Deep Kernel Transfer in Gaussian Processes for Few-shot Learning

Massimiliano Patacchiola [1] **Jack Turner** [1] **Elliot J. Crowley** [1] **Michael O'Boyle** [1] **Amos Storkey** [1]

Abstract

Humans tackle new problems by making inferences that go far beyond the information available, reusing what they have previously learned, and weighing different alternatives in the face of uncertainty. Incorporating these abilities in an artificial system is a major objective in machine learning. Towards this goal, we adapt Gaussian Processes (GPs) to tackle the problem of few-shot learning. We propose a simple, yet effective variant of deep kernel learning in which the kernel is transferred across tasks, which we call *deep kernel transfer*. This approach is straightforward to implement, provides uncertainty quantification, and does not require estimation of task-specific parameters. We empirically demonstrate that the proposed method outperforms several state-of-the-art algorithms in few-shot regression, classification, and cross-domain adaptation. Code is available at: `https://github.com/BayesWatch/deep-kernel-transfer`.

1. Introduction

One of the key differences between state-of-the-art machine learning methods, such as deep learning (LeCun et al., 2015), and human learning is that the former needs a large amount of data in order to find relevant patterns across samples, whereas the latter acquires rich structural information from a handful of examples. Moreover, deep learning methods struggle in providing a measure of uncertainty, which is a crucial requirement to deal with scarce data, whereas humans can effectively weigh up different alternatives given limited evidence. In this regard, some authors have suggested that the human ability for few-shot inductive reasoning could derive from a Bayesian inference mechanism (Steyvers et al., 2006; Tenenbaum et al., 2011). Accordingly, we argue that a probabilistic treatment of few-shot learning is an indispensable prerequisite, and propose the use of Gaussian Processes (GPs, Rasmussen & Williams, 2006) as a framework for such a treatment.

GPs are a Bayesian non-parametric method representing distributions over functions, that work efficiently in the low-data regime and provide a measure of uncertainty with respect to new samples. Deep neural networks have been combined with GPs to provide powerful *deep kernels* as scalable and expressive closed-form covariance functions (Hinton & Salakhutdinov, 2008; Wilson et al., 2016). If one has a large number of small but related tasks, as in few-shot learning, it is possible to define a common prior that induces knowledge transfer. This prior can be a deep kernel with parameters shared across tasks, so that given a new unseen task it is possible to effectively estimate the posterior distribution over a query set conditioned on a small support set. This is our proposed approach, which we refer to as deep kernel learning with transfer, or *deep kernel transfer* for short.

GPs are well suited for the few-shot setting, although they have received scarce consideration in the few-shot literature. Methods developed for few-shot learning largely involve complicated meta-learning routines. However, recent work has demonstrated that simple baselines such as feature transfer (Chen et al., 2019) and nearest neighbors (Wang et al., 2019) can attain state-of-the-art performance on few-shot problems. In a similar vein, we show that a simple GP trained via kernel transfer is efficient in the few-shot regime and provides several advantages over standard methods, such as the ability to quantify uncertainty and demonstrate flexibility in cross-domain adaptation. This approach does not require a complex meta-learning routine, since both the hyperparameters of the GP and the weights of the neural network can be learned concurrently by maximizing the marginal likelihood. Our comparisons show that GPs obtain state-of-the-art results in few-shot regression and cross-domain classification, while being competitive in within-domain classification. Taken all together, our findings illustrate the importance of utilizing simple methods, especially when they are well suited to the problem at hand.

Our contributions are (i) providing a simple and principled approach to deal with the few-shot learning problem through the use of GPs, showcasing their strength across domains; (ii) introducing a robust method for dealing with few-shot regression and cross-domain classification, two challenging tasks that are less often considered in the literature.

[1]School of Informatics, University of Edinburgh. Correspondence to: Massimiliano Patacchiola <mpatacch@ed.ac.uk>.

2. Background

2.1. Few-shot Learning

The terminology describing the few-shot learning setup is dispersive; the reader is invited to see Chen et al. (2019) for a comparison. Here, we use the nomenclature derived from the meta-learning literature which is the most prevalent at time of writing. Let $\mathcal{S} = \{(x_l, y_l)\}_{l=1}^{L}$ be a *support-set* containing input-output pairs, with L equal to one (1-shot) or five (5-shot), and $\mathcal{Q} = \{(x_m, y_m)\}_{m=1}^{M}$ be a *query-set* (sometimes referred to in the literature as a target-set), with M typically one order of magnitude greater than L. For ease of notation the support and query sets are grouped in a *task* $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$, with the dataset $\mathcal{D} = \{\mathcal{T}_t\}_{t=1}^{N}$ defined as a collection of such tasks. Models are trained on random tasks sampled from $\mathcal{D}$, then given a new task $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{Q}_*\}$ sampled from a test set, the objective is to condition the model on the samples of the support $\mathcal{S}_*$ to estimate the membership of the samples in the query set $\mathcal{Q}_*$.

In the most common scenario, training, validation and test datasets each consist of distinct tasks sampled from the same overall distribution over tasks. Note that the target value y can be a continuous value (regression) or a discrete one (classification), though much previous work has focused on classification. We also consider the *cross-domain* scenario, where the test tasks are sampled from a different distribution over tasks than the training tasks; this is potentially more representative of many real-world scenarios.

2.2. Gaussian Processes

A GP is a real valued continuous random process; the collected values of this process at any finite set of points have a joint Gaussian distribution. A Gaussian process can be used as a model for functions (Rasmussen & Williams, 2006). GPs have been mainly used to tackle regression problems, however a treatment for classification is also possible (see Section 3). A GP is fully specified by a mean function $m(x)$ defined for all points x in our space, and a positive-definite covariance function $k(x, x')$ defined for all pairs x and x'. The implied distribution over functions f can be denoted by

$$f(\cdot) \sim \mathcal{GP}\left(m(\cdot), k(\cdot, \cdot)\right), \tag{1}$$

with the defining property that, at any points x and x',

$$m(x) = \mathbb{E}[f(x)], \tag{2a}$$
$$k(x, x') = \mathrm{cov}(f(x), f(x')). \tag{2b}$$

Typically, we do not have any prior knowledge about the mean $m(x)$ and therefore it is assumed to be zero. The covariance (or kernel) function $k(x, x')$ is a way to express how the correlation of the outputs at two points depends on the relationship between their two locations in input space.

More generally, given a set of training data $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ where x_n is the input for datapoint n and y_n is the associated continuous variable, we assume that the output y_n has been generated by a process $f(x_n)$ corrupted by homoscedastic Gaussian noise ϵ_n with variance σ^2:

$$y_n = f(x_n) + \epsilon_n, \quad \text{with} \quad \epsilon_n \sim \mathcal{N}(\epsilon_n | 0, \sigma^2). \tag{3}$$

To keep the notation uncluttered, we stack inputs, outputs, and generating processes in three vectors $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{f}$. Since the noise is independent for each data point, the joint distribution of the target values $\mathbf{y}$ conditioned on the values of $\mathbf{f}$ is given by the isotropic Gaussian

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma^2\mathbf{I}), \tag{4}$$

where $\mathbf{I}$ is an $N \times N$ identity matrix. By the definition of the GP the marginal distribution $p(\mathbf{y})$ is given by:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K} + \sigma^2\mathbf{I}), \quad \text{with} \quad K_{ij} = k(x_i, x_j). \tag{5}$$

The simplest kernel has a linear expression

$$k_{\text{LIN}}(x, x') = v\langle x, x' \rangle, \tag{6}$$

where $\langle \cdot \rangle$ denotes an inner product, and v is a variance hyperparameter. The use of a linear kernel is computationally convenient and it induces a form of Bayesian linear regression, however this is often too simplistic. For this reason, a variety of other kernels has been proposed in the literature. Common choices are: the Radial Basis Function kernel (RBF), defined via a squared Euclidean distance; the Matérn kernel, based on Bessel functions; the Cosine Similarity kernel (CosSim), which uses the cosine similarity as a distance metric; and the spectral mixture kernel (Wilson & Adams, 2013), derived from modeling a spectral density with a Gaussian mixture. Kernels can be combined applying some operations (e.g. sum, product, warping, etc) that preserve the positive definiteness of the covariance matrix. Details about the kernels used in this work are reported in Appendix A.

Our objective is to predict the clean signal f_* given a new input x_*, as we are interested in the joint distribution of the observed outputs and the function values at a test location. For ease of notation, let us define $\mathbf{k}_* = k(x_*, \mathbf{x})$ to denote the N-dimensional vector of covariances between x_* and the N training points in $\mathcal{D}$. Similarly, let us write $k_{**} = k(x_*, x_*)$ for the variance of x_*, and $\mathbf{K}$ to identify the $N \times N$ covariance matrix on the training inputs in $\mathcal{D}$. The predictive distribution $p(y_*|x_*, \mathcal{D})$ is obtained by Bayes' rule, and given the conjugacy of the prior, this is a Gaussian with mean and covariance specified as

$$\mathbb{E}[f_*] = \mathbf{k}_*^{\top}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \tag{7a}$$
$$\mathrm{cov}(f_*) = k_{**} - \mathbf{k}_*^{\top}(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}_*. \tag{7b}$$

Hereon, we absorb the noise $\sigma^2\mathbf{I}$ into the covariance matrix $\mathbf{K}$ and treat it as part of a vector of learnable parameters $\boldsymbol{\theta}$, that also include the hyperparameters of the kernel, e.g. the variance of the linear kernel defined in Equation (6).

Marginal likelihood (evidence). The fully Bayesian predictive distribution, taking account of hyperparameters θ, is given by the integral

$$p(y_*|x_*, \mathcal{D}) = \int p(y_*|x_*, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta} \tag{8}$$

which is typically intractable; many approximate schemes for computing this (e.g. MCMC sampling) are computationally expensive. A *maximum likelihood type II (ML-II)* alternative is to assume the posterior over $\boldsymbol{\theta}$ is dominated by the likelihood and sharply-peaked, meaning we can instead choose point parameters that maximize the *marginal likelihood*

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\mathcal{D}|\boldsymbol{\theta}). \tag{9}$$

Equation (8) can then be approximated by

$$p(y_*|x_*, \mathcal{D}) \approx p(y_*|x_*, \mathcal{D}, \hat{\boldsymbol{\theta}}). \tag{10}$$

Deep kernel learning. In deep kernel learning (Hinton & Salakhutdinov, 2008; Wilson et al., 2016) the input $\mathbf{x}$ is mapped to a latent vector $\mathbf{h}$ through a non-linear function $\mathcal{F}_\phi(\mathbf{x}) \rightarrow \mathbf{h}$ (e.g. a neural network) parameterized by a set of weights $\boldsymbol{\phi}$. The embedding is defined such that the dimensionality of the input is significantly reduced, meaning that if $\mathbf{x} \in \mathbb{R}^J$ and $\mathbf{h} \in \mathbb{R}^K$ then $J \gg K$. Once the input has been encoded in $\mathbf{h}$ the latent vector is passed to the GP to perform regression (or classification). When the inputs are images a common choice for $\mathcal{F}_\phi$ is a Convolutional Neural Network (CNN). The parameters of the model are learned through ML-II following the same procedure adopted for the hyperparameters of the kernel. Specifically we construct a kernel

$$k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}, \boldsymbol{\phi}) = k'(\mathcal{F}_\phi(\mathbf{x}), \mathcal{F}_\phi(\mathbf{x}')|\boldsymbol{\theta}) \tag{11}$$

from some latent space kernel k' with hyperparameters $\boldsymbol{\theta}$ by passing the inputs through the non-linear function $\mathcal{F}_\phi$. The hyperparameters $\boldsymbol{\theta}$ and the parameters of the model $\boldsymbol{\phi}$ are jointly learned by maximizing the log marginal likelihood as previously described in Equation (9). This is achieved by updating the weights of the CNN by backpropagating the error.

3. The Method

The Bayesian approach to few-shot learning has predominantly followed the route of hierarchical modeling and multi-task learning (Finn et al., 2018; Gordon et al., 2019; Yoon et al., 2018). The underlying directed graphical model distinguishes between a set of shared parameters $\boldsymbol{\theta}$, common to all tasks, and a set of task-specific parameters $\boldsymbol{\psi}_n$. Learning consists of finding an estimate of $\boldsymbol{\theta}$, forming the posterior distribution over the task-specific parameters $p(\boldsymbol{\psi}_n|x_*, \mathcal{D}, \boldsymbol{\theta})$, and then computing the posterior predictive distribution $p(y_*|x_*, \boldsymbol{\theta})$. This approach is principled from a probabilistic perspective, but is problematic, as it requires managing two levels of inference via amortized distributions or sampling, often requiring cumbersome architectures. The graphical model of hierarchical learning is shown in Figure 1a.

In order to avoid these drawbacks, we propose a simpler solution. We absorb the task-specific parameters into a latent variable z (see Figure 1b), which is kept disconnected from the shared parameters, and we exclusively focus on finding $\boldsymbol{\theta}$ (marginalizing out z). With this simplification there is no need to estimate the posterior distribution over the task-specific parameters, meaning that it is possible to directly compute the posterior predictive $p(y_*|x_*, \mathcal{D}, \boldsymbol{\theta})$ skipping an intermediate inference step. We argue that this approach can be very effective in the few-shot setting, especially when coupled with the GP framework.

With GPs, the reasoning is similar. In this setting, the latent z (as Figure 1b) determines the particular sample of Gaussian process function. Marginalizing over z for the data for a specific task is analytic and leads to the Gaussian process marginal likelihood. We wish to find the parameters of a (deep) kernel that has the highest marginal likelihood. Given data $\mathbf{y}$, the marginal likelihood measures the *expectedness* of the data under the given set of parameters. Let all the input data (support and query) for task t be denoted by $\mathcal{T}_t^x$ and the target data be $\mathcal{T}_t^y$. Let $\mathcal{D}^x$ and $\mathcal{D}^y$ denote the respective collections of these datasets over all tasks. Collect all the target data items for task t into vector $\mathbf{y}_t$, and denote the kernel between all task inputs by K_t: this kernel depends on current estimates for the hyperparameters $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ and weights $\boldsymbol{\phi} = \hat{\boldsymbol{\phi}}$. Then the full marginal likelihood over all tasks is given by

$$\begin{aligned} \log P(\mathcal{D}^y|\mathcal{D}^x, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) &= \sum_t \log P(\mathcal{T}_t^y|\mathcal{T}_t^x, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) \\ = \sum_t - &\underbrace{\frac{1}{2}\mathbf{y}_t^\top [K_t(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}})]^{-1}\mathbf{y}_t}_{\text{data-fit}} - \underbrace{\frac{1}{2}\log|K_t(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}})|}_{\text{penalty}} + c, \end{aligned} \tag{12}$$

where c is a constant. The parameters are estimated via ML-II by maximizing (12) using gradient ascent. In practice we use a stochastic gradient ascent with each batch containing the data for a single task.

Our claim is that, though the number of data points for each task is potentially small (resulting in low cost Gaussian process computations), the total number of points over all tasks contributing to the marginal likelihood (12) is sufficiently large to make ML-II appropriate for finding a set of shared weights and parameters without underfitting or overfitting. Those parameters provide a model with good generalization capability to new unseen tasks. This removes the need for inferring the task-specific parameters ψ_n. Our claim will be substantiated by the results reported in Section 5. Note that this approach differs from direct deep kernel learning (Figure 1c), where the marginalization is over all data, ignoring the task distinctions. The problem also differs from multitask learning where the tasks typically share the same input values.

For few-shot learning, the Bayesian selection principle is applied *across tasks* over the dataset $\mathcal{D} = \{\mathcal{T}_t\}_{t=1}^{N}$. More precisely, we assume that the same hyperparameters $\boldsymbol{\theta}$ and weights $\boldsymbol{\phi}$ are shared across all datapoints in the support $\mathcal{S} = \{(x_l, y_l)\}_{l=1}^{L}$ and query $\mathcal{Q} = \{(x_m, y_m)\}_{m=1}^{M}$ sets belonging to each task. What differs between tasks is the precise function sampled from the Gaussian process. Therefore, once a common prior has been found, knowledge can be transferred.

For stochastic gradient training, at each iteration, a task $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\}$ is sampled from $\mathcal{D}$, then the log marginal likelihood of Equation (12) is estimated over $\mathcal{S} \cup \mathcal{Q}$ (assuming $y \in Q$ to be observed) and the parameters of the GP are updated via a gradient step on the marginal likelihood objective for that task. This procedure allows us to find a kernel that can represent the task in its entirety over both support and query sets.

At test time, given a new task $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{Q}_*\}$ the prediction on the query set $\mathcal{Q}_*$ is made via conditioning on the support set $\mathcal{S}_*$, using the parameters that have been learned at training time. The pseudocode for this process is given in Algorithm 1.

Algorithm 1 Few-shot GP train and test procedures

```
Require: $\mathcal{D} = \{\mathcal{T}_n\}_{n=1}^{N}$ train dataset
Require: $\mathcal{T}_* = \{\mathcal{S}_*, \mathcal{Q}_*\}$ test task
Require: $\boldsymbol{\theta}$ , $\boldsymbol{\phi}$: GP hyperparameters, Net weights
Require: $\alpha$ , $\beta$: step size hyperparameters
 1: procedure TRAIN($\mathcal{D}$, $\alpha$, $\beta$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}$)
 2:     while not done do
 3:         Sample task $\mathcal{T} = \{\mathcal{S}, \mathcal{Q}\} \sim \mathcal{D}$
 4:         Assign $\mathbf{x} \leftarrow \forall x \in \mathcal{S} \cup \mathcal{Q}$, $\mathbf{y} \leftarrow \forall y \in \mathcal{S} \cup \mathcal{Q}$
 5:         Estimate loss $\mathcal{L} = -\log p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}, \boldsymbol{\phi})$      ▷ Eq. (12)
 6:         Update GP $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{S} \cup \mathcal{Q}}$
 7:         Update Net $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \beta \nabla_{\phi} \mathcal{L}_{\mathcal{S} \cup \mathcal{Q}}$
 8:     end while
 9: end procedure
10: procedure TEST($\mathcal{T}_*$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}$)
11:     Assign $\mathbf{x} \leftarrow \forall x \in \mathcal{S}_*$, $\mathbf{y} \leftarrow \forall y \in \mathcal{S}_*$
12:     Assign $\mathbf{x}_* \leftarrow \forall x \in \mathcal{Q}_*$
13:     Estimate $\mathbb{E}(\mathbf{f}_*)$ and cov($\mathbf{f}_*$)      ▷ Eq. (7)
14: end procedure
```

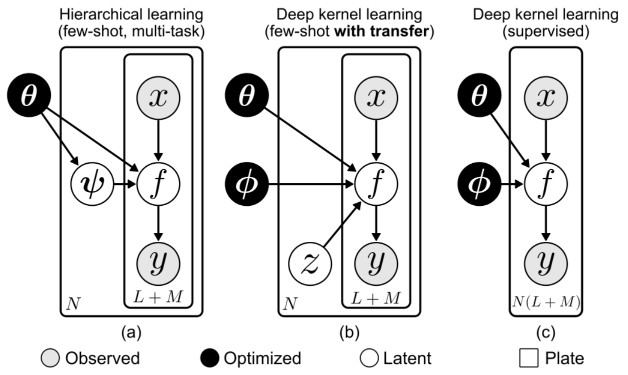
Hierarchical learning (few-shot, multi-task)
Deep kernel learning (few-shot with transfer)
Deep kernel learning (supervised)
θ
ψ
ϕ
x
f
y
z
N
$L+M$
$N(L+M)$
(a)
(b)
(c)
Observed
Optimized
Latent
Plate

Figure 1. Graphical models of: (a) hierarchical learning, (b) our method, and (c) fully supervised learning. The plate indicates that the underlying nodes are repeated with edges preserved. N is the number of tasks in the training set, L and M are the number of elements in the support and query sets for each task. In deep kernel learning for few-shot, the z denotes the choice of sample of function f from the GP, with kernel determined by $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

It is possible to redefine the GP framework such that the same treatment discussed for regression can be extended to classification. However, this does not come without problems, since a non-Gaussian likelihood breaks the conjugacy. For instance, in the case of binary classification the Bernoulli likelihood induces an intractable marginalization of the evidence and therefore it is not possible to estimate the posterior in a closed form. The common approach to deal with this issue is to draw samples directly from the posterior through MCMC, or approximate it through variational methods. However, these solutions incur a significant computational cost for few-shot learning: for each new task, the posterior is estimated by approximation or sampling, introducing an inner loop that increases the time complexity from constant $\mathcal{O}(1)$ to linear $\mathcal{O}(K)$, with K being the number of inner cycles. An alternative solution would be to treat the classification problem as if it were a regression one, therefore reverting to analytical expressions for both the evidence and the posterior. In the literature this has been called *label regression (LR)* (Kuss, 2006) or *least-squares classification (LSC)* (Rifkin & Klautau, 2004; Rasmussen & Williams, 2006). Experimentally, LR and LSC tend to be more effective than other approaches in both binary (Kuss, 2006) and multi-class (Rifkin & Klautau, 2004) settings. Here, we derive a classifier based on LR which is computationally cheap and straightforward to implement.

The starting point is binary classification with the class being a Bernoulli random variable $c \in \{0, 1\}$. The GP is trained as a regressor with a target $y_+ = 1$ to denote the case $c = 1$, and $y_- = -1$ to denote the case $c = 0$. Even though $y \in \{-1, 1\}$ there is no guarantee that $f(x) \in [y_-, y_+]$. Predictions are made by computing the predictive mean and passing it through a sigmoid function, inducing a probabilistic interpretation. Note that it is still possible to use ML-II to make point estimates of $\boldsymbol{\theta}$ and ϕ.

When generalizing from a binary to a multi-label task it is possible to apply the *one-versus-rest* scheme where C binary classifiers are used to classify each class against all the rest. The log marginal likelihood of Equation (12) is replaced by the sum of the marginals for each one of the C individual class outputs $\mathbf{y}_c$, as

$$\log p(\mathbf{y}|\mathbf{x}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) = \sum_{c=1}^{C} \log p(\mathbf{y}_c|\mathbf{x}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}). \tag{13}$$

Given a new input x_* and the C outputs of all the binary classifiers, a decision is made by selecting the output with the highest probability $c_* = \operatorname{argmax}_c\big(\sigma(m_c(x_*))\big)$, where the predictive mean $m(x)$ has been previously defined in Equations (2) and (7), $\sigma(\cdot)$ is the sigmoid function, and $c_* \in \{1, ..., C\}$.

4. Related Work

There is a wealth of literature on feature transfer (Pan & Yang, 2009). As a baseline for few-shot learning, the standard procedure consists of two phases: pre-training and fine-tuning. During pre-training, a network and classifier are trained on examples for the base classes. When fine-tuning, the network parameters are fixed and a new classifier is trained on the novel classes. This approach has its limitations; part of the model has to be trained from scratch for each new task, and often overfits. Chen et al. (2019) extend this by proposing the use of cosine distance between examples (called Baseline++). However, this still relies on the assumption that a fixed fine-tuning protocol will balance the bias-variance tradeoff correctly for every task.

Alternatively, one can compare new examples in a learned metric space. Matching Networks (MatchingNets, Vinyals et al., 2016) use a softmax over cosine distances as an attention mechanism, and an LSTM to encode the input in the context of the support set, considered as a sequence. Prototypical Networks (ProtoNets, Snell et al., 2017) are based on learning a metric space in which classification is performed by computing distances to prototypes, where each prototype is the mean vector of the embedded support points belonging to its class. Relation Networks (RelationNets, Sung et al., 2018) use an embedding module to generate representations of the query images that are compared by a relation module with the support set, to identify matching categories.

Meta-learning (Bengio et al., 1992; Schmidhuber, 1992) methods have become very popular for few-shot learning tasks. MAML (Finn et al., 2017) has been proposed as a way to meta-learn the parameters of a model over many tasks, so that the initial parameters are a good starting point from which to adapt to a new task. MAML has provided inspiration for numerous meta-learning approaches (Antoniou et al., 2019; Rajeswaran et al., 2019).

Gordon et al. (2019) have recently proposed an amortization network—VERSA—that takes few-shot learning datasets as inputs, and outputs a distribution over task-specific parameters which can be used to meta-learn probabilistic inference for prediction. In several works, MAML has been interpreted as a Bayesian hierarchical model (Finn et al., 2018; Grant et al., 2018; Jerfel et al., 2019). Bayesian MAML (Yoon et al., 2018) combines efficient gradient-based meta-learning with nonparametric variational inference, while keeping an application-agnostic approach. Concurrent to our work, Tossou et al. (2019) have presented a variant of kernel learning for GPs called Adaptive Deep Kernel Learning (ADKL), which is based on the principle of finding an appropriate kernel for each task using a task encoder network. The difference between our method and ADKL is that we use the same kernel for all tasks, and therefore do not need an additional module for task encoding as we can rely on a single set of shared general-purpose hyperparameters.

5. Experiments

In the few-shot setting a fair comparison between methods is often obfuscated by substantial differences in the implementation details of each algorithm. Chen et al. (2019) have recently investigated this issue, releasing an open-source benchmark to allow for a fair comparison between methods. We integrated our algorithm into this framework using PyTorch and GPyTorch (Gardner et al., 2018). In classification and cross-domain experiments, each method uses the same backbone (a four layer CNN), optimizer (Adam), and learning rate (10^{-3}). For head pose regression we reduce this to a three layer CNN, and for wave regression we use a two layer MLP. We use shallow backbones because they have been shown to highlight differences between methods (Chen et al., 2019). In all experiments the proposed method is marked as *GPShot*. Training details are reported in Appendix B.

5.1. Regression

We perform regression experiments on two tasks: amplitude prediction for unknown periodic functions, and head pose trajectory estimation from images. The former was treated as a few-shot regression problem by Finn et al. (2017) to motivate MAML: support and query scalars are uniformly sampled from a periodic wave with amplitude $\in [0.1, 5.0]$, phase $\in [0, \pi]$, and range $\in [-5.0, 5.0]$, and Gaussian noise ($\mu = 0$, $\sigma = 0.1$). The training set is composed of 5 support and 5 query points, and the test set is composed of 5 support and 200 query points. We first test *in-range*: the same domain as the training set as in Finn et al. (2017). We also consider *out-of-range* regression, with test points drawn

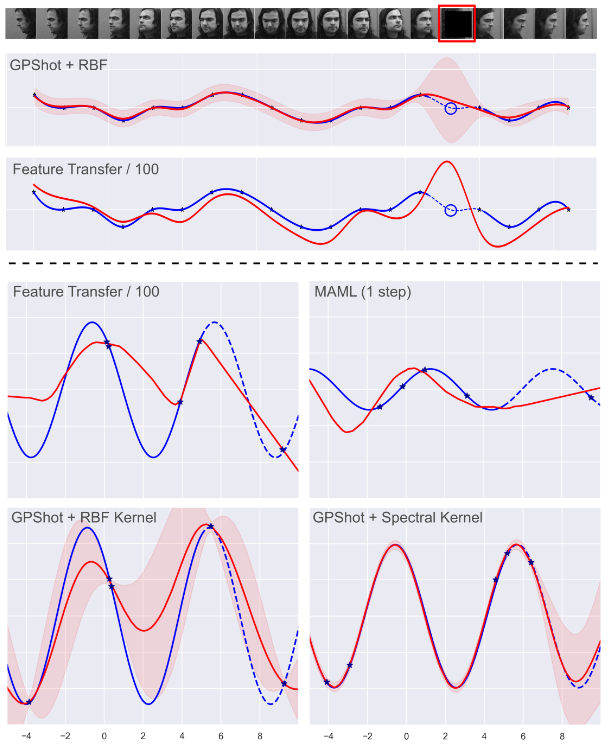

Figure 2. Regression experiments. [top] Uncertainty estimation for an outlier (Cutout noise, red frame) in the head trajectory estimation. GPShot is able to estimate a mean value (red line) close to the true value (blue circle) showing large variance. Feature transfer performs poorly at the same location. [bottom] Qualitative comparison between different methods in the unknown function approximation (out-of-range), given 5 support points. GPShot better fits (red line) the true function (solid blue) and the out-of-bound portion never seen at training time (dashed blue). Uncertainty (red shadow) increases in low-confidence regions.

from an extended domain $[-5.0, 10.0]$ where portions from the range $[5.0, 10.0]$ have *not been seen* at training time.

For head pose regression, we used the Queen Mary University of London multiview face dataset (QMUL, Gong et al., 1996), it comprises of grayscale face images of 37 people (32 train, 5 test). There are 133 facial images per person, covering a viewsphere of $\pm 90°$ in yaw and $\pm 30°$ in tilt at $10°$ increment. Each task consists of randomly sampled trajectories taken from this discrete manifold, where *in-range* includes the full manifold and *out-of-range* allows training only on the leftmost 10 angles, and testing on the full manifold; the goal is to predict tilt. To highlight the benefits of our method versus other approaches, we perform an experiment on *quantifying uncertainty*, sampling head pose trajectories and corrupting one input with Cutout (DeVries & Taylor, 2017), randomly covering 95% of the image.

For the periodic function prediction experiment, we compare our approach against feature transfer and MAML (Finn

Table 1. Average Mean-Squared Error (MSE) and standard deviation over three runs on few-shot regression for periodic functions [top] and head pose trajectory [bottom], using 10 samples for train, and 5 samples for test. Same domain is marked as *in-range*, extended unseen domain as *out-of-range*. Our method (GPShot) has the lowest error (highlighted in bold).

Method	**in-range**	**out-of-range**
	Periodic functions	
ADKL (Tossou et al., 2019)	0.14	–
R2-D2 (Bertinetto et al., 2019)	0.46	–
Feature Transfer/1	2.94 ± 0.16	6.13 ± 0.76
Feature Transfer/100	2.67 ± 0.15	6.94 ± 0.97
MAML (1 step)	2.76 ± 0.06	8.45 ± 0.25
GPBaseline + RBF	2.85 ± 1.14	3.65 ± 1.63
GPBaseline + Spectral	2.08 ± 2.31	4.11 ± 1.92
GPShot + RBF [ours]	1.38 ± 0.03	2.61 ± 0.16
GPShot + Spectral [ours]	$\mathbf{0.08 \pm 0.06}$	$\mathbf{0.10 \pm 0.06}$
	Head pose trajectory	
Feature Transfer/1	0.25 ± 0.04	0.20 ± 0.01
Feature Transfer/100	0.22 ± 0.03	0.18 ± 0.01
MAML (1 step)	0.21 ± 0.01	0.18 ± 0.02
GPShot + RBF [ours]	0.12 ± 0.04	0.14 ± 0.03
GPShot + Spectral [ours]	$\mathbf{0.10 \pm 0.01}$	$\mathbf{0.11 \pm 0.02}$

et al., 2017). Moreover we report the results of ADKL (Tossou et al., 2019), and R2-D2 (Bertinetto et al., 2019) obtained on a similar task (as defined in Yoon et al., 2018). To highlight the importance of kernel transfer, we include a baseline condition where a GP is trained from scratch on the support points of every incoming task (GPBaseline). Few methods have tackled few-shot regression from images, so in the head pose trajectory estimation we compare against feature transfer and MAML. For regression with feature transfer, a network is trained to predict the output of a function over all tasks, before being fine-tuned on a new task (with 1 or 100 steps of size 10^{-3}). MAML is one of the few methods that can deal with both regression and classification. Models are compared using the average Mean-Squared Error (MSE) between predictions and true values. Additional details on the training setup are reported in Appendix B.

Results for the regression experiments are summarized in Table 1 and a qualitative comparison is provided in Figure 2. The proposed method (GPShot) obtains a lower MSE than feature transfer and MAML on both experiments. For unknown periodic function estimation, using a spectral kernel gives a large advantage over RBF, being more precise in both in-range and out-of-range (1.38 vs 0.08, and 2.61 vs 0.10 MSE). Uncertainty is correctly estimated in regions with low point density, and increases overall in the out-of-range region. Conversely, feature transfer severely underfits (1 step, 2.94 MSE) or overfits (100 step, 2.67), and was unable to model out-of-range points (6.13 and 6.94). MAML is effective in-range (2.76), but significantly worse out-of-

Table 2. Average accuracy and standard deviation (percentage) on the few-shot classification setting (5-ways). [top] Results reported in recent literature. For a fair comparison we selected only those methods that have been trained with a similar backbone and training schedule. [center-bottom] Methods trained from scratch (three runs) with the same backbone (a four layer CNN), optimizer (Adam), and learning rate (10^{-3}). Test performed on novel classes with 3000 randomly generated tasks. The proposed method (GPShot) is competitive across various datasets and conditions. Best results highlighted in bold. *Reported by Jerfel et al. (2019) using a comparable backbone.

	CUB		**mini-ImageNet**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
ML-LSTM (Ravi & Larochelle, 2017)	–	–	43.44 ± 0.77	60.60 ± 0.71
SNAIL (Mishra et al., 2018)	–	–	45.10	55.20
iMAML-HF (Rajeswaran et al., 2019)	–	–	49.30 ± 1.88	–
LLAMA (Grant et al., 2018)	–	–	49.40 ± 1.83	–
VERSA (Gordon et al., 2019)*	–	–	48.53 ± 1.84	–
Amortized VI (Gordon et al., 2019)	–	–	44.13 ± 1.78	55.68 ± 0.91
Meta-Mixture (Jerfel et al., 2019)	–	–	49.60 ± 1.50	64.60 ± 0.92
SimpleShot (Wang et al., 2019)	–	–	49.69 ± 0.19	$\mathbf{66.92 \pm 0.17}$
Feature Transfer	46.19 ± 0.64	68.40 ± 0.79	39.51 ± 0.23	60.51 ± 0.55
Baseline++ (Chen et al., 2019)	61.75 ± 0.95	$\mathbf{78.51 \pm 0.59}$	47.15 ± 0.49	66.18 ± 0.18
MatchingNet (Vinyals et al., 2016)	60.19 ± 1.02	75.11 ± 0.35	48.25 ± 0.65	62.71 ± 0.44
ProtoNet (Snell et al., 2017)	52.52 ± 1.90	75.93 ± 0.46	44.19 ± 1.30	64.07 ± 0.65
MAML (Finn et al., 2017)	56.11 ± 0.69	74.84 ± 0.62	45.39 ± 0.49	61.58 ± 0.53
RelationNet (Sung et al., 2018)	62.52 ± 0.34	78.22 ± 0.07	48.76 ± 0.17	64.20 ± 0.28
GPShot + CosSim [ours]	$\mathbf{63.37 \pm 0.19}$	77.73 ± 0.26	48.64 ± 0.45	62.85 ± 0.37
GPShot + BNCosSim [ours]	62.96 ± 0.62	77.76 ± 0.62	$\mathbf{49.73 \pm 0.07}$	64.00 ± 0.09

Table 3. Average accuracy and standard deviation (percentage) over three runs on 1-shot and 5-shot classification (5-ways), for different backbones in the CUB dataset. We use the same setup as in the classification setting. The results for the ResNet are the ones reported in Chen et al. (2019). The proposed method (GPShot) has the best score in 1-shot Conv-4, and 5-shot ResNet, while being competitive in the other conditions. Best results highlighted in bold.

	Conv-4		**ResNet-10**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Feature Transfer	46.19 ± 0.64	68.40 ± 0.79	63.64 ± 0.91	81.27 ± 0.57
Baseline++ (Chen et al., 2019)	61.75 ± 0.95	$\mathbf{78.51 \pm 0.59}$	69.55 ± 0.89	85.17 ± 0.50
MatchingNet (Vinyals et al., 2016)	60.19 ± 1.02	75.11 ± 0.35	71.29 ± 0.87	83.47 ± 0.58
ProtoNet (Snell et al., 2017)	52.52 ± 1.90	75.93 ± 0.46	$\mathbf{73.22 \pm 0.92}$	85.01 ± 0.52
MAML (Finn et al., 2017)	56.11 ± 0.69	74.84 ± 0.62	70.32 ± 0.99	80.93 ± 0.71
RelationNet (Sung et al., 2018)	62.52 ± 0.34	78.22 ± 0.07	70.47 ± 0.99	83.70 ± 0.55
GPShot + CosSim [ours]	$\mathbf{63.37 \pm 0.19}$	77.73 ± 0.26	70.81 ± 0.52	83.26 ± 0.50
GPShot + BNCosSim [ours]	62.96 ± 0.62	77.76 ± 0.62	72.27 ± 0.30	$\mathbf{85.64 \pm 0.29}$

range (8.45). ADKL and R2-D2 (0.14 and 0.46) are better than GPShot with an RBF kernel (1.38), but worse than GPShot with a Spectral kernel (0.08). This indicates that the combination of an appropriate kernel with our method is more effective than an adaptive approach. The GP baseline performs significantly worse than GPShot in all conditions, confirming the necessity of using kernel transfer for few-shot problems. Figure 2(bottom) shows that both feature transfer and MAML are unable to fit the true function, especially out-of-range; additional samples are reported in Appendix C. We observe similar results for head pose estimation, with GPShot reporting lower MSE in all cases (Table 1). Qualitative results for the uncertainty quantification experiment are shown in Figure 2(top). For the corrupted input, GPShot predicts a value close to the true one, while giving a high level of uncertainty (red shadow). Feature transfer performs poorly, predicting an unrealistic pose. In Appendix C we examine the latent spaces generated by the RBF and spectral kernel.

5.2. Classification

We perform few-shot classification on two challenging datasets: the Caltech-UCSD Birds (CUB-200, Wah et al.,

Table 4. Average accuracy and standard deviation (percentage) over three runs on the cross-domain setting (5-ways). We use the same setup as in the classification setting. The proposed method (GPShot) has the best score on most conditions. Best results highlighted in bold.

	Omniglot→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++ (Chen et al., 2019)	56.84 ± 0.91	80.01 ± 0.92	39.19 ± 0.12	$\mathbf{57.31 \pm 0.11}$
MatchingNet (Vinyals et al., 2016)	75.01 ± 2.09	87.41 ± 1.79	36.98 ± 0.06	50.72 ± 0.36
ProtoNet (Snell et al., 2017)	72.04 ± 0.82	87.22 ± 1.01	33.27 ± 1.09	52.16 ± 0.17
MAML (Finn et al., 2017)	72.68 ± 1.85	83.54 ± 1.79	34.01 ± 1.25	48.83 ± 0.62
RelationNet (Sung et al., 2018)	75.62 ± 1.00	87.84 ± 0.27	37.13 ± 0.20	51.76 ± 1.48
GPShot + Linear [ours]	$\mathbf{75.97 \pm 0.70}$	89.51 ± 0.44	38.72 ± 0.42	54.20 ± 0.37
GPShot + CosSim [ours]	73.06 ± 2.36	88.10 ± 0.78	$\mathbf{40.22 \pm 0.54}$	55.65 ± 0.05
GPShot + BNCosSim [ours]	75.40 ± 1.10	$\mathbf{90.30 \pm 0.49}$	40.14 ± 0.18	56.40 ± 1.34

2011), and mini-ImageNet (Ravi & Larochelle, 2017). CUB is widely used for fine-grained classification, and it consists of 11788 images across 200 classes. We divide the dataset in 100 classes for train, 50 for validation, and 50 for test (Hilliard et al., 2018; Chen et al., 2019). The mini-ImageNet dataset consists of a subset of 100 classes (600 images for each class) taken from the ImageNet dataset (Russakovsky et al., 2015). We use 64 classes for train, 16 for validation and 20 for test, as is common practice (Ravi & Larochelle, 2017; Chen et al., 2019). All the experiments are 5-way (5 random classes) with 1 or 5-shot (1 or 5 samples per class in the support set). A total of 16 samples per class are provided for the query set.

We compare several kernels: linear, RBF, Matérn, Polynomial, CosSim, and BNCosSim. Where BNCosSim is a variant of CosSim with features centered through BatchNorm (BN) statistics (Ioffe & Szegedy, 2015), this has shown to improve performance (Wang et al., 2019). We compare our approach to several state-of-the-art methods, such as MAML (Finn et al., 2017), ProtoNets (Snell et al., 2017), MatchingNet (Vinyals et al., 2016), and RelationNet (Sung et al., 2018). We further compare against feature transfer, and Baseline++ from Chen et al. (2019). All these methods have been trained from scratch with the same backbone and learning schedule. We additionally report the results for approaches with comparable training procedures and convolutional architectures (Mishra et al., 2018; Ravi & Larochelle, 2017; Wang et al., 2019) including recent hierarchical Bayesian methods (Gordon et al., 2019; Grant et al., 2018; Jerfel et al., 2019). We have excluded approaches that use deeper backbones or more sophisticated learning schedules (Antoniou & Storkey, 2019; Oreshkin et al., 2018; Qiao et al., 2018; Ye et al., 2018) so that the quality of the algorithms can be assessed separately from the power of the underlying discriminative model.

Results are reported in Table 2 (average accuracy as a percentage). GPShot achieves the highest accuracy in CUB 1-shot (63.37%), being competitive in 5-shot (77.76%). In mini-ImageNet 1-shot, GPShot with BNCosSim performs better than any other approach (49.73%), including hierarchical Bayesian methods such as LLAMA (49.40%) and VERSA (48.53%). In mini-ImageNet 5-shot the average accuracy of GPShot (64.00%) is in line with other state-of-the-art methods such as ProtoNets (64.07%) and RelationNets (64.20%) but less effective than Baseline++ (66.18%) and SimpleShot (66.92%). Across different kernels, those with first-order covariance functions are the best overall, as showed in the kernel comparison reported in Appendix D, Table 5. This is likely due to a low-curvature manifold induced by the neural network in the latent space, increasing the linear separability of data. Overall our results confirm the findings of Chen et al. (2019) regarding the effectiveness of cosine metrics, and those of Wang et al. (2019) on the importance of feature normalization (Appendix D and E).

In Table 3 we report additional tests on CUB (5-ways) using a ResNet-10 (He et al., 2016). The results show that GPShot performs well with deeper backbones, outperforming all other methods in 5-shot (85.64%) and obtaining the second best result in 1-shot (72.27%). The difference in performance between CosSim and BNCosSim is larger for the deeper backbone, indicating that centering the features is particularly important when additional layers are added to the network.

5.3. Cross-domain classification

In cross-domain classification, the objective is to train a model on tasks sampled from one distribution, that then generalizes to tasks sampled from a different distribution. Specifically, we combine datasets so that the training split is drawn from one, and the validation and test split are taken from another. We experiment on mini-ImageNet→CUB (train split from mini-ImageNet and val/test split from CUB) and Omniglot→EMNIST. The Omniglot dataset (Lake et al., 2011) contains 1623 black and white characters taken from

50 different languages. Following standard practice, the number of classes is increased to 6492 by adding examples rotated by 90°, and we use 4114 for training. The EMNIST dataset (Cohen et al., 2017) contains single digits and characters from the English alphabet. We split the 62 classes into 31 for validation and 31 for test. We compare our method to the previously-considered approaches, using identical settings for number of epochs and model selection strategy (see Appendix B).

Results are given in Table 4. GPShot achieves the highest accuracy in most conditions. In Omniglot→EMNIST 1-shot, the best performance is achieved with a linear kernel (75.97%), and in 5-shot, with BNCosSim (90.30%) and polynomial first-order (90.72%, see Appendix E). In mini-ImageNet→CUB 1-shot, GPShot surpasses all the other methods obtaining the highest accuracy with CosSim (40.22%) and BNCosSim (40.14%); in the 5-shot case only Baseline++ is able to perform marginally better (57.31% vs 56.40%). Note that most competing methods experience difficulties in this setting, as shown by their low accuracies and large standard deviations. A comparison of kernels shows that first order ones are more effective (see Appendix E, Table 6).

6. Conclusion

In this work, we have demonstrated a highly flexible model based on GPs and deep kernel learning on a variety of domains. Compared with some other approaches in the literature for few-shot learning, our proposal performs better in regression and cross-domain classification while providing a measure of uncertainty. Future work could focus on exploiting the flexibility of the model in related settings (e.g. continual/online learning, reinforcement learning).

Acknowledgements

This work was supported by a Huawei DDMPLab Innovation Research Grant. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732204 (Bonseyes). This work is supported by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 16.0159. The opinions expressed and arguments employed herein do not necessarily reflect the official views of these funding bodies.

References

Antoniou, A. and Storkey, A. Learning to learn by self-critique. In *Advances in Neural Information Processing Systems*, 2019.

Antoniou, A., Edwards, H., and Storkey, A. How to train your MAML. In *International Conference on Learning Representations*, 2019.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, 1992.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C., and Huang, J.-B. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, 2018.

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. GPyTorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In *Advances in Neural Information Processing Systems*, 2018.

Gong, S., McKenna, S., and Collins, J. J. An investigation into face pose distributions. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, 1996.

Gordon, J., Bronskill, J., Bauer, M., Nowozin, S., and Turner, R. Meta-learning probabilistic inference for prediction. In *International Conference on Learning Representations*, 2019.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, 2018.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Hilliard, N., Phillips, L., Howland, S., Yankov, A., Corley, C. D., and Hodas, N. O. Few-shot learning with metric-agnostic conditional embeddings. *arXiv preprint arXiv:1802.04376*, 2018.

Hinton, G. and Salakhutdinov, R. Using deep belief nets to learn covariance kernels for gaussian processes. In *Advances in Neural Information Processing Systems*, 2008.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Jerfel, G., Grant, E., Griffiths, T., and Heller, K. A. Reconciling meta-learning and continual learning with online mixtures of tasks. In *Advances in Neural Information Processing Systems*, 2019.

Kuss, M. *Gaussian process models for robust regression, classification, and reinforcement learning*. PhD thesis, Technische Universität, 2006.

Lake, B., Salakhutdinov, R., Gross, J., and Tenenbaum, J. One shot learning of simple visual concepts. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2011.

LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521(7553):436, 2015.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Oreshkin, B., López, P. R., and Lacoste, A. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, 2018.

Pan, S. J. and Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22 (10):1345–1359, 2009.

Qiao, S., Liu, C., Shen, W., and Yuille, A. L. Few-shot image recognition by predicting parameters from activations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Rajeswaran, A., Finn, C., Kakade, S. M., and Levine, S. Meta-learning with implicit gradients. In *Advances in Neural Information Processing Systems*, 2019.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT press Cambridge, MA, 2006.

Ravi, S. and Larochelle, H. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

Rifkin, R. and Klautau, A. In defense of one-vs-all classification. *Journal of machine learning research*, 5:101–141, 2004.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, 2017.

Steyvers, M., Griffiths, T. L., and Dennis, S. Probabilistic inference in human semantic memory. *Trends in Cognitive Sciences*, 10(7):327–334, 2006.

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H., and Hospedales, T. M. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

Tossou, P., Dura, B., Laviolette, F., Marchand, M., and Lacoste, A. Adaptive deep kernel learning. *arXiv preprint arXiv:1905.12131*, 2019.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 2016.

Wah, C., Branson, S., Welinder, P., Perona, P., and Belongie, S. The Caltech-UCSD birds-200-2011 dataset, 2011.

Wang, Y., Chao, W.-L., Weinberger, K. Q., and van der Maaten, L. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *arXiv preprint arXiv:1911.04623*, 2019.

Wilson, A. and Adams, R. Gaussian process kernels for pattern discovery and extrapolation. In *International Conference on Machine Learning*, 2013.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial Intelligence and Statistics*, 2016.

Ye, H.-J., Hu, H., Zhan, D.-C., and Sha, F. Learning embedding adaptation for few-shot learning. *arXiv preprint arXiv:1812.03664*, 2018.

Yoon, J., Kim, T., Dia, O., Kim, S., Bengio, Y., and Ahn, S. Bayesian model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, 2018.

A. Kernels

Polynomial. This computes a covariance matrix based on the Polynomial kernel between inputs

$$k'(x, x') = (x^\top x' + c)^p, \tag{14}$$

where p is the degree of the polynomial and c is an offset parameter. We used $p = 1$ and $p = 2$ in our experiments.

Radial Basis Function kernel (RBF). The RBF is a stationary kernel given by the squared Euclidean distance between the two inputs

$$k'(x, x') = \exp\left(-\frac{||x - x'||^2}{2l^2}\right), \tag{15}$$

where l is a lengthscale parameters learned at training time.

Matérn kernel. This is a stationary kernel which is a generalization of the RBF and the absolute exponential kernel. It is parameterized by a value $\nu > 0$, commonly chosen as $\nu = 1.5$ (giving once-differentiable functions) or $\nu = 2.5$ (giving twice differentiable functions). The kernel is defined as follows:

$$k'(x, x') = |x - x'|^\nu K_\nu(|x - x'|). \tag{16}$$

We used a value of $\nu = 2.5$ in our experiments.

Spectral mixture kernel. The spectral mixture kernel was introduced by Wilson & Adams (2013) as a powerful stationary kernel for estimating periodic functions. The kernel models a spectral density with a Gaussian mixture

$$k'(\tau) = \sum_{q=1}^{Q} w_q \prod_{p=1}^{P} \exp\left\{-2\pi^2 \tau_p^2 v_q^{(p)}\right\} \cos\left(2\pi\tau_p \mu_q^{(p)}\right), \tag{17}$$

where $\tau = x - x'$, w_q are weights that specify the contribution of each mixture component, μ_q are the component periods, and v_q are lengthscales determining how quickly a component varies with the inputs x. We used 4 mixtures in our experiments.

Cosine similarity kernel (CosSim). The cosine similarity kernel consists in taking the product between the unit-normalized input vectors

$$k'(x, x') = \frac{xx'}{||x||\ ||x'||}. \tag{18}$$

The cosine similarity ranges from -1 (opposite) to 1 (same), with 0 indicating decorrelation (orthogonal). Following the suggestions in Wang et al. (2019) we experimented with another variant, meaning centering the input vectors through BatchNorm (BN) statistics (Ioffe & Szegedy, 2015) before the normalization (BNCosSim).

B. Training Details

Regression. In the function prediction experiment, we use the same backbone network described in Finn et al. (2017): a two-layer MLP, where each layer has 40 units and ReLU activations. We use the Adam optimizer with learning rate 10^{-3} over 5×10^5 training iterations. For the head pose estimation backbone, we use a three-layer convolutional neural network, each with 36 output channels, stride 2, and dilation 2 to downsample the 100×100 input images. We train for 100 steps using the Adam optimizer with learning rate 10^{-3}.

Classification. At training time we apply standard data augmentation (random crop, horizontal flip, and color jitter). The 1-shot training consists of 600 epochs, and 5-shot of 400, for MAML it corresponds to 60000 and 40000 episodes, and for Feature Transfer and Baseline++ to 400 and 600 supervised epochs with a mini-batch size of 16. In GPShot, the hyperparameters of the kernel are optimized with a learning rate one order of magnitude lower than that used for training the CNN. This helped with convergence. In all experiments we used first-order MAML for memory efficiency. This does not significantly affect results (see Chen et al., 2019). In all cases the validation set has been used to select the training epoch/episode with the best accuracy.

The Convolutional Neural Network (CNN) used for classification is given in Figure 3.

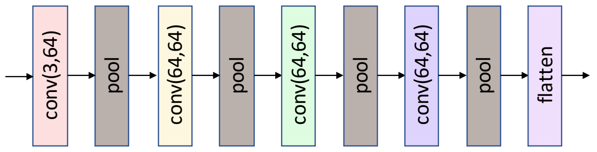

Figure 3. The CNN used as a backbone for classification. It consists of 4 convolutional layers, each consisting of a 2D convolution, a batch-norm layer, and a ReLU non-linearity. The first convolution changes the the number of channels of the input to 64, and the remaining convolutions retain this channel dimension. Each convolutional layer is followed by a max-pooling operation that decreases the spatial resolution of its input by a half. Finally, the output is flattened into a vector when is used as a feature.

C. Results regression experiments

Here, we provide additional samples of the few-shot regression experiments for a qualitative comparison (Figure 4).

Additionally we compare the latent spaces in the head trajectory estimation experiment. We reduced the number of hidden units to $\mathbf{h} = \{h_1, h_2\}$ and used a hyperbolic tangent activation function (tanh) to project the values to a Cartesian plane with $h_i \in [-1, 1]$. We then sampled 100 trajectories from the test set and recorded the value of $\mathbf{h}$ for the targets. The resulting plot is shown in Figure 5. The spectral ker-

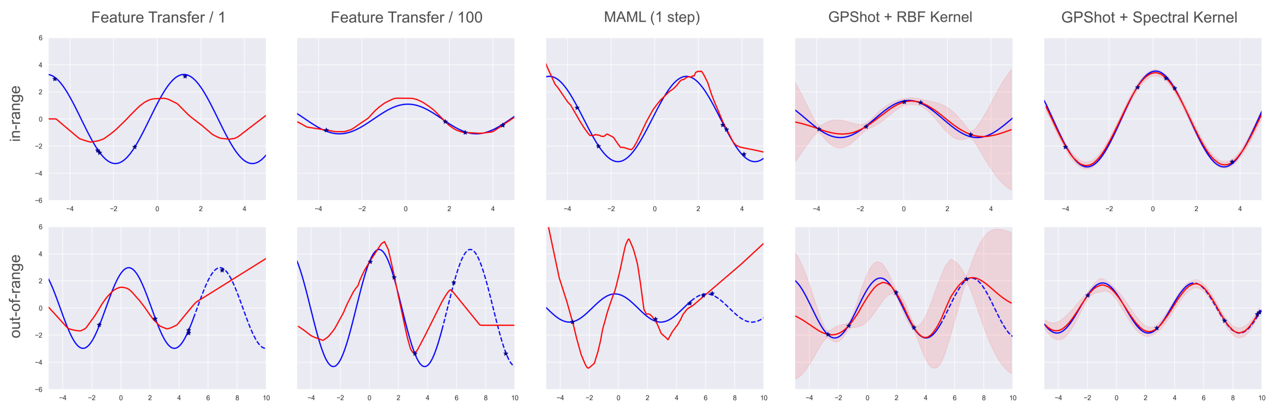

Figure 4. Additional samples for the unknown periodic function prediction experiment. We compare methods for in-range (top row) and out-of-range (bottom row) conditions. The true function is plotted in solid blue, the out-of-range portion in dotted blue, the approximation in red, and the uncertainty is given by a red shadow. The 5 support points (blue stars) are uniformly sampled in the available range.

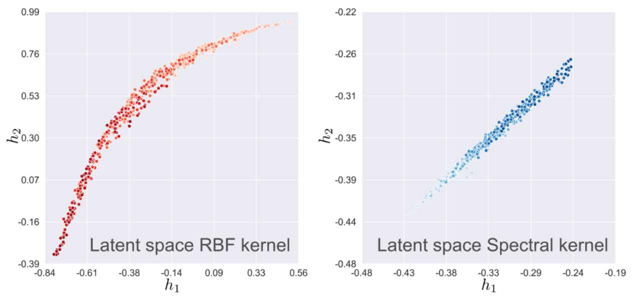

Figure 5. Latent space representation enforced by an RBF (left) and Spectral (right) kernel on the head trajectory experiments.

nel enforces a more compact manifold, clustering the head poses on a linear gradient based on the value of the target, leading to more accurate predictions.

D. Results classification experiments

We report additional results for the classification experiments, in particular a comparison between different kernels.

Kernel comparison. In Table 5 we show a comparison between different kernels (linear, RBF, Matérn, Polynomial $p = 1$ and $p = 2$, CosSim, BNCosSim) trained on CUB and mini-ImageNet. In this setting using a BNCosSim kernel gives a large advantage in almost all conditions. This result is in line with the findings of Wang et al. (2019), who showed how centering and unit normalizing the features considerably improve the performance in classification tasks. The overall performance of CosSim and BNCosSim is also in accordance with the findings of Chen et al. (2019) and their implementation of Baseline++, an effective feature transfer method based on the cosine distance. Further investigations are necessary in this direction to understand the reason why cosine metrics and normalization are so important in few-shot learning.

E. Results cross-domain experiments

Here, we report additional results for the cross-domain experiments.

Kernel comparison. In Table 6 we show a comparison between different kernels (linear, RBF, Matérn, Polynomial $p = 1$ and $p = 2$, CosSim, BNCosSim) trained on Omniglot→EMNIST and mini-ImageNet→CUB. Overall using a BNCosSim kernel still gives an advantage in almost all conditions, showing stable results. The best accuracy is achieved using more specialized kernels, however they often reach peak performance in specific conditions while underperforming in others.

Table 5. Average accuracy and standard deviation (percentage) on the few-shot classification setting (5-ways) for different kernels. Methods trained from scratch (three runs) with the same backbone (a four layer CNN), optimizer (Adam), and learning rate (10^{-3}). Test performed on novel classes with 3000 randomly generated tasks.

	CUB		**mini-ImageNet**	
Kernel	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Linear	60.23 ± 0.76	74.74 ± 0.22	48.44 ± 0.36	62.88 ± 0.46
RBF	55.34 ± 2.56	73.20 ± 1.41	45.92 ± 1.08	61.42 ± 0.74
Matérn	58.20 ± 0.63	73.21 ± 1.30	47.65 ± 0.85	62.59 ± 0.12
Polynomial ($p = 1$)	59.54 ± 1.10	74.51 ± 0.98	47.78 ± 0.60	62.54 ± 0.96
Polynomial ($p = 2$)	5718 ± 0.40	71.14 ± 0.58	46.36 ± 0.34	60.26 ± 0.40
CosSim	$\mathbf{63.37 \pm 0.19}$	77.73 ± 0.26	48.64 ± 0.45	62.85 ± 0.37
BNCosSim	62.96 ± 0.62	$\mathbf{77.76 \pm 0.62}$	$\mathbf{49.73 \pm 0.07}$	$\mathbf{64.00 \pm 0.09}$

Table 6. Average accuracy and standard deviation (percentage) over three runs on the cross-domain setting (5-ways) for different kernels. We use the same setup as in the classification setting.

	Omniglot→EMNIST		**mini-ImageNet→CUB**	
Kernel	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Linear	$\mathbf{75.97 \pm 0.70}$	89.51 ± 0.44	38.72 ± 0.42	54.20 ± 0.37
RBF	74.46 ± 0.41	88.38 ± 0.53	36.22 ± 0.40	51.30 ± 0.52
Matérn	75.46 ± 0.20	88.04 ± 1.81	36.98 ± 0.41	51.35 ± 0.16
Polynomial ($p = 1$)	74.33 ± 0.67	$\mathbf{90.72 \pm 0.47}$	38.24 ± 0.30	54.11 ± 0.40
Polynomial ($p = 2$)	75.58 ± 1.18	88.06 ± 0.70	36.83 ± 0.46	51.92 ± 0.87
CosSim	73.06 ± 2.36	88.10 ± 0.78	$\mathbf{40.22 \pm 0.54}$	55.65 ± 0.05
BNCosSim	75.40 ± 1.10	90.30 ± 0.49	40.14 ± 0.18	$\mathbf{56.40 \pm 1.34}$