HyperShot: Few-Shot Learning by Kernel HyperNetworks

Marcin Sendera [†1] Marcin Przewięźlikowski[†1] Konrad Karanowski[2]
Maciej Zięba[2,3] Jacek Tabor[1] Przemysław Spurek[1]

[1]Faculty of Mathematics and Computer Science, Jagiellonian University
6 Łojasiewicza Street, 30-348 Kraków, Poland

[2]Department of Artificial Intelligence, University of Science and Technology
Wyb. Wyspiańskiego 27, 50-370, Wrocław, Poland

[3]Tooploox
Tęczowa 7, 53-601, Wrocław, Poland
marcin.{sendera, przewiezlikowski}@doctoral.uj.edu.pl

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 HyperShot - the fusion of kernels and hypernetwork paradigm. Compared to reference approaches that apply a gradient-based adjustment of the parameters, our model aims 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 embeddings of the support examples instead of direct feature values provided by the backbone models. Thanks to this approach, our model can adapt to highly different tasks. [*]

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 small number of labeled examples. The goal of few-shot learning is not to recognize a fixed set of labels but to learn how 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.

Two new Few-shot learning techniques have recently emerged. The first one is based on the kernel method and Gaussian processes [20, 31, 39]. The universal deep kernel has enough data to generalize well to unseen tasks without over-fitting. The second technique makes use of the Hypernetworks [10, 22, 21, 40, 43, 44], 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 combines the Hypernetworks paradigm with kernel methods to realize a new strategy that mimicks 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.

[*]The source code is available at: https://github.com/gmum/few-shot-hypernets-public.

[†]Denotes equal contribution.

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 [10, 39] – a neural network that takes kernel representation and produces decision rules in the form of a classifier. In our approach, the Hypernetwork aggregates the information from the support set and produces weights of the target model dedicated to the specific task, classifying the query set.

Our model, dubbed HyperShot, inherits the flexibility from Hypernetworks and the ability to learn the relation between objects from the kernel-based methods.

We perform an extensive experimental study of our approach 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.

The contributions of this work are three-fold:

- In this paper, we propose a model which 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, which combines the Hypernetworks paradigm with kernel methods to produce the weights dedicated for each task.

2. HyperShot: Hypernetwork for few-shot learning

In this section, we present our HyperShot model for few-shot learning.

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 [4, 38]. 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. Whereas, for *few-shot* scenarios, each class usually has from 2 to 5 representatives in the support set $\mathcal{S}$. Let:

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

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 [10], 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 hyper-network with a smaller number of parameters than the target network. Making an analogy between hyper-networks and generative models, the authors of [32] 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 $p(\mathbf{y}|S, \mathbf{x})$, given a query

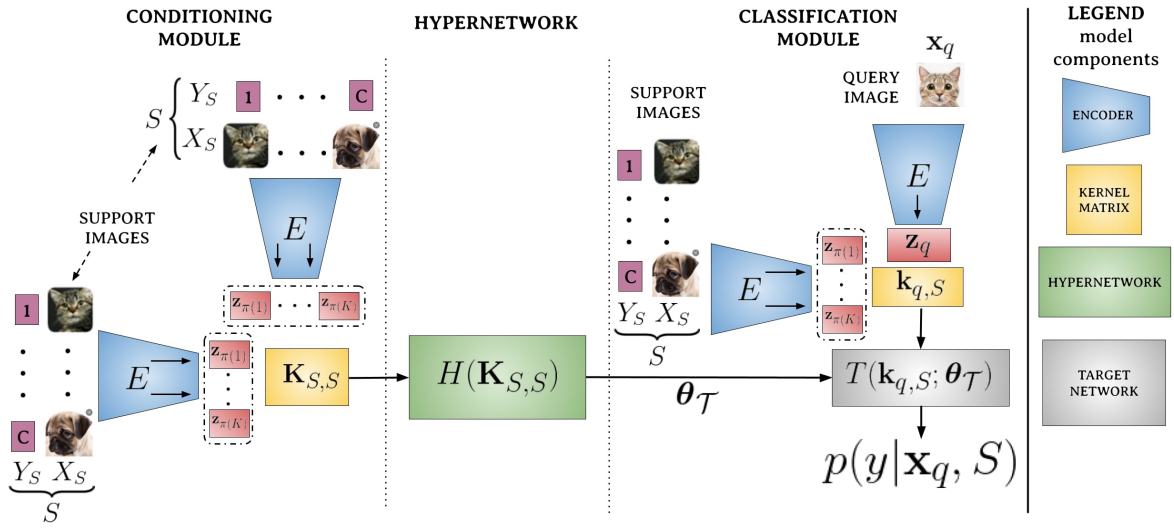

Figure 1. The general architecture of HyperShot. First, the examples from a support set are sorted according to the corresponding class labels and transformed by encoding network $E(\cdot)$ to obtain the matrix of ordered embeddings of the support examples, $\mathbf{Z}_S$. The low-dimensional representations 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 $\boldsymbol{\theta}_T$ for the target classification module $T(\cdot)$. 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.

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.

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 $p(\mathbf{y}|S,\mathbf{x})$ 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.

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

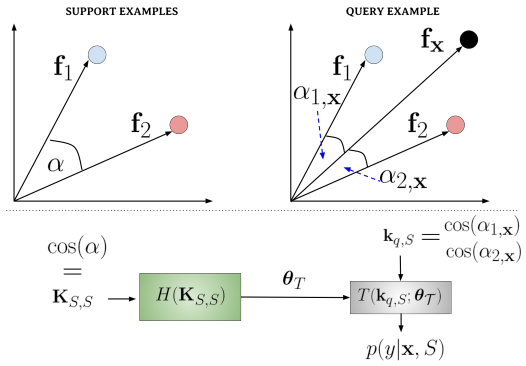

Figure 2. Simple 2D example illustrating the application of cosine kernel for HyperShot. We consider the two support examples from different classes represented by vectors $\mathbf{f_1}$ and $\mathbf{f_2}$. For this simple scenario, the input of hypernetwork is represented simply by the cosine of α, 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).

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 created 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:

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

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 $p(\mathbf{y}|\mathcal{S}, \mathbf{x})$ 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 unlabelled query examples represented by input values $\mathcal{X}_*$ that the model should classify. We can simply take the probability values $p(\mathbf{y}|\mathcal{S}_*, \mathbf{x})$ 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 are provided by Algorithm 1.

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 $\mathcal{S} = \{(\mathbf{x}_m, \mathbf{y}_m)\}_{m=1}^M$
    $L = -\sum_{m=1}^M \sum_{k=1}^K y_m^k \log p(y_m^k | \mathcal{S}_i, \mathbf{x}_m, \boldsymbol{\theta}_H, \boldsymbol{\theta}_k, \boldsymbol{\theta}_E)$
    Update: $\boldsymbol{\theta}_E \leftarrow \boldsymbol{\theta}_E - \alpha\nabla_{\boldsymbol{\theta}_E}\mathcal{L}$,
      $\boldsymbol{\theta}_H \leftarrow \boldsymbol{\theta}_H - \alpha\nabla_{\boldsymbol{\theta}_H}\mathcal{L}$,
      $\boldsymbol{\theta}_k \leftarrow \boldsymbol{\theta}_k - \alpha\nabla_{\boldsymbol{\theta}_k}\mathcal{L}$
    $n = n + 1$
  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 $\arg\max_\mathbf{y} p(\mathbf{y}|\mathcal{S}_*, \mathbf{x}, \hat{\boldsymbol{\theta}}_H, \hat{\boldsymbol{\theta}}_k, \hat{\boldsymbol{\theta}}_E)$
  end for
end function

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 [44] 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.

3. Related Work

Feature transfer [45] is a baseline procedure for few-shot learning and consists of pre-training the neural network and a classifier. During meta-validation, the classifier is fine-tuned to the novel tasks. [4] extend this idea by using cosine distance between the examples.

In recent years, a variety of meta-learning methods [3, 11, 30] 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:

Memory-based methods [15, 16, 17, 27, 29, 42] are based on the idea to train a meta-learner with memory to learn novel concepts.

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 [36] and Prototypical Networks [33], 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. [19] achieve this by conditioning the network on specific tasks, thus making the learned space task-dependent. [12] transform embeddings of support and query examples in the feature space to make their distributions closer to Gaussian. [35] propose Relation Nets, which learn the metric function instead of using a fixed one, such as Euclidean or cosine distance. Similar to the above methods, HyperShot uses a kernel function that predicts the relations between the examples in a given task. The key difference is that instead of performing a nearest-neighbor classification based on the kernel values, in HyperShot, the kernel matrix is classified by a task-specific classifier generated by the hypernetwork.

Optimization-based methods follow the idea of an optimization process over support set within the meta-learning framework like MetaOptNet [13], Model-Agnostic Meta-Learning (MAML), and its extensions [5, 18, 23, 25, 6, 18]. 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 also aims to produce task-specific models but utilizes a hypernetwork instead of optimization to achieve that goal.

Gaussian processes [26] 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, [20], 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. [39] introduced a technique of learning dense Gaussian processes by inducing variables. This approach achieves substantial performance improvement over the alternative methods. Similarly, HyperShot also depends on learning a model that estimates task-specific functions' parameters. However, HyperShot employs a hypernetwork instead of a Gaussian process to achieve that goal.

Hypernetworks [10] 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 hyper-networks that predict a shallow classifier's parameters given the support examples [2, 7, 22]. Subsequent works have extended those models by calculating cosine similarity between the query examples and the generated classifier weights [8] and utilizing a probabilistic model that predicts a distribution over the parameters suitable for the given task [9]. More recently, [21, 43, 44] 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 [40]. 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, HyperShot exploits those differences by utilizing kernels, which helps improve its performance.

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 the 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 [4].

In this section, we describe the experimental analysis and performance of the HyperShot in the large variety of few-shot benchmarks. Specifically, we consider both classification (see Section 4.1) and cross-domain adaptation (see Section 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. Additionally, we perform an ablation study of the possible adaptation procedures of HyperShot to few-shot scenarios, as well as architectural choices – presented in Section 4.3.

In all of the reported experiments, the tasks consist of 5 classes (5-way) and 1 or 5 support examples (1 or 5-shot). 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 non-linearity; each layer consists of 64 channels) [4] and have been trained from scratch.

We report the performance of two variants of HyperShot:

- **HyperShot** - models generated by the hypernetworks for each task.
- **HyperShot + adaptation** - models generated by hypernetworks adapted to the support examples of each task for 10 training steps[†].

In all cases, we observe a modest performance boost thanks to adapting the hypernetwork.

Comprehensive details for each training procedure are reported in the Appendix.

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 the HyperShot and other methods on two challenging and widely considered datasets: Caltech-USCD Birds (**CUB**) [37] and **mini-ImageNet** [27]. The following experiments are in the most popular setting, 5-way, consisting of 5 random classes. In all experiments, the query set of each task consists of 16 samples for each class (80 in total). We provide the additional training details in the Appendix. We compare HyperShot to a vast pool of the state-of-the-art algorithms, including the canonical methods (like Matching Networks [36], Prototypical Networks [33], MAML [5], and its extensions) as well as the recently popular Bayesian methods mostly build upon the Gaussian Processes framework (like DKT [20]).

We consider the more challenging 1-shot classification task, as well as the 5-shot setting and report the results in Table 1. The additional comparing methods on larger backbones, as well as a bigger number of baselines, are included in Appendix.

In the 1-shot scenario, HyperShot achieves the second and third-best accuracies in the **CUB** dataset with and without utilizing an adapting procedure (66.13% with adapting, 65.27% without) and performs better than any other model, except for FEAT [40] (68.87%). In the **mini-ImageNet** dataset, our approach is among the top approachess (53.18%), slightly losing with DFSVLwF [8] (56.20%).

Considering the 5-shot scenario, HyperShot is the third-best model achieving 80.07% in the **CUB** dataset and 69.62% in the **mini-ImageNet**, whereas the best model, FEAT [40], achieves 82.90% and 71.66% on the mentioned datasets, respectively.

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

4.2. Cross-domain adaptation

In the cross-domain adaptation setting, the model is evaluated on tasks coming from a different distribution than the one it 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 HyperShot 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:
mini-ImageNet $\rightarrow$ **CUB** (model trained on **mini-ImageNet** and evaluated on **CUB**) and **Omniglot** $\rightarrow$ **EMNIST** in the 1-shot and 5-shot settings. We report the results in Table 2. In most settings, HyperShot achieves the highest accuracy, except for 1-shot **mini-ImageNet** $\rightarrow$ **CUB** classification, where its accuracy is on par with the accuracy achieved by DKT [20] (40.14% and 40.03% achieved by DKT and HyperShot, respectively). We note that just like

[†]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. The classification accuracy results for the inference tasks on **CUB** and **mini-ImageNet** datasets in the 1-shot and 5-shot settings. The highest results are in bold and second-highest in italic (the larger, the better).

	CUB		mini-ImageNet	
Method	1-shot	5-shot	1-shot	5-shot
Feature Transfer [45]	46.19 ± 0.64	68.40 ± 0.79	39.51 ± 0.23	60.51 ± 0.55
Baseline++ [4]	61.75 ± 0.95	78.51 ± 0.59	47.15 ± 0.49	66.18 ± 0.18
MatchingNet [36]	60.19 ± 1.02	75.11 ± 0.35	48.25 ± 0.65	62.71 ± 0.44
ProtoNet [33]	52.52 ± 1.90	75.93 ± 0.46	44.19 ± 1.30	64.07 ± 0.65
RelationNet [35]	62.52 ± 0.34	78.22 ± 0.07	48.76 ± 0.17	64.20 ± 0.28
DKT + BNCosSim [20]	62.96 ± 0.62	77.76 ± 0.62	49.73 ± 0.07	64.00 ± 0.09
PPA [22]	–	–	54.53 ± 0.40	–
MAML [5]	56.11 ± 0.69	74.84 ± 0.62	45.39 ± 0.49	61.58 ± 0.53
MAML++ [1]	–	–	52.15 ± 0.26	68.32 ± 0.44
Bayesian MAML [41]	55.93 ± 0.71	–	53.80 ± 1.46	64.23 ± 0.69
Meta-SGD [14]	–	–	50.47 ± 1.87	64.03 ± 0.94
PAMELA [24]	–	–	53.50 ± 0.89	*70.51 ± 0.67*
FEAT [40]	$\mathbf{68.87 \pm 0.22}$	$\mathbf{82.90 \pm 0.15}$	*55.15 ± 0.20*	$\mathbf{71.61 \pm 0.16}$
DFSVLwF [8]	–	–	$\mathbf{56.20 \pm 0.86}$	–
HyperShot	65.27 ± 0.24	79.80 ± 0.16	52.42 ± 0.46	68.78 ± 0.29
HyperShot+ adaptation	*66.13 ± 0.26*	*80.07 ± 0.22*	53.18 ± 0.45	69.62 ± 0.20

in the case of regular classification, adapting the hypernetwork on the individual tasks consistently improves its performance.

4.3. Ablation study

In order to investigate different architectural choices in adapting HyperShot to the specific task, we provide a comprehensive ablation study. We focused mostly on the four major components of the HyperShot design, i.e., the method of processing multiple support examples per class, number of neck layers, the number of head layers, and the size of the hidden layers, presented in Tables 3, 4, and 5. In case of the experiments focusing on aggregating the number of support examples in the 5-way 5-shot setting, we perform the benchmarks on **CUB** and **mini-ImageNet**, using a 4-layer convolutional backbone. In the remaining experiments we tested HyperShot on the **CUB** dataset in the 5-way 1-shot setting with ResNet-10 backbone.

Aggregating support examples in the 5-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 5-way 1-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 [10], 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 [33].

To verify this approach, in the 5-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 5-shot classification task on **CUB** and **mini-ImageNet** datasets, as well as the task of cross-domain **Omniglot** $\rightarrow$ **EMNIST** classification. We report the accuracies in Table 3. It is evident that averaging the embeddings before calculating the kernel matrix yields superior results.

Hidden size: Firstly, as presented in Table 4, we compare different sizes of hidden layers. The results agree with the intuition that the wider layers, the better 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

Table 2. The classification accuracy results for the inference tasks on cross-domain tasks (**Omniglot→EMNIST** and **mini-ImageNet→CUB**) datasets in the 1-shot setting. The highest results are bold and second-highest in italic (the larger, the better).

	Omni→EMNIST		mini-ImageNet→CUB	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Feature Transfer	64.22 ± 1.24	86.10 ± 0.84	32.77 ± 0.35	50.34 ± 0.27
Baseline++ [4]	56.84 ± 0.91	80.01 ± 0.92	39.19 ± 0.12	57.31 ± 0.11
MatchingNet [36]	75.01 ± 2.09	87.41 ± 1.79	36.98 ± 0.06	50.72 ± 0.36
ProtoNet [33]	72.04 ± 0.82	87.22 ± 1.01	33.27 ± 1.09	52.16 ± 0.17
MAML [5]	72.68 ± 1.85	83.54 ± 1.79	34.01 ± 1.25	48.83 ± 0.62
RelationNet [35]	75.62 ± 1.00	87.84 ± 0.27	37.13 ± 0.20	51.76 ± 1.48
DKT [20]	75.40 ± 1.10	*90.30 ± 0.49*	$\mathbf{40.14 \pm 0.18}$	56.40 ± 1.34
Bayesian MAML [41]	63.94 ± 0.47	65.26 ± 0.30	33.52 ± 0.36	51.35 ± 0.16
OVE PG GP + Cosine (ML) [34]	68.43 ± 0.67	86.22 ± 0.20	39.66 ± 0.18	55.71 ± 0.31
OVE PG GP + Cosine (PL) [34]	77.00 ± 0.50	87.52 ± 0.19	37.49 ± 0.11	57.23 ± 0.31
HyperShot	*78.06 ± 0.24*	89.04 ± 0.18	39.09 ± 0.28	*57.77 ± 0.33*
HyperShot + adaptation	$\mathbf{80.65 \pm 0.30}$	$\mathbf{90.81 \pm 0.16}$	*40.034 ± 0.41*	$\mathbf{58.86 \pm 0.38}$

Table 3. The classification *accuracy* results for HyperShot in the 5-shot setting with two variants of the support embeddings aggregation. The performance measured on **Omniglot→EMNIST**, **CUB**, and **mini-ImageNet→CUB** tasks. The larger, the better.

	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

Table 4. Comparison between various hidden sizes in the HyperShot's layers. The classification *accuracy* results on **CUB** task and 5-way 1-shot setting. The larger, the better.

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

Table 5. Comparison between various HyperShot's architectures (different number of **neck layers** and **head layers**). The classification *accuracy* results on **CUB** task and 5-way 1-shot setting. The larger, the better.

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

for the achieved results, as presented in Table 5. 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.

5. Conclusion

In this work, we introduced HyperShot — a new framework that uses kernel methods combined with hypernetworks. Our method uses the kernel-based representation 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, our model can adapt to highly different tasks.

We evaluate the HyperShot model on various one-shot and few-shot image classification tasks. HyperShot demonstrates high accuracy in all tasks, performing comparably or better to state-of-the-art solutions. Furthermore, the model has a strong ability to generalize, as evidenced by its performance on cross-domain classification tasks.

Limitations The main limitation of HyperShot is the considerable (up to 10000 epochs) training time. This could possibly be reduced by employing a pre-training scheme similar to [28, 40]. We are planning to investigate this in the future.

Impact The results of this research show that Few-Shot methods which utilize kernels and hypernetworks achieve good performance in most benchmarks, particularly cross-domain classification. Thus, HyperShot and other similar models offer a promising direction in the research towards better generalization of Few-Shot models.

Acknowlegdements

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 and M. Sendera was supported by the National Centre of Science (Poland) Grant No. 2020/37/B/ST6/03463.

References

[1] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml, 2018.

[2] Matthias Bauer, Mateo Rojas-Carulla, Jakub Bartlomiej Świątkowski, Bernhard Schölkopf, and Richard E. Turner. Discriminative k-shot learning using probabilistic models, 2017.

[3] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. 1992.

[4] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[6] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 9537–9548, 2018.

[7] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional neural processes. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713. PMLR, 10–15 Jul 2018.

[8] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting, 2018.

[9] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard Turner. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2018.

[10] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[11] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey, 2020.

[12] Yuqing Hu, Vincent Gripon, and Stéphane Pateux. Leveraging the feature distribution in transfer-based few-shot learning, 2021.

[13] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[14] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning, 2017.

[15] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

[16] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *International Conference on Machine Learning*, pages 2554–2563. PMLR, 2017.

[17] Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *International Conference on Machine Learning*, pages 3664–3673. PMLR, 2018.

[18] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[19] Boris N Oreshkin, Pau Rodriguez, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. *arXiv preprint arXiv:1805.10123*, 2018.

[20] Massimiliano Patacchiola, Jack Turner, Elliot J Crowley, Michael O'Boyle, and Amos J Storkey. Bayesian meta-learning for the few-shot setting via deep kernels. *Advances in Neural Information Processing Systems*, 33, 2020.

[21] Toby Perrett, Alessandro Masullo, Tilo Burghardt, Majid Mirmehdi, and Dima Damen. Temporal-relational crosstransformers for few-shot action recognition. *CoRR*, abs/2101.06184, 2021.

[22] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan Yuille. Few-shot image recognition by predicting parameters from activations, 2017.

[23] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

[24] Jathushan Rajasegaran, Salman H. Khan, Munawar Hayat, Fahad Shahbaz Khan, and Mubarak Shah. Meta-learning the learning trends shared across tasks. *CoRR*, abs/2010.09291, 2020.

[25] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in Neural Information Processing Systems*, 32:113–124, 2019.

[26] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

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

[28] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization, 2019.

[29] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[30] Jürgen Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, 01 1992.

[31] Marcin Sendera, Jacek Tabor, Aleksandra Nowak, Andrzej Bedychaj, Massimiliano Patacchiola, Tomasz Trzciński, Przemysław Spurek, and Maciej Zięba. Non-gaussian gaussian processes for few-shot regression, 2021.

[32] Abdul-Saboor Sheikh, Kashif Rasul, Andreas Merentitis, and Urs Bergmann. Stochastic maximum likelihood optimization via hypernetworks. *arXiv preprint arXiv:1712.01141*, 2017.

[33] Jake Snell, Kevin Swersky, and Richard S Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

[34] Jake Snell and Richard Zemel. Bayesian few-shot classification with one-vs-each pólya-gamma augmented gaussian processes. In *International Conference on Learning Representations*, 2020.

[35] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1199–1208, 2018.

[36] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638, 2016.

[37] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[38] Yaqing Wang, Quanming Yao, James Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning, 2020.

[39] Ze Wang, Zichen Miao, Xiantong Zhen, and Qiang Qiu. Learning to learn dense gaussian processes for few-shot learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[40] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8808–8817, 2020.

[41] Jaesik Yoon, Taesup Kim, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7343–7353, 2018.

[42] Xiantong Zhen, Ying-Jun Du, Huan Xiong, Qiang Qiu, Cees Snoek, and Ling Shao. Learning to learn variational semantic memory. In *NeurIPS*, 2020.

[43] Andrey Zhmoginov, Mark Sandler, and Max Vladymyrov. Hypertransformer: Model generation for supervised and semi-supervised few-shot learning, 2022.

[44] Zhenxi Zhu, Limin Wang, Sheng Guo, and Gangshan Wu. A closer look at few-shot video classification: A new baseline and benchmark, 2021.

[45] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2020.