathbf{x})$. Further, the kernel vector $\mathbf{k}_{\mathbf{x},S}$ between the query embedding and sorted support vectors $\mathbf{Z}_S$ is calculated in the following way:

$$\mathbf{k}_{\mathbf{x},S} = [k(\mathbf{z}_{\mathbf{x}}, \mathbf{z}_{\pi(1)}), \ldots, k(\mathbf{z}_{\mathbf{x}}, \mathbf{z}_{\pi(K)})]^{\mathrm{T}}. \tag{4}$$

The vector $\mathbf{k}_{\mathbf{x},S}$ is further provided on the input of target model $T(\cdot)$ that is using the parameters $\boldsymbol{\theta}_T$ returned by hypernetwork $H(\cdot)$. The target model returns the probability distribution $p(\mathbf{y} \mid S, \mathbf{x})$ for each class considered in the task.

The function $\pi(\cdot)$ enforces some ordering of the input delivered to $T(\cdot)$. Practically, any other permutation of the classes for the input vector $\mathbf{k}_{\mathbf{x},S}$. In such a case, the same permutation should be applied to rows and columns of $\mathbf{K}_{S,S}$. As a consequence, the hypernetwork is able to produce the dedicated target parameters for each of the possible permutations. Although this approach does not guarantee the permutation invariance for real-life scenarios, thanks to dedicated parameters for any ordering of the input, it should be satisfied for major cases.

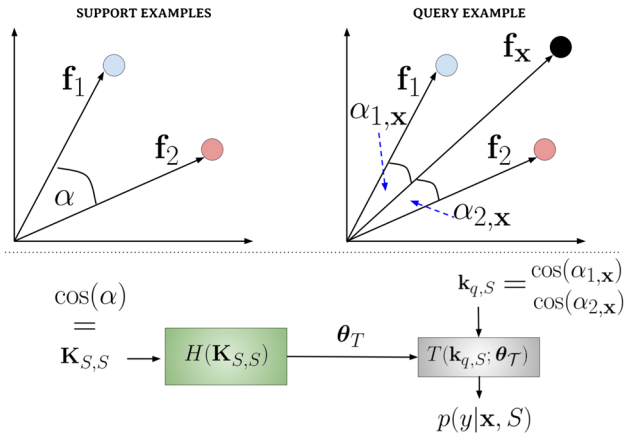

Fig. 2 Simple 2D example illustrating the application of cosine kernel for HyperShot. We consider the two support examples from different classes represented by vectors $\mathbf{f_1}$ and $\mathbf{f_2}$. For this simple scenario, the input of hypernetwork is represented simply by the cosine of α, which is an angle between vectors $\mathbf{f}_1$ and $\mathbf{f}_2$. We aim at classifying the query example $\mathbf{x}$ represented by a vector $\mathbf{f}_{\mathbf{x}}$. Considering our approach, we deliver to the target network $T(\cdot)$ the cosine values of angles between first ($\alpha_{\mathbf{x},1}$) and second ($\alpha_{\mathbf{x},2}$) support vectors and classify the query example using the weights $\boldsymbol{\theta}_T$ created by hypernetwork $H(\cdot)$ from $\cos\alpha$ (remaining components on the diagonal of $\mathbf{K}_{S,S}$ are constant for cosine kernel)

2.3 Kernel function

One of the key components of our approach is a kernel function $k(\cdot, \cdot)$. In this work, we consider the dot product of the transformed vectors given by:

$$k(\mathbf{z}_1, \mathbf{z}_2) = \mathbf{f}(\mathbf{z}_1)^{\mathrm{T}} \mathbf{f}(\mathbf{z}_2), \tag{5}$$

where $\mathbf{f}(\cdot)$ can be a parametrized transformation function, represented by MLP model, or simply an identity operation, $\mathbf{f}(\mathbf{z}) = \mathbf{z}$. In Euclidean space, this criterion can be expressed as $k(\mathbf{z}_1, \mathbf{z}_2) = ||\mathbf{f}(\mathbf{z}_1)|| \cdot ||\mathbf{f}(\mathbf{z}_2)|| \cos\alpha$, where α is an angle between vectors $\mathbf{f}(\mathbf{z}_1)$ and $\mathbf{f}(\mathbf{z}_2)$. The main feature of this function is that it considers the vectors' norms, which can be problematic for some tasks that are outliers regarding the representations created by $\mathbf{f}(\cdot)$. Therefore, we consider in our experiments also the cosine kernel function given by:

$$k_c(\mathbf{z}_1, \mathbf{z}_2) = \frac{\mathbf{f}(\mathbf{z}_1)^{\mathrm{T}} \mathbf{f}(\mathbf{z}_2)}{\mathbf{f}(\mathbf{z}_1) \cdot \mathbf{f}(\mathbf{z}_2)}, \tag{6}$$

that represents the normalized version dot product. Considering the geometrical representation, $k_c(\mathbf{z}_1, \mathbf{z}_2)$ can be expressed as $\cos\alpha$ (see the example given by Fig. 2). The support set is represented by two examples from different classes, $\mathbf{f}_1$ and $\mathbf{f}_2$. The target model parameters $\boldsymbol{\theta}_T$ are cre-

Algorithm 1 HyperShot - training and prediction functions

Require: Training set $\mathcal{D} = \{\mathcal{T}_n\}_{n=1}^{N}$, and $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{X}_*\}$ test task.
Parameters: $\boldsymbol{\theta}_H$ - parameters , $\boldsymbol{\theta}_k$ - kernel parameters, and $\boldsymbol{\theta}_E$ - encoder parameters
Hyperparameters: N_{train} - number of training iterations, N_{tune} number of tuning iterations, α- step size.

function TRAIN($\mathcal{D}$, α, N_{train}, $\boldsymbol{\theta}_H$, $\boldsymbol{\theta}_k$, $\boldsymbol{\theta}_E$)
  while $n \leq N_{train}$ **do**
    Sample task $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\} \sim \mathcal{D}$
    Assign support $\mathcal{S} = \{(\mathbf{x}_m, \mathbf{y}_m)\}_{m=1}^{M}$
    $L = -\sum_{m=1}^{M}\sum_{k=1}^{K} y_m^k \log p(y_m^k \mid \mathcal{S}_i, \mathbf{x}_m, \boldsymbol{\theta}_H, \boldsymbol{\theta}_k, \boldsymbol{\theta}_E)$
    Update: $\boldsymbol{\theta}_E \leftarrow \boldsymbol{\theta}_E - \alpha\nabla_{\theta_E}\mathcal{L}$,
      $\boldsymbol{\theta}_H \leftarrow \boldsymbol{\theta}_H - \alpha\nabla_{\theta_H}\mathcal{L}$,
      $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k - \alpha\nabla_{\theta_k}\mathcal{L}$
    $n = n + 1$
  end while
  return $\boldsymbol{\theta}_H, \boldsymbol{\theta}_k, \boldsymbol{\theta}_E$
end function
function PREDICT($\mathcal{T}_*$, α, N_{tune}, $\boldsymbol{\theta}_H$, $\boldsymbol{\theta}_k$, $\boldsymbol{\theta}_E$)
  Create tuning task: $\mathcal{T}_t = \{\mathcal{S}_*, \mathcal{S}_*\}$
  Adapt $\hat{\boldsymbol{\theta}}_H, \hat{\boldsymbol{\theta}}_k, \hat{\boldsymbol{\theta}}_E$ = TRAIN($\mathcal{T}_t$, α, N_{tune}, $\boldsymbol{\theta}_H$, $\boldsymbol{\theta}_k$, $\boldsymbol{\theta}_E$)
  for each $\mathbf{x} \in \mathcal{X}_*$ **do**
    return $\arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathcal{S}_*, \mathbf{x}, \hat{\boldsymbol{\theta}}_H, \hat{\boldsymbol{\theta}}_k, \hat{\boldsymbol{\theta}}_E)$
  end for
end function

ated based only on the cosine value of the angle between vectors $\mathbf{f}_1$ and $\mathbf{f}_2$. During the classification stage, the query example is represented by $\mathbf{f}_\mathbf{x}$, and the classification is applied on the cosine values of angles between $\mathbf{f}_\mathbf{x}$ and $\mathbf{f}_1$, and $\mathbf{f}_\mathbf{x}$ and $\mathbf{f}_2$, respectively.

2.4 Training and prediction

The training procedure assumes the following parametrization of the model components. The encoder $E := E_{\boldsymbol{\theta}_E}$ is parametrized by $\boldsymbol{\theta}_E$, the hypernetwork $H = H_{\boldsymbol{\theta}_H}$ by $\boldsymbol{\theta}_H$ and the kernel function k by $\boldsymbol{\theta}_k$. We assume that training set $\mathcal{D}$ is represented by tasks $\mathcal{T}_i$ composed of support $\mathcal{S}_i$ and query $\mathcal{Q}_i$ examples. The training is performed by optimizing the cross-entropy criterion:

$$L = \sum_{\mathcal{T}_i \in \mathcal{D}} l_{\mathcal{T}_i} = \sum_{\mathcal{T}_i \in \mathcal{D}} \sum_{m=1}^{M} \sum_{k=1}^{K} y_{i,m}^k - \log p(y_{i,m}^k \mid \mathcal{S}_i, \mathbf{x}_{i,m}), \tag{7}$$

where $(\mathbf{x}_{i,n}, \mathbf{y}_{i,n})$ are examples from query set $\mathcal{Q}_i$, where $Q_i = \{(\mathbf{x}_{i,m}, \mathbf{y}_{i,m})\}_{m=1}^{M}$. The distribution for currently considered classes $p(\mathbf{y} \mid \mathcal{S}, \mathbf{x})$ is returned by target network T of HyperShot. During the training, we jointly optimize the parameters $\boldsymbol{\theta}_H$, $\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}_E$, minimizing the L loss.

During the inference stage, we consider the task $\mathcal{T}_*$, composed of a set of labeled support examples $\mathcal{S}_*$ and a set of unlabeled query examples represented by input values $\mathcal{X}_*$ that the model should classify. We can simply take the probability values $p(\mathbf{y} \mid \mathcal{S}_*, \mathbf{x})$ assuming the given support set $\mathcal{S}_*$ and single query observation $\mathbf{x}$ from $\mathcal{X}_*$, using the model with trained parameters $\boldsymbol{\theta}_H$, $\boldsymbol{\theta}_k$, and $\boldsymbol{\theta}_E$. However, we observe that slightly better results are obtained while adapting the model's parameters on the considered task. We do not have access to labels for query examples. Therefore, we imitate the query set for this task simply by taking support examples and creating the adaptation task $\mathcal{T}_i = \{\mathcal{S}_*, \mathcal{S}_*\}$ and updating the parameters of the model using several gradient iterations. The detailed presentation o training and prediction procedures is provided by Algorithm 1.

2.5 Adaptation to few-shot scenarios

The proposed approach uses the ordering function $\pi(\cdot)$ that keeps the consistency between support kernel matrix $\mathbf{K}_{S,S}$ and the vector of kernel values $\mathbf{k}_{\mathbf{x},S}$ for query example $\mathbf{x}$. For few-shot scenarios, each class has more than one representative in the support set. As a consequence, there are various possibilities to order the feature vectors in the support set inside the considered class. To eliminate this issue, we follow [8] and propose to apply the aggregation function to the embeddings $\mathbf{z}$ considering the support examples from the same class. Thanks to this approach, the kernel matrix is calculated based on the aggregated values of the latent space of encoding network E, making our approach independent of the ordering among the embeddings from the same class. In experimental studies, we examine the quality of *mean* aggregation operation (averaged) against simple class-wise concatenation of the embeddings (fine-grained) in ablation studies.

2.6 Extension to Bayesian setting—BayesHyperShot

Finally, we introduce the Bayesian extension of HyperShot, where the distribution over the parameters is represented by Gaussian prior:

$$p(\boldsymbol{\theta}_\mathcal{T}) = \mathcal{N}(\boldsymbol{\theta}_\mathcal{T} \mid 0, I).$$

Thanks to that assumption, the target model $\mathcal{T}$ serves probabilistic and can be used as a Bayesian network during inference.

In order to achieve that, we postulate to use of variational amortized posterior:

$$q(\boldsymbol{\theta}_\mathcal{T} \mid \mathcal{S}_i, \boldsymbol{\theta}_H) = \mathcal{N}(\boldsymbol{\theta}_\mathcal{T} \mid \mu_{\boldsymbol{\theta}_H}(\mathcal{S}_i), \sigma_{\boldsymbol{\theta}_H}(\mathcal{S}_i)),$$

where the Gaussian parameters $\mu(\mathcal{S}_i)$ and $\sigma(\mathcal{S}_i)$ are returned by hypernetwork $H_{\boldsymbol{\theta}_H}$ for a given support set $\mathcal{S}_i$, i.e., $(\mu(\mathcal{S}_i), \sigma(\mathcal{S}_i)) = H_{\boldsymbol{\theta}_H}(\mathcal{S}_i, \boldsymbol{\theta}_H)$. Compared to the basic HyperShot model (option 1 Fig. 1) that predicts deterministic

target weight, the proposed extension delivers the distribution over parameters for a given support set option 2 Fig. 1).

We train the BayesHyperShot using the procedure given by Algorithm 1 with the modified training objective, given by:

$$\mathcal{L}_B = \sum_{\mathcal{T}_i \in \mathcal{D}} \left[\frac{1}{P} \sum_{p=1}^{P} \left[l_{\mathcal{T}_i}(\boldsymbol{\theta}^p_{\mathcal{T}_i}) - \gamma KL(q(\boldsymbol{\theta}^p_{\mathcal{T}_i} \mid \mathcal{S}_i, \boldsymbol{\theta}_H) \mid \mathcal{N}(0, I)) \right] \right],$$

where $\boldsymbol{\theta}^1_{\mathcal{T}_i}, \ldots, \boldsymbol{\theta}^P_{\mathcal{T}_i} \sim q(\boldsymbol{\theta}_{\mathcal{T}} \mid \mathcal{S}_i, \boldsymbol{\theta}_H)$ are the target parameters sampled by variational posterior modeled using the hypernetwork, $l_{\mathcal{T}_i}(\boldsymbol{\theta}^p_{\mathcal{T}_i})$ is cross-entropy loss function calculated using target model with weights $\boldsymbol{\theta}^p_{\mathcal{T}_i}$ and $KL(\cdot, \cdot)$ is Kullback–Leibler divergence between the posterior and standard Gaussian. In order to stabilize the training procedure, we use hyperparameter γ that controls the trade-off between cross-entropy loss and a regularization term. During training, we apply an annealing scheme [13], for which γ grows from zero to a fixed constant during training. The final value $\gamma_{\max}$ is a hyperparameter of the model.

During the inference stage, we sample the set of target parameters $\boldsymbol{\theta}^1_{\mathcal{T}_i}, \ldots, \boldsymbol{\theta}^P_{\mathcal{T}_i} \sim q(\boldsymbol{\theta}_{\mathcal{T}} \mid \mathcal{S}_i, \boldsymbol{\theta}_H)$. The final prediction is made simply by averaging among sampled target models, $\frac{1}{P} \sum_{p=1}^{P} p(\mathbf{y} \mid \mathcal{S}_*, \mathbf{x}, \hat{\boldsymbol{\theta}}_H, \hat{\boldsymbol{\theta}}_k, \hat{\boldsymbol{\theta}}_E, \boldsymbol{\theta}^p_{\mathcal{T}_i})$.

3 Related work

In recent years, various meta-learning methods [14–16] have been proposed to tackle the problem of few-shot learning. The various meta-learning architectures for few-shot learning can be roughly categorized into several groups, which we below divide into non-Bayesian and Bayesian families of approaches.

3.1 Non-Bayesian approaches

In non-Bayesian few-shot learning, the goal is typically to learn a set of parameters that can be used to classify new examples based on a limited amount of training data.

Transfer learning [17] is a simple yet effective baseline procedure for few-shot learning which consists of pre-training the neural network and a classifier on all of the classes available during meta-training. During meta-validation, the classifier is then fine-tuned to the novel tasks. In [10], the authors proposed Baseline++, an extension of this idea that uses cosine distance between the examples.

Metric-based methods meta-learn a deep representation with a metric in feature space, such that distance between examples from the support and query set with the same class have a small distance in such space. Some of the earliest works exploring this notion are matching networks [18] and prototypical networks [19], which form *prototypes* based on embeddings of the examples from the support set in the learned feature space and classify the query set based on the distance to those prototypes. Numerous subsequent works aim to improve the expressiveness of the prototypes through various techniques. Oreshkin et al. [20] achieve this by conditioning the network on specific tasks, thus making the learned space task-dependent. Hu [21] transform embeddings of support and query examples in the feature space to make their distributions closer to Gaussian. Sung et al. [22] propose Relation Nets, which learn the metric function instead of using a fixed one, such as Euclidean or cosine distance.

Optimization-based methods follow the idea of an optimization process over support set within the meta-learning framework like MetaOptNet [23], Meta-SGD [24] or model-agnostic meta-learning (MAML) [25] and its extensions [26–31]. Those techniques aim to train general models, which can adapt their parameters to the support set at hand in a small number of gradient steps. Similar to such techniques, HyperShot (the classical realization of our framework) also aims to produce task-specific models but utilizes a hypernetwork instead of optimization to achieve that goal.

Hypernetworks-based methods [4] have been proposed as a solution to few-shot learning problems in a number of works but have not been researched as widely as the approaches mentioned above. Multiple works proposed various variations of hypernetworks that predict a shallow classifier's parameters given the support examples [5, 32, 33]. More recently, [7–9] explored generating all of the parameters of the target network with a transformer-based hypernetwork, but found that for larger target networks, it is sufficient to generate only the parameters of the final classification layer. A particularly effective approach is to use transformer-based hypernetworks as set-to-set functions which make the generated classifier more discriminative [6]. A key characteristic of the above approaches is that during inference, the hypernetwork predicts weights responsible for classifying each class independently, based solely on the examples of that class from the support set. This property makes such solutions agnostic to the number of classes in a task, useful in practical applications. However, it also means that the hypernetwork does not take advantage of the inter-class differences in the task at hand.

In contrast, models in our framework (in particular HyperShot) exploit those differences by utilizing kernels, which helps improve its performance.

3.2 Bayesian approaches

There are many few-shot learning methods that utilizes Bayesian approach to estimate the parameters of a model. We grouped them in three categories:

Bayesian optimization-based methods reformulate MAML as a hierarchical Bayesian model [34–39]. Contrary to this group of methods, the BayesHyperShot does not utilize a bi-level optimization scheme, such as MAML. Moreover, we look at the adaptation of the target network's weights not at the optimization process itself.

Probabilistic weight generation methods focus on predicting a distribution over the parameters suitable for the given task [40]. Similarly, our BayesHyperShot also predicts the probability over the parameters of the target network performing the few-shot classification. The key difference is that in BayesHyperShot, the target network combines such an approach with a kernel mechanism.

Gaussian processes-based methods [41] possess many properties useful in few-shot learning, such as natural robustness to the limited amounts of data and the ability to estimate uncertainty. When combined with meta-learned deep kernels, In [1], Gaussian processes were demonstrated to be a suitable tool for few-shot regression and classification, dubbed deep kernel transfer (DKT). The assumption that such a universal deep kernel has enough data to generalize well to unseen tasks has been challenged in subsequent works. [3] introduced a technique of learning dense Gaussian processes by inducing variables. This approach achieves substantial performance improvement over the alternative methods. Similarly, Bayesian version of our model (BayesHyperShot) also depends on learning a model that estimates task-specific functions' parameters. However, BayesHyperShot employs a hypernetwork instead of a Gaussian process to achieve that goal.

4 Experiments

In the typical few-shot learning setting, making a valuable and fair comparison between proposed models is often complicated because of the existence of significant differences in architectures and implementations of known methods. In order to limit the influence of the deeper backbone (feature extractor) architectures, we follow the unified procedure proposed by [10].

In this section, we describe the experimental analysis and performance of the proposed methods—HyperShot and BayesHyperShot—in a large variety of few-shot benchmarks. Specifically, we consider both classification (see Sect. 4.1) and cross-domain adaptation (see Sect. 4.2) tasks. Whereas the classification problems are focused on the most typical few-shot applications, the latter cross-domain benchmarks check the ability of the models to adapt to out-of-distribution tasks. We compare the classical realization of our framework (HyperShot) against the non-Bayesian approaches and the Bayesian version (BayesHyperShot) against Bayesian approaches only. We limit our comparison to models designed for the standard inductive few-shot setting. Notably, we exclude models which utilize anything besides the support set for fitting to the task at hand, such as methods that are transductive [42] or which utilize additional unlabeled data [43]. Additionally, in Sect. 4.3 we conduct an experiment showing the uncertainty estimation via BayesHyperShot. Finally, we perform an ablation study of the possible adaptation procedures of HyperShot to few-shot scenarios, as well as architectural choices—presented in Sect. 4.4.

In all of the reported experiments, the tasks consist of 5 classes (5 ways) and 1 or 5 support examples (1 or 5 shots). Unless indicated otherwise, all compared models use a known and widely utilized backbone consisting of four convolutional layers (each consisting of a 2D convolution, a batch-norm layer, and a ReLU nonlinearity; each layer consists of 64 channels) [10] and have been trained from scratch.

We report the performance of two versions of our general framework—classical (HyperShot) and Bayesian (BayesHyperShot). Note that each of them has two variants:

- HyperShot/BayesHyperShot–models generated by the hypernetworks for each task.
- HyperShot/BayesHyperShot + adaptation–models generated by hypernetworks adapted to the support examples of each task for 10 training steps.[1]

In all cases, we observe a modest performance boost thanks to adapting the hypernetwork. Comprehensive details and hyperparameters for each training procedure are reported in Appendices B and C.

4.1 Classification

Firstly, we consider a classical few-shot learning scenario, where all the classification tasks (both training and inference) come from the same dataset. The main aim of the proposed classification experiments is to find the ability of the few-shot models to adapt to never-seen tasks from the same data distribution.

We benchmark the performance of our models and other methods on two challenging and widely considered datasets: Caltech-USCD Birds (CUB) [44] and mini-ImageNet [45]. The following experiments are in the most popular setting, 5 ways, consisting of 5 random classes. In all experiments, the query set of each task consists of 16 samples for each class (80 in total).

HyperShot We start with a non-Bayesian perspective and compare HyperShot to a vast pool of state-of-the-art algo-

[1] In the case of the adapted hypernetworks, we tune a copy of the hypernetwork on the support set separately for each validation task. This way, we ensure that our model does not take unfair advantage of the validation tasks.

Table 1 Classification accuracy results for the non-Bayesian approaches

Method	CUB		Mini-ImageNet	
	One-shot	Five-shot	One-shot	Five-shot
Feature transfer [17]	46.19 ± 0.64	68.40 ± 0.79	39.51 ± 0.23	60.51 ± 0.55
Baseline++ [10]	61.75 ± 0.95	78.51 ± 0.59	47.15 ± 0.49	66.18 ± 0.18
ProtoNet [19]	52.52 ± 1.90	75.93 ± 0.46	44.19 ± 1.30	64.07 ± 0.65
RelationNet [22]	62.52 ± 0.34	78.22 ± 0.07	48.76 ± 0.17	64.20 ± 0.28
FO-MAML [26]	–	–	48.70 ± 1.84	63.11 ± 0.92
Reptile [26]	–	–	49.97 ± 0.32	65.99 ± 0.58
FEAT [6]	**68.87 ± 0.22**	**82.90 ± 0.15**	**55.15 ± 0.20**	**71.61 ± 0.16**
MAML [25]	56.11 ± 0.69	74.84 ± 0.62	45.39 ± 0.49	61.58 ± 0.53
MAML++ [27]	–	–	52.15 ± 0.26	68.32 ± 0.44
iMAML-HF [30]	–	–	49.30 ± 1.88	–
SignMAML [28]	–	–	42.90 ± 1.50	60.70 ± 0.70
Unicorn-MAML [29]	–	–	54.89	–
Meta-SGD [24]	–	–	50.47 ± 1.87	64.03 ± 0.94
PAMELA [46]	–	–	*53.50 ± 0.89*	*70.51 ± 0.67*
HyperMAML [31]	66.11 ± 0.28	78.89 ± 0.19	51.84 ± 0.57	66.29 ± 0.43
HyperShot [47]	65.27 ± 0.24	79.80 ± 0.16	52.42 ± 0.46	68.78 ± 0.29
HyperShot + adaptation [47]	*66.13 ± 0.26*	*80.07 ± 0.22*	53.18 ± 0.45	69.62 ± 0.20

We consider the inference tasks on *CUB* and *mini-ImageNet* datasets in the one-shot and five-shot settings. The highest results are in bold and the second-highest in italic (the larger, the better)

rithms, including the canonical methods (like matching networks [18], prototypical networks [19], MAML [25], and its extensions) as well as the recently popular: Unicorn-MAML [29] PAMELA [46].

We consider the more challenging one-shot classification task, as well as the five-shot setting and report the results in Table 1.

In the one-shot scenario, HyperShot achieves the second-best accuracy in the CUB dataset with adapting procedure (66.13% with adapting, 65.27% without) and performs better than any other model, except for FEAT [6] (68.87%). In the mini-ImageNet dataset, our approach is among the top approaches (53.18%), slightly losing with FEAT [6] (55.15%). Considering the five-shot scenario, HyperShot is the second-best model achieving 80.07% in the CUB dataset and 69.62% in the mini-ImageNet, whereas the best model, FEAT [6], achieves 82.90% and 71.61% on the mentioned datasets, respectively.

The obtained results clearly show that HyperShot achieves results comparable to state-of-the-art non-Bayesian models on the standard set of few-shot classification settings.

BayesHyperShot Then, we compare the Bayesian version of our general framework—BayesHyperShot against the state-of-the-art Bayesian methods. These Bayesian models are mostly built upon the Gaussian Processes framework (like DKT [1]). We consider the more challenging one-shot classification task, as well as the five-shot setting and report the results in Table 2. In the one-shot scenario, BayesHyperShot achieves the third-best accuracy in the CUB dataset despite the adapting procedure (66.30% in both cases) and performs similarly to the best model—BayesHMAML [36] (respectively, 66.92% with adaptation and 66.57% without). In the mini-ImageNet dataset, our Bayesian approach is among the top methods (51.11%), slightly losing with Bayesian MAML [34] (53.80%) and BayesHMAML (52.69%).

Considering the five-shot scenario, BayesHyperShot is the best model achieving 80.60% in the CUB dataset and 67.21% in the mini-ImageNet, whereas the most significant competitor, BayesHMAML [36] achieves 80.47% and 68.24% on the mentioned datasets, respectively.

According to the results, the performance of BayesHyperShot is comparable to or better than other state-of-the-art Bayesian models on the standard set of few-shot classification settings.

4.2 Cross-domain adaptation

In the cross-domain adaptation setting, the models are evaluated on tasks coming from a different distribution than the one they had been trained on. Therefore, such a task is more challenging than standard classification and is a plausible indicator of a model's ability to generalize. In order to benchmark the performance of our framework in cross-domain adaptation, we merge data from two datasets so that the training fold is drawn from the first dataset and validation and testing fold—from another one. Specifically, we test HyperShot on two cross-domain classification tasks:

Table 2 Classification accuracy results for the Bayesian approaches

Method	CUB		Mini-ImageNet	
	One-shot	Five-shot	One-shot	Five-shot
LLAMA [35]	–	–	49.40 ± 1.83	–
VERSA [40]	–	–	48.53 ± 1.84	67.37 ± 0.86
Amortized VI [40]	–	–	44.13 ± 1.78	55.68 ± 0.91
Meta-Mixture [39]	–	–	49.60 ± 1.50	64.60 ± 0.92
DKT + BNCosSim [1]	62.96 ± 0.62	77.76 ± 0.62	49.73 ± 0.07	64.00 ± 0.09
VAMPIRE [38]	–	–	51.54 ± 0.74	64.31 ± 0.74
ABML [37]	49.57 ± 0.42	68.94 ± 0.16	45.00 ± 0.60	–
Bayesian MAML [34]	55.93 ± 0.71		$\mathbf{53.80 \pm 1.46}$	64.23 ± 0.69
BayesHMAML [36]	$\mathit{66.57 \pm 0.47}$	79.86 ± 0.31	52.54 ± 0.46	$\mathit{67.39 \pm 0.35}$
BayesHMAML + adaptation [36]	$\mathbf{66.92 \pm 0.38}$	$\mathit{80.47 \pm 0.38}$	$\mathit{52.69 \pm 0.38}$	$\mathbf{68.24 \pm 0.47}$
BayesHyperShot	66.30 ± 0.42	80.43 ± 0.34	50.81 ± 0.33	66.51 ± 0.71
BayesHyperShot + adaptation	66.30 ± 0.42	$\mathbf{80.60 \pm 0.37}$	51.11 ± 0.35	67.21 ± 0.72

We consider the inference tasks on *CUB* and *mini-ImageNet* datasets in the one-shot and five-shot settings. The highest results are in bold and the second-highest in italic (the larger, the better)

Table 3 Classification accuracy results for the non-Bayesian approaches

Method	Omniglot→EMNIST		mini-ImageNet→CUB	
	One-shot	Five-shot	One-shot	Five-shot
Feature transfer [17]	64.22 ± 1.24	86.10 ± 0.84	32.77 ± 0.35	50.34 ± 0.27
Baseline++ [10]	56.84 ± 0.91	80.01 ± 0.92	$\mathit{39.19 \pm 0.12}$	57.31 ± 0.11
ProtoNet [19]	72.04 ± 0.82	87.22 ± 1.01	33.27 ± 1.09	52.16 ± 0.17
RelationNet [22]	75.62 ± 1.00	87.84 ± 0.27	37.13 ± 0.20	51.76 ± 1.48
MAML [25]	74.81 ± 0.25	83.54 ± 1.79	34.01 ± 1.25	48.83 ± 0.62
HyperMAML [31]	79.07 ± 1.09	$\mathit{89.22 \pm 0.78}$	36.32 ± 0.61	49.43 ± 0.14
HyperShot [47]	78.06 ± 0.24	89.04 ± 0.18	39.09 ± 0.28	$\mathit{57.77 \pm 0.33}$
HyperShot + adaptation [47]	$\mathbf{80.65 \pm 0.30}$	$\mathbf{90.81 \pm 0.16}$	$\mathbf{40.03 \pm 0.41}$	$\mathbf{58.86 \pm 0.38}$

We consider the inference tasks on cross-domain tasks (Omniglot→EMNIST and mini-ImageNet→CUB) datasets in the one-shot and five-shot setting. The highest results are bold and second-highest in italic (the larger, the better)

mini-ImageNet → CUB (model trained on mini-ImageNet and evaluated on CUB) and Omniglot → EMNIST in the one-shot and five-shot settings. Similarly to the previous experiment, we treated Bayesian and non-Bayesian approaches separately.

HyperShot We start with the non-Bayesian approaches and report the results in Table 3. In every setting, HyperShot achieves the highest accuracy and, as such, is much better than any other non-Bayesian approach. We note that just like in the case of regular classification, adapting the hypernetwork on the individual tasks consistently improves its performance.

BayesHyperShot The results among the Bayesian methods are reported in Table 4. We observer that in every setting BayesHyperShot is comparable but slightly worse than the best models. The difference is usually between 1 and 3 percent points, making the BayesHyperShot among three or four best Bayesian methods. It is worth noting that in this setting, performing the adaptation procedure on BayesHyperShot could result in worse performance than not adapting at all.

4.3 Uncertainty quantification

The most important feature of the Bayesian realization of our general framework is that we have the full insight into the model's uncertainty. In fact, due to the BayesHyperShot's probabilistic construction, we can quantify the level of model's certainty and answer the question if the model is sure about the specific prediction. Moreover, this property, enable us to say if the given sample comes from the known distribution—the distribution related to the current few-shot task or if it is out-of-distribution example.

In the following experiment, we present the uncertainty quantification of the BayesHyperShot model. We consider a

Table 4 Classification accuracy results for the Bayesian approaches

Method	Omniglot→EMNIST One-shot	Five-shot	Mini-ImageNet→CUB One-shot	Five-shot
DKT [1]	75.40 ± 1.10	**90.30 ± 0.49**	**40.14 ± 0.18**	*56.40 ± 1.34*
OVE PG GP + Cosine (ML) [48]	68.43 ± 0.67	86.22 ± 0.20	*39.66 ± 0.18*	55.71 ± 0.31
OVE PG GP + Cosine (PL) [48]	77.00 ± 0.50	87.52 ± 0.19	37.49 ± 0.11	**57.23 ± 0.31**
Bayesian MAML [34]	63.94 ± 0.47	65.26 ± 0.30	33.52 ± 0.36	51.35 ± 0.16
BayesHMAML [36]	*80.95 ± 0.46*	89.21 ± 0.27	36.90 ± 0.34	49.24 ± 0.38
BayesHMAML + adaptation [36]	**81.05 ± 0.47**	*89.76 ± 0.26*	37.23 ± 0.44	50.79 ± 0.59
BayesHyperShot	79.71 ± 0.23	89.54 ± 0.25	37.85 ± 0.22	51.37 ± 0.69
BayesHyperShot + adaptation	79.65 ± 0.33	89.70 ± 0.12	37.02 ± 0.36	51.62 ± 0.72

We consider the inference tasks on cross-domain tasks (Omniglot→EMNIST and mini-ImageNet→CUB) datasets in the one-shot and five-shot setting. The highest results are bold and second-highest in italic (the larger, the better)

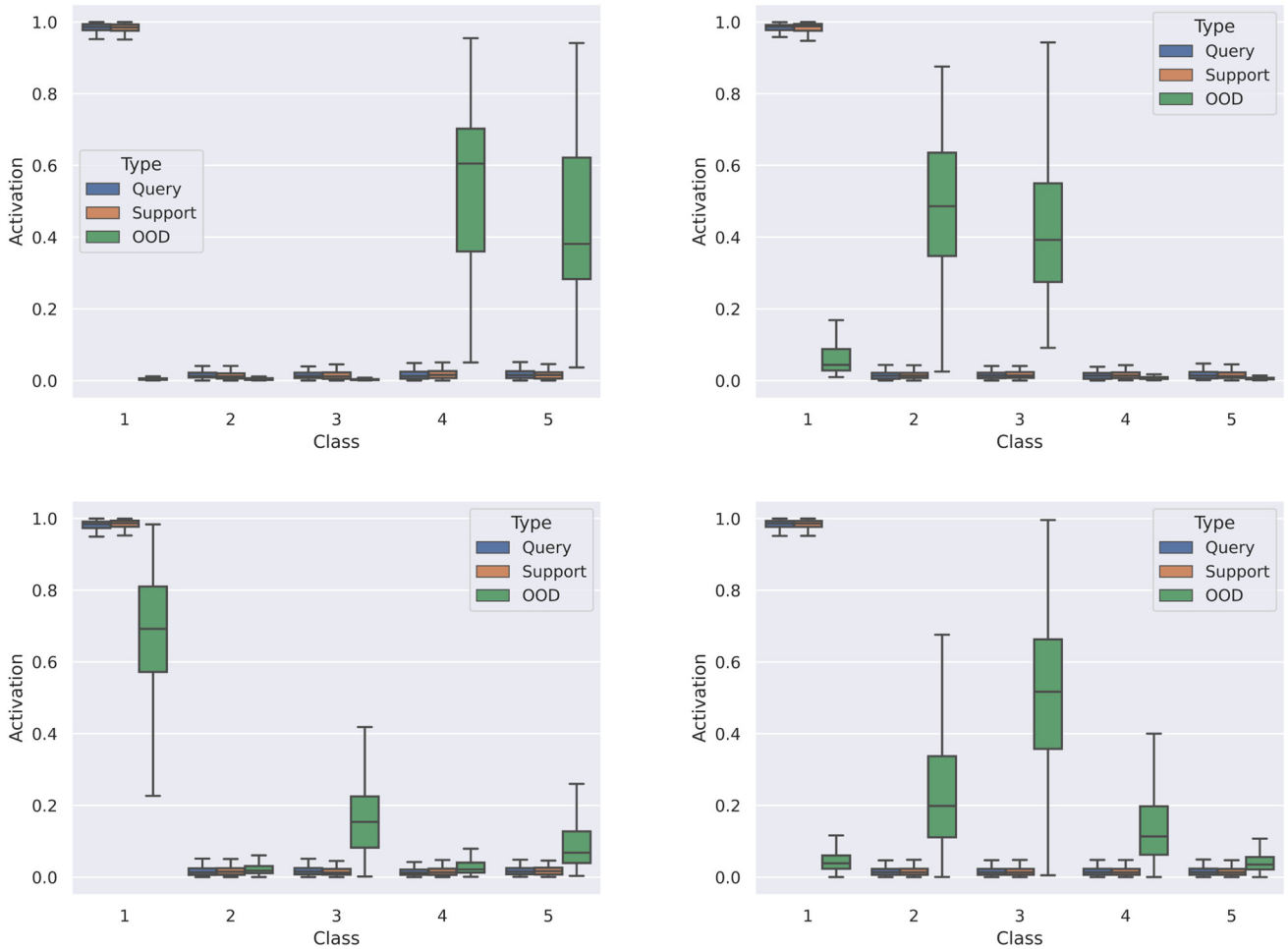

Fig. 3 We visualize box plots of distributions of activations produced by the sampled models for the four different sets of support/query/out-of-distribution images. As we can see, BayesHyperShot always yields similar predictions for elements from both the support and query sets. On the other hand, we observe a high variance of activations when processing out-of-distribution images, which indicates the high uncertainty of the model in such cases

model trained on the specific dataset and setting (here, it was Omniglot → EMNIST). Then, we have taken three different types of samples to quantify the level of uncertainty of our model's predictions. Specifically, we test the images from:

Table 5 Classification *accuracy* results for HyperShot in the five-shot setting with two variants of the support embeddings aggregation

	Omni→EMNIST	CUB	Mini-ImageNet
HyperShot (fine-grained)	87.55 ± 0.19	78.05 ± 0.20	67.07 ± 0.47
HyperShot (averaged)	89.04 ± 0.18	79.80 ± 0.16	69.62 ± 0.28

The performance measured on Omniglot→EMNIST, *CUB*, and mini-ImageNet→CUB tasks. The larger, the better

- Support set;
- Query set;
- Coming from the same dataset, but with class not present in the support set (out of distribution).

We observe that the examples coming from the support set or query set are classified with very high certainty. The model classifies such images as from a given class (label 1) or not (label 0). Note that label 1 appears for only one class, and label 0 for the rest.

However, the most important is the result obtained for the samples of classes not present in the support set. We observe that the model is uncertain about the classification—it usually gives nonzero probabilities for each of the possible classes. The present increase in the entropy of probabilities allows us to classify such examples as out-of-distribution samples. The results are presented in Fig. 3.

4.4 Ablation study

In order to investigate different architectural choices in adapting our framework to the specific task, we provide a comprehensive ablation study. In this ablation study, we consider the HyperShot model only for better clarity and apply our findings when selecting the parameters of BayesHyperShot. We focused mostly on the four major components of the HyperShot design, i.e., the method of processing multiple support examples per class, the number of neck layers, the number of head layers and the size of the hidden layers, presented in Tables 5, 6 and 7. In the case of the experiments focusing on aggregating the number of support examples in the five-way five-shot setting, we perform the benchmarks on CUB and mini-ImageNet, using a four-layer convolutional backbone. In the remaining experiments, we tested HyperShot on the CUB dataset in the five-way one-shot setting with ResNet-10 backbone.

Aggregating support examples in the five-shot setting

In HyperShot, the hypernetwork generates the weights of the information about the support examples, expressed through the support–support kernel matrix. In the case of five-way one-shot classification, each task consists of 5 support examples, and therefore, the size of the kernel matrix is (5×5), and the input size of the hypernetwork is 25. However, with a growing number of the support examples, increasing the size of the kernel matrix would be impractical and could lead to overparametrization of the hypernetwork.

Since hypernetworks are known to be sensitive to large input sizes [4], we consider a way to maintain a constant input size of HyperShot, independent of the number of support examples of each class by using means of support embeddings of each class for kernel calculation, instead of individual embeddings. Prior works suggest that when there are multiple examples of a class, the averaged embedding of such class represents it sufficiently in the embedding space [19].

To verify this approach, in the five-shot setting, we train HyperShot with two variants of calculating the inputs to the kernel matrix:

- Fine-grained—utilizing a hypernetwork that takes as an input a kernel matrix between each of the embeddings of the individual support examples. This kernel matrix has a shape of (25×25).
- Averaged—utilizing a hypernetwork where the kernel matrix is calculated between the means of embeddings of each class. The kernel matrix in this approach has a shape of (5×5).

We benchmark both variants of HyperShot on the five-shot classification task on CUB and mini-ImageNet datasets, as well as the task of cross-domain Omniglot → EMNIST classification. We report the accuracies in Table 5. It is evident that averaging the embeddings before calculating the kernel matrix yields superior results.

Hidden size: Firstly, as presented in Table 6, we compare different sizes of hidden layers. The results agree with the intuition that the wider the layers, the better the results. However, we also observe that some hidden sizes (e.g., 8188) could be too large to learn effectively. Because of that, we propose to use hidden sizes of 2048 or 4096 as the standard.

Neck and head layers: Then, we compared the influence of the number of neck layers and head layers of HyperShot for the achieved results, as presented in Table 7. We observed that the most critical is the number of head layers—specific for each target network's layers. Because of that, we propose using the standard number of 3 head layers and using various neck layers—tuning them to the specific task.

Table 6 Comparison between various hidden sizes in the HyperShot's layers

Hidden size	Accuracy
256	70.16 ± 0.45
512	71.70 ± 0.46
1024	70.89 ± 0.62
2048	$\mathbf{72.43 \pm 0.59}$
4096	71.99 ± 0.70
8188	72.05 ± 0.33

The classification *accuracy* results on CUB task and five-way one-shot setting. The larger, the better

Table 7 Comparison between various HyperShot's architectures (different number of neck layers and head layers)

Neck layers	Head layers	Accuracy
1	**3**	$\mathbf{73.00 \pm 0.55}$
2	1	71.53 ± 0.33
2	2	68.06 ± 0.59
2	3	71.99 ± 0.70
3	3	70.81 ± 0.39

The classification *accuracy* results on CUB task and five-way one-shot setting. The larger, the better

5 Conclusion

In this work, we introduced a novel general framework that uses kernel methods combined with hypernetworks. Our method directly relies on the kernel-based representations of the support examples and a hypernetwork paradigm to create the query set's classification module. We concentrate on relations between embeddings of the support examples instead of direct feature values. Thanks to this approach, models that realize our framework can adapt to highly different tasks.

Specifically, in this paper, we propose a classical (HyperShot) and a Bayesian (BayesHyperShot) realization of our framework. We evaluate both models on various one-shot and few-shot image classification tasks. Both HyperShot and BayesHyperShot demonstrate high accuracy in all tasks, performing comparably or better to state-of-the-art solutions. Moreover, both models have a strong ability to generalize, as evidenced by their performances on cross-domain classification tasks.

Finally, we demonstrate the ability of the Bayesian version of our framework to properly quantify the uncertainty of the model's prediction. As such, BayesHyperShot is able to recognize an out-of-distribution samples and return the level of certainty of the classification.

Acknowledgements This research was funded in part by National Science Centre, Poland, 2022/45/N/ST6/03374. The work of M. Sendera was supported by the National Centre of Science (Poland) Grant No. 2022/45/N/ST6/03374. The work of J. Tabor was supported by the National Centre of Science (Poland) Grant No. 2019/33/B/ST6/00894. The work of P. Spurek and M. Przewięźlikowski was supported by the National Centre of Science (Poland) Grant No. 2021/43/B/ST6/01456. The work of M. Zięba was supported by the National Centre of Science (Poland) Grant No. 2020/37/B/ST6/03463.

Appendix A. Comparison to the original HyperShot publication

This work is an extension of "HyperShot: Few-Shot Learning by Kernel Hypernetworks," originally published at the WACV 2023 Conference [47]. In this section, we address the points raised by the reviewers of the original publication, and outline which parts of this work are extensions of the original publication.

A.1 Addressing the WACV 2023 reviews

Hypernetwork architecture The reviewers raised a concern about a lack of a detailed description of our Hypernetwork architecture. While the general architecture had been visualized as a part of Fig. 4, we also expanded Sect. 2.2 to include a more detailed description of the architecture.

Comparisons to related work The reviewers suggested several additional works to which we could compare HyperShot. While we added several of them to the tables reported in our experiments [49, 50], we chose to omit others [42, 43, 51] as the settings of the few-shot problems they solve differ from the standard inductive few-shot classification.

Application to other tasks besides image classification An important point raised by the reviewers was the lack of experiments on different problems besides few-shot image classification. While we acknowledge that there are various different computer vision problems for which data-efficient solutions would be desirable, we felt that such experiments are out of the scope of our work, as HyperShot and BayesHyperShot were tailor-made for the classification problem, which is by far the most popular few-shot benchmark. An application of kernel Hypernetworks to other few-shot problems could be an interesting point of future work.

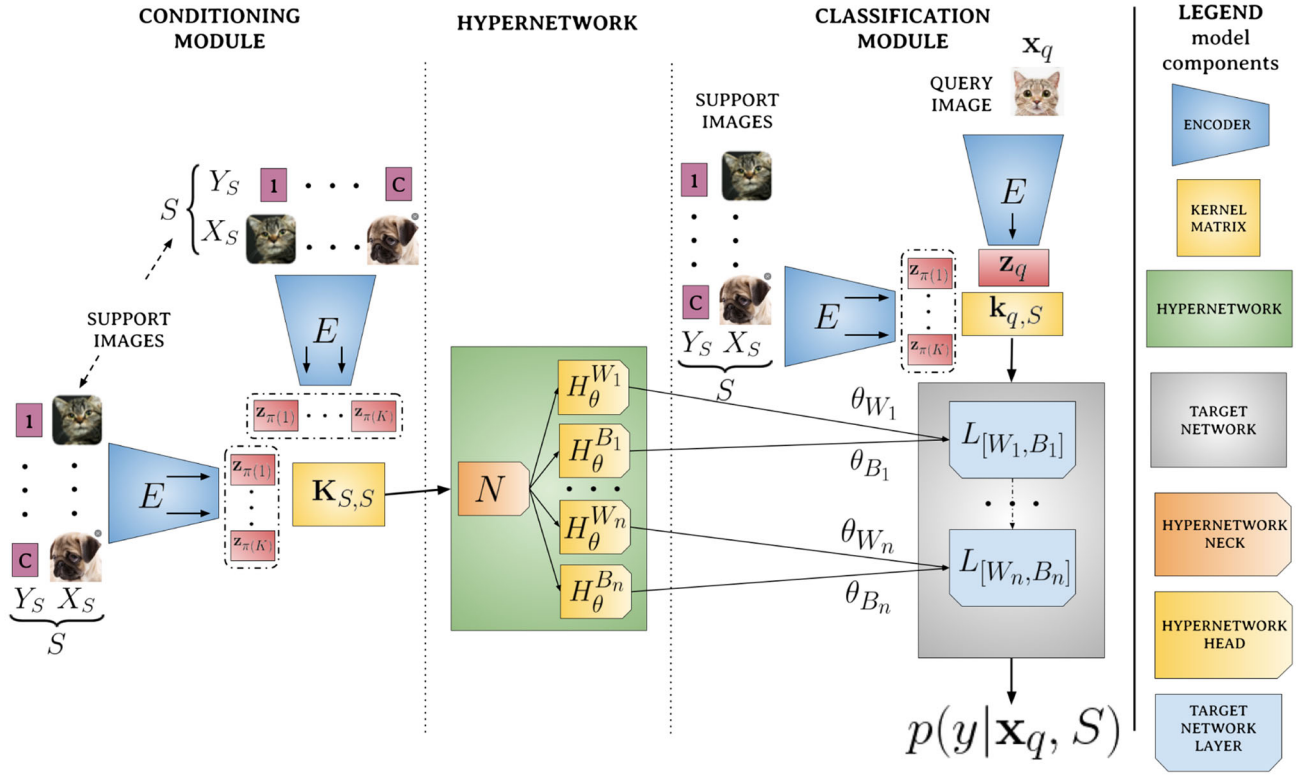

Fig. 4 A detailed outline of the architecture of HyperShot, with the denoted flow of parameters generated by the hypernetwork heads

A.2 Extensions compared to the WACV 2023 paper

The most important difference of this work compared to the WACV 2023 publications is the generalization of the previously introduced HyperShot method to a Bayesian framework, resulting in a model which we call BayesHyperShot. We conduct a series of experiments on the HyperShot model and show that apart from learning from small amounts of data, it is capable of modeling uncertainty of predictions.

Appendix B. Training details

In this section, we present in detail the architecture and hyperparameters of HyperShot.

Architecture overview

From a high-level perspective, the architecture of HyperShot consists of three parts:

- Backbone—a convolutional feature extractor.
- Neck—a sequence of zero or more fully connected layers with ReLU nonlinearities in between.
- Heads—for each parameter of the target network, a sequence of one or more linear layers, which predicts the values of that parameter. All heads of HyperShot have identical lengths, hidden sizes and input sizes that depend on the generated parameter's size.

The target network generated by both HyperShot and BayesHyperShot reuses its backbone. We outline this architecture in Fig. 4.

Backbone For each experiment described in the main body of this work, we follow [1] in using a shallow backbone (feature extractor) for HyperShot as well as referential models. This backbone consists of four convolutional layers, each consisting of a convolution, batch normalization and ReLU nonlinearity. Apart from the first convolution, which has the input size equal to the number of image channels, each convolution has an input and output size of 64. We apply max-pooling between each convolution, which decreases by half the resolution of the processed feature maps. The output of the backbone is flattened so that the further layers can process it.

Datasets For the purpose of making a fair comparison, we follow the procedure presented in, e.g., [1, 10]. In the case of the CUB dataset [44], we split the whole amount of 200 classes (11788 images) across train, validation and test consisting of 100, 50 and 50 classes, respectively [10]. The mini-ImageNet dataset [45] is created as the subset of ImageNet [52], which consists of 100 different classes represented by 600 images

Table 8 Hyperparameters

Hyperparameter	CUB	Mini-ImageNet	Mini-ImageNet → CUB	Omniglot → EMNIST
Kernel function	Cosine similarity	Cosine similarity	Cosine similarity	Cosine similarity
Learning rate	0.001	0.001	0.001	0.001
Hypernetwork's head layers no	3	3	3	2
Hypernetwork's neck layers no	2	2	2	1
Hypernetwork layers' hidden dim	4096	4096	4096	512
Support embeddings aggregation	Averaged	Averaged	Averaged	Averaged
Task set size	1	1	1	1
Target network layers no	1	1	1	2
Target network activation	ReLU	ReLU	ReLU	ReLU
Adaptation epochs (if used)	10	10	10	10
Adaptation learning rate	0.0001	0.0001	0.0001	0.0001
Optimizer	Adam	Adam	Adam	Adam
Epochs no	10000	10000	10000	2000

for each one. We followed the standard procedure and divided the mini-ImageNet into 64 classes for the train, 16 for the validation set and the remaining 20 classes for the test. The well-known Omniglot dataset [53] is a collection of characters from 50 different languages. The Omniglot contains 1623 white and black characters in total. We utilize the standard procedure to include the examples rotated by 90° and increase the size of the dataset to 6492, from which 4114 were further used in training. Finally, the EMNIST dataset [54] collects the characters and digits coming from the English alphabet, which we split into 31 classes for the test and 31 for validation.

Data augmentation We apply data augmentation during model training in all experiments, except Omniglot → EMNIST cross-domain classification. The augmentation pipeline is identical to the one used by [1] and consists of the random crop, horizontal flip and color jitter steps.

Appendix C. Hyperparameters

Below, we outline the hyperparameters of architecture and training procedures used in each experiment.

We use cosine similarity as a kernel function and averaged support embeddings aggregation in all experiments. HyperShot is trained with the learning rate of 0.001 with the Adam optimizer [55] and no learning rate scheduler. Task-specific adaptation is also performed with the Adam optimizer and the learning rate of 0.0001.

For the natural image tasks (CUB, mini-ImageNet, mini-ImageNet → CUB classification), we use a hypernetwork with the neck length of 2, head lengths of 3 and a hidden size of 4096, which produce a target network with a single fully connected layer. We perform training for 10000 epochs.

For the simpler Omniglot → EMNIST character classification task, we train a smaller hypernetwork with the neck length of 1, head lengths of 2 and the hidden size of 512, which produces a target network with two fully connected layers and a hidden size of 128. We train this hypernetwork for a shorter number of epochs, namely 2000.

We summarize all the above hyperparameters in Table 8.

Appendix D. Source code

The source code required for running the experiments is available at https://github.com/gmum/few-shot-hypernets-public.

References

1. Patacchiola, M., Turner, J., Crowley, E.J., O'Boyle, M., Storkey, A.J.: Bayesian meta-learning for the few-shot setting via deep kernels. Adv. Neural Inf. Process. Syst. **33**, 16108–16118 (2020)
2. Sendera, M., Tabor, J., Nowak, A., Bedychaj, A., Patacchiola, M., Trzcinski, T., Spurek, P., Zieba, M.: Non-gaussian gaussian processes for few-shot regression. Adv. Neural Inf. Process. Syst. **34**, 10285–10298 (2021)
3. Wang, Z., Miao, Z., Zhen, X., Qiu, Q.: Learning to learn dense gaussian processes for few-shot learning. Adv. Neural Inf. Process. Syst. **34**, 13230–13241 (2021)
4. Ha, D., Dai, A.M., Le, Q.V.: Hypernetworks. In: International Conference on Learning Representations (2017). https://openreview.net/forum?id=rkpACe1lx
5. Qiao, S., Liu, C., Shen, W., Yuille, A.L.: Few-shot image recognition by predicting parameters from activations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7229–7238 (2018)
6. Ye, H.-J., Hu, H., Zhan, D.-C., Sha, F.: Few-shot learning via embedding adaptation with set-to-set functions. In: IEEE/CVF

Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8808–8817 (2020)
7. Zhmoginov, A., Sandler, M., Vladymyrov, M.: Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In: International Conference on Machine Learning, pp. 27075–27098. PMLR (2022)
8. Zhu, Z., Wang, L., Guo, S., Wu, G.: A Closer Look at Few-Shot Video Classification: A New Baseline and Benchmark. arXiv (2021). https://doi.org/10.48550/ARXIV.2110.12358. arXiv:2110.12358
9. Perrett, T., Masullo, A., Burghardt, T., Mirmehdi, M., Damen, D.: Temporal-relational crosstransformers for few-shot action recognition. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 475–484 (2021). https://doi.org/10.1109/CVPR46437.2021.00054
10. Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C.F., Huang, J.-B.: A closer look at few-shot classification. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=HkxLXnAcFQ
11. Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M.: Generalizing from a few examples: A survey on few-shot learning. ACM Comput. Surv. (csur) **53**(3), 1–34 (2020)
12. Sheikh, A.-S., Rasul, K., Merentitis, A., Bergmann, U.: Stochastic maximum likelihood optimization via hypernetworks. arXiv preprint arXiv:1712.01141 (2017)
13. Bowman, S.R., Vilnis, L., Vinyals, O., Dai, A.M., Jozefowicz, R., Bengio, S.: Generating sentences from a continuous space. In: 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, pp. 10–21. Association for Computational Linguistics (ACL) (2016)
14. Bengio, S., Bengio, Y., Cloutier, J., Gescei, J.: On the optimization of a synaptic learning rule. In: Optimality in Biological and Artificial Networks, pp. 281–303 (2013)
15. Hospedales, T., Antoniou, A., Micaelli, P., Storkey, A.: Meta-learning in neural networks: A survey. IEEE Trans. Pattern Anal. Mach. Intell. **44**(9), 5149–5169 (2021)
16. Schmidhuber, J.: Learning to control fast-weight memories: an alternative to dynamic recurrent networks. Neural Comput. **4**(1), 131–139 (1992). https://doi.org/10.1162/neco.1992.4.1.131
17. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. Proc. IEEE (2020). https://doi.org/10.1109/JPROC.2020.3004555
18. Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al.: Matching networks for one shot learning. Adv. Neural. Inf. Process. Syst. **29**, 3630–3638 (2016)
19. Snell, J., Swersky, K., Zemel, R.: Prototypical networks for few-shot learning. Adv. Neural Inf. Process. Syst. **30** (2017)
20. Oreshkin, B., Rodríguez López, P., Lacoste, A.: Tadam: Task dependent adaptive metric for improved few-shot learning. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31 (2018). https://proceedings.neurips.cc/paper_files/paper/2018/file/66808e327dc79d135ba18e051673d906-Paper.pdf
21. Hu, Y., Gripon, V., Pateux, S.: Leveraging the feature distribution in transfer-based few-shot learning. In: Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part II 30, pp. 487–499. Springer (2021)
22. Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P.H., Hospedales, T.M.: Learning to compare: Relation network for few-shot learning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1199–1208 (2018)
23. Lee, K., Maji, S., Ravichandran, A., Soatto, S.: Meta-learning with differentiable convex optimization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 10657–10665 (2019)
24. Li, Z., Zhou, F., Chen, F., Li, H.: Meta-sgd: Learning to learn quickly for few-shot learning. arXiv preprint arXiv:1707.09835 (2017)
25. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: International Conference on Machine Learning, pp. 1126–1135. PMLR (2017)
26. Nichol, A., Achiam, J., Schulman, J.: On first-order meta-learning algorithms. arXiv preprint arXiv:1803.02999 (2018)
27. Antoniou, A., Edwards, H., Storkey, A.: How to train your MAML. In: International Conference on Learning Representations (2019). https://openreview.net/forum?id=HJGven05Y7
28. Fan, C., Ram, P., Liu, S.: Sign-maml: Efficient model-agnostic meta-learning by signsgd. In: 5th Workshop on Meta-Learning at NeurIPS 2021 (2021)
29. Ye, H.-J., Chao, W.-L.: How to train your MAML to excel in few-shot classification. In: International Conference on Learning Representations (2022). https://openreview.net/forum?id=49h_IkpJtaE
30. Rajeswaran, A., Finn, C., Kakade, S.M., Levine, S.: Meta-learning with implicit gradients. Adv. Neural. Inf. Process. Syst. **32**, 113–124 (2019)
31. Przewięźlikowski, M., Przybysz, P., Tabor, J., Zięba, M., Spurek, P.: Hypermaml: Few-shot adaptation of deep models with hypernetworks. arXiv preprint arXiv:2205.15745 (2022)
32. Bauer, M., Rojas-Carulla, M., Świątkowski, J.B., Schölkopf, B., Turner, R.E.: Discriminative k-shot learning using probabilistic models. arXiv preprint arXiv:1706.00326 (2017)
33. Garnelo, M., Rosenbaum, D., Maddison, C., Ramalho, T., Saxton, D., Shanahan, M., Teh, Y.W., Rezende, D., Eslami, S.M.A.: Conditional neural processes. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1704–1713 (2018)
34. Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., Ahn, S.: Bayesian model-agnostic meta-learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, pp. 7343–7353 (2018)
35. Grant, E., Finn, C., Levine, S., Darrell, T., Griffiths, T.: Recasting gradient-based meta-learning as hierarchical bayes. In: International Conference on Learning Representations (2018)
36. Borycki, P., Kubacki, P., Przewięźlikowski, M., Kuśmierczyk, T., Tabor, J., Spurek, P.: Hypernetwork approach to Bayesian maml. arXiv preprint arXiv:2210.02796 (2022)
37. Ravi, S., Beatson, A.: Amortized Bayesian meta-learning. In: International Conference on Learning Representations (2018)
38. Nguyen, C., Do, T.-T., Carneiro, G.: Uncertainty in model-agnostic meta-learning using variational inference. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 3090–3100 (2020)
39. Jerfel, G., Grant, E., Griffiths, T.L., Heller, K.: Reconciling meta-learning and continual learning with online mixtures of tasks. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 9122–9133 (2019)
40. Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., Turner, R.: Meta-learning probabilistic inference for prediction. In: International Conference on Learning Representations (2018)
41. Rasmussen, C.E.: Gaussian processes in machine learning. In: Summer School on Machine Learning, pp. 63–71. Springer (2003)
42. Shen, X., Xiao, Y., Hu, S.X., Sbai, O., Aubry, M.: Re-ranking for image retrieval and transductive few-shot classification. In: Advances in Neural Information Processing Systems, vol. 34, pp. 25932–25943 (2021)
43. Gidaris, S., Bursuc, A., Komodakis, N., Pérez, P., Cord, M.: Boosting few-shot visual learning with self-supervision. In: Proceedings of the IEEE International Conference on Computer Vision (2019)

44. Wah, C., Branson, S., Welinder, P., Perona, P., Belongie, S.: The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology (2011)
45. Ravi, S., Larochelle, H.: Optimization as a model for few-shot learning. In: ICLR (2017)
46. Rajasegaran, J., Khan, S.H., Hayat, M., Khan, F.S., Shah, M.: Meta-learning the learning trends shared across tasks. CoRR **abs/2010.09291** (2020)
47. Sendera, M., Przewięźlikowski, M., Karanowski, K., Zięba, M., Tabor, J., Spurek, P.: Hypershot: Few-shot learning by kernel hypernetworks. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 2469–2478 (2023)
48. Snell, J., Zemel, R.: Bayesian few-shot classification with one-vs-each pólya-gamma augmented gaussian processes. In: International Conference on Learning Representations (2020)
49. Tian, Y., Wang, Y., Krishnan, D., Tenenbaum, J.B., Isola, P.: Rethinking few-shot image classification: a good embedding is all you need? In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16, pp. 266–282. Springer (2020)
50. Rizve, M.N., Khan, S., Khan, F.S., Shah, M.: Exploring complementary strengths of invariant and equivariant representations for few-shot learning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10836–10846 (2021)
51. Jian, Y., Torresani, L.: Label hallucination for few-shot classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 7005–7014 (2022)
52. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015)
53. Lake, B., Salakhutdinov, R., Gross, J., Tenenbaum, J.: One shot learning of simple visual concepts. In: Proceedings of the Annual Meeting of the Cognitive Science Society, vol. 33 (2011)
54. Cohen, G., Afshar, S., Tapson, J., van Schaik, A.: Emnist: Extending mnist to handwritten letters. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2921–2926 (2017). https://doi.org/10.1109/IJCNN.2017.7966217
55. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations (2014)