Deep Kernel Transfer in Gaussian Processes for Few-shot Learning

Massimiliano Patacchiola
University of Edinburgh

Jack Turner
University of Edinburgh

Elliot J. Crowley
University of Edinburgh

Amos Storkey
University of Edinburgh

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 introduce a Bayesian method based on Gaussian Processes (GPs) that can learn efficiently from a limited amount of data and generalize across new tasks and domains. We frame few-shot learning as a model selection problem by learning a deep kernel across tasks, and then using this kernel as a covariance function in a GP prior for Bayesian inference. This probabilistic treatment allows for cross-domain flexibility, and uncertainty quantification. We provide substantial experimental evidence, showing that the proposed method is better than several state-of-the-art algorithms in few-shot regression and cross-domain classification.

1 INTRODUCTION

One of the key differences between state-of-the-art machine learning methods, such as deep learning (LeCun et al., 2015; Schmidhuber, 2015), and human learning is that the former need 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). Following this line of research, we argue that a probabilistic treatment of few-shot learning is an indispensable prerequisite, and propose the use of Gaussian Processes (GPs, Rasmussen and 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 and 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. Both the hyperparameters of the GP and the weights of the neural network can be efficiently learned in parallel to maximize the marginal likelihood. We show that a GP trained this way is efficient in the few-shot regime and provides several advantages compared with standard methods, such as the ability to quantify uncertainty, and demonstrate flexibility in cross-domain adaptation. A comparison across methods shows that GPs obtain state-of-the-art results in few-shot regression and cross-domain classification, while being competitive in within-domain classification.

Our contributions are as follows:

1. We provide a principled way to deal with the few-shot learning problem in the context of GPs, showcasing their strength across domains.

2. We introduce a robust method for dealing with few-shot regression and cross-domain classification, two challenging scenarios that have been scarcely considered in the literature.

3. We conduct thorough empirical analysis to show the effectiveness of our methodology, and open source our implementation (https://github.com/BayesWatch/deep-kernel-transfer).
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 \( \mathcal{S} = \{(x_1, y_1)\}_{n=1}^N \) 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 \( \mathcal{D} = \{\mathcal{T}_n\}_{n=1}^N \) defined as a collection of such tasks. Models are trained on random tasks sampled from \( \mathcal{D} \). Then, given a new task \( \mathcal{T}_n = \{\mathcal{S}_n, \mathcal{Q}_n\} \) sampled from a test set, the objective is to condition the model on the samples of the support \( \mathcal{S}_n \) to estimate the membership of the samples in the query set \( \mathcal{Q}_n \).

In the most common scenario, the inputs \( x \in \mathcal{D} \) belong to the same distribution \( p(x) \) and are distributed across training, validation, and test sets such that their class membership is non-overlapping. Note that \( y \) can be a continuous value (regression) or a discrete one (classification), even though most of the previous work has focused on classification. We also consider the cross-domain scenario, where the inputs are sampled from different distributions at training and test time; this is more representative of real-world scenarios.

2.2 Gaussian Processes

A GP is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams, 2006). GPs have been mainly used to tackle regression problems, however a treatment for classification is also possible (see Section 3.3). Given the inputs \( x \) and \( x' \) a GP is fully specified by a mean function \( m(x) \) and a covariance function \( k(x, x') \), that define a distribution over functions

\[
f(x) \sim \mathcal{GP}(m(x), k(x, x')),
\]

where the function \( m(x) \) and the kernel \( k(x, x') \) are defined as

\[
m(x) = \mathbb{E}[f(x)], \tag{2a}
\]

\[
k(x, x') = \text{cov}(f(x), f(x')). \tag{2b}
\]

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 the property that for a pair of input points the corresponding outputs will be more correlated than dissimilar pairs.

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 has been generated by a process \( f_n(x_n) \) corrupted by homoscedastic Gaussian noise \( \epsilon_n \) with variance \( \sigma^2 \);

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

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 \( \mathbf{f} \) conditioned on the values of \( \mathbf{y} \) is given by the isotropic Gaussian

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

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 a Gaussian with mean \( m(\mathbf{x}) = 0 \) and covariance matrix \( \mathbf{K} \) defined by the kernel \( k(x, x') \):

\[
p(\mathbf{y}) = \mathcal{N}(|\mathbf{y}|0, \mathbf{K}), \quad \text{with} \quad K_{ij} = k(x_i, x_j). \tag{5}
\]

Notice that the kernel function must define a positive semi-definite matrix, therefore inducing a proper covariance matrix. The simplest kernel has a linear expression

\[
k_{\text{LIN}}(x, x') = vx^\top x', \tag{6}
\]

where \( v \) 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; and the spectral mixture kernel (Wilson and 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. Additional details about the kernels used in this work are reported in Appendix A.

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

$$E[f_x] = k_*^\top (K + \sigma^2 I)^{-1} y.$$  \hspace{1cm} (7a)

$$\text{cov}(f_x) = k_* - k_*^\top (K + \sigma^2 I)^{-1} k_*.$$  \hspace{1cm} (7b)

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

**Marginal likelihood (evidence).** We would now like to learn \(\theta\). The fully Bayesian predictive distribution is given by marginalizing the evidence over the hyperparameters

$$p(y_*|x_*, D) = \int p(y_*|x_*, \theta)p(\theta|D)d\theta.$$  \hspace{1cm} (8)

This integral is intractable and a complete Bayesian treatment is only possible through MCMC sampling, however in most cases this is computationally expensive. An alternative solution is to perform evidence approximation by assuming that the posterior over \(\theta\) is sharply-peaked, which gives the maximum likelihood type II (ML-II) estimate of the hyperparameters

$$\hat{\theta} = \arg\max_{\theta} p(D|\theta).$$  \hspace{1cm} (9)

ML-II assigns a point estimate over \(\theta\), meaning that Equation (8) is approximated by

$$p(y_*|x_*, D) \approx p(y_*|x_*, D, \hat{\theta}),$$  \hspace{1cm} (10)

and therefore the posterior becomes tractable again.

**Deep kernel learning.** In deep kernel learning (Hinton and Salakhutdinov, 2008; Wilson et al., 2016) the input \(x\) is mapped to a latent vector \(z\) through a non-linear function \(\mathcal{F}_\phi(x) \rightarrow z\) (e.g. a neural network) parameterized by a set of weights \(\phi\). The embedding is defined such that the dimensionality of the input is significantly reduced, meaning that if \(x \in \mathbb{R}^J\) and \(z \in \mathbb{R}^K\) then \(J \gg K\). Once the input has been encoded in \(z\) 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, starting from the kernel \(k(x, x'|\theta)\) the inputs are passed through the non-linear function

$$k(x, x'|\theta) \rightarrow k(\mathcal{F}_\phi(x), \mathcal{F}_\phi(x')|\theta, \phi).$$  \hspace{1cm} (11)

The hyperparameters \(\theta\) and the parameters of the model \(\phi\) are jointly learned by maximizing the log marginal likelihood as previously described in Equations (8), (9), and (10). This is achieved by updating the weights of the CNN by backpropagating the error.

**Computational cost.** One of the burdens of GPs is the \(O(N^3)\) computational cost for \(N\) training points arising from the inversion of the covariance matrix at inference time. There have been different proposals to reduce this cost e.g. inducing points (Hensman et al., 2015) or structure-exploiting algebra (Wilson and Nickisch, 2015). In few-shot learning the severity of the problem is significantly reduced as, by definition, the training set has a limited size, therefore the use of advanced techniques is superfluous. This is another point in favor of GPs in this particular setting.

### 3 THE METHOD

#### 3.1 Bayesian model selection

Contrary to the canonical approach to few-shot learning, we favor a Bayesian approach in terms of model selection (MacKay, 1992). It is well known that choosing a model based on a maximum likelihood estimation results in overparameterized models that generalize poorly and violate Occam’s razor. The Bayesian view of model comparison involves the use of probabilities to represent uncertainty in the choice of model.

Let us define the dataset \(D = \{(x_n, y_n)\}_{n=1}^N\) as a set of input and output pairs, and \(M\) as a set of models that can be used to fit the data. Let \(p(M_i)\) and \(p(M_j)\) denote the prior beliefs on models \(M_i\) and \(M_j\). It is possible to perform a direct comparison between the two candidates by estimating the posterior odds of \(p(M_i|D)\) over \(p(M_j|D)\), which in the case of uniform priors becomes the Bayes factor (Good, 1958). Most of the time, handling a large number of models is computationally prohibitive, so a common approach is to select a single model which appears to be most plausible given the observed data. Assuming the prior \(p(M)\) to be uniform over all models this corresponds to selecting the model with the highest evidence \(p(D|M)\).

With GPs it is possible to follow similar reasoning. Finding a model in this case means finding the parameter of a (deep) kernel, with the best candidate being the kernel that guarantees the highest evidence. Once the data \(y\) becomes available, the marginal likelihood measures the expectedness of the data under the given set of parameters. The evidence can be expressed analytically by fixing hyperparameters \(\theta\) and weights \(\phi\) to their point estimates \(\hat{\theta}\) and \(\hat{\phi}\), and taking the loga-
The data-fit is in a negative quadratic form and it is the only term which depends on the training outputs $y$. The complexity penalty term embeds Occam’s razor. Note that the tradeoff between penalty and data-fit is automatic, meaning that there is no weighting parameter which needs to be set to balance the two terms. The model parameters can be estimated via ML-II by taking the derivative of the expression and maximizing it via gradient ascent.

### 3.2 Few-shot learning as model selection

For few-shot learning, the Bayesian selection principle is applied across tasks over the dataset $D = \{T_n\}_{n=1}^N$. More precisely, we assume that the same hyperparameters $\theta$ and weights $\phi$ are shared across each datapoint in the support $S = \{(x_l, y_l)\}_{l=1}^L$ and query $Q = \{(x_m, y_m)\}_{m=1}^M$ sets belonging to each task. Therefore, once a common prior has been found, knowledge can be transferred. At training time, a task $T = \{S, Q\}$ is sampled from $D$, then the log marginal likelihood of Equation (12) is estimated over $S \cup Q$ (assuming $y \in Q$ to be observed) and the parameters of the GP are updated through ML-II to maximize the evidence. 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 $T_* = \{S_*, Q_*\}$ the prediction on the query set $Q_*$ is made via conditioning on the support set $S_*$, using the parameters that have been learned at training time. The graphical model representing GP few-shot learning is reported in Figure 1, and the pseudocode is given in Algorithm 1.

### 3.3 Few-shot classification

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 $O(1)$ to linear $O(K)$, 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 and Klautau, 2004; Rasmussen and Williams, 2006). Experimentally, LR and LSC tend to be more effective than other approaches in both binary (Kuss, 2006) and multi-class (Rifkin and 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 $\theta$ and $\phi$.

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. Assuming independence, the log marginal likelihood of Equation (12) is replaced by the
Algorithm 1 Few-shot GP train and test procedures

Require: \( D = \{T_n\}_{n=1}^N \) train dataset
Require: \( T_c = \{S_c, Q_c\} \) test task
Require: \( \theta, \phi \): GP hyperparameters, Net weights
Require: \( \alpha, \beta \): step size hyperparameters

1: procedure Train(\( D, \alpha, \beta, \theta, \phi \))
2: \hspace{1cm} while not done do
3: \hspace{1.2cm} Sample task \( T = \{S, Q\} \sim D \)
4: \hspace{1.2cm} Assign \( x \leftarrow x \in S \cup Q, y \leftarrow y \in S \cup Q \)
5: \hspace{1.2cm} Estimate loss \( L = -\log p(y|x, \theta, \phi) \) \( \triangleright \) Eq. (12)
6: \hspace{1.2cm} Update GP \( \theta \leftarrow \theta - \alpha \nabla_\theta L_{S \cup Q} \)
7: \hspace{1.2cm} Update Net \( \phi \leftarrow \phi - \beta \nabla_\phi L_{S \cup Q} \)
8: \hspace{1cm} end while
9: end procedure

10: procedure Test(\( T_c \))
11: Assign \( x \leftarrow x \in S_c, y \leftarrow y \in S_c \)
12: Assign \( x_s \leftarrow x \in Q_c \)
13: Estimate \( E(f_c) \) and \( \text{cov}(f_c) \) \( \triangleright \) Eq. (7)
14: end procedure

\[
\begin{align*}
\log p(y|x, \hat{\theta}, \hat{\phi}) &= \sum_{c=1}^{C} \log p(y_c|x, \hat{\theta}, \hat{\phi}). \\
\end{align*}
\] (13)

Given a new input \( x_s \) and the \( C \) outputs of all the binary classifiers, a decision is made by selecting the output with the highest probability

\[
c_s = \arg\max_c (\sigma(m_c(x_s))),
\] (14)

where the predictive mean \( m(x) \) has been previously defined in Equations (2) and (7), \( \sigma(\cdot) \) is the sigmoid function, and \( c_s \in \{1, \ldots, C\} \).

\section{4 RELATED WORK}

The problem of few-shot learning has been tackled from several perspectives, which we now summarize.

\textbf{Feature transfer}. There exists a wealth of literature on feature transfer (Pan and 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 several limitations; part of the model has to be trained from scratch for each new task, and this method has a propensity to overfit. As an extension to this, Chen et al. (2019) propose the use of cosine distance between examples (which they dub Baseline++). However, this still relies on the assumption that a fixed fine-tuning protocol will balance the bias-variance tradeoff correctly for every task.

\textbf{Metric learning}. Another approach is to 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 a Long-Short Term Memory (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 representing each class. Each prototype is the mean vector of the embedded support points belonging to its class. The Euclidean distance is used to estimate the similarity between a new point and each prototype, to assign it to the appropriate 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 to the support set, to identify matching categories.

\textbf{Meta-learning}. Meta-learning models attempt to optimise the process of learning for new tasks. One approach consists of training a meta-learner that learns how to update the parameters of an underlying model (Bengio et al., 1992; Schmidhuber, 1992). Model-Agnostic Meta-Learning (MAML, Finn et al., 2017) has been proposed as a way to meta-learn the parameters of a model over many tasks by backpropagating through a limited number of update steps over multiple support sets, such that the initial parameters provide a good starting point to learn specific parameters on a new task. The method is model-agnostic in the sense that it can be applied to any model trained through gradient descent, including classification, regression, and reinforcement learning models.

\textbf{Multi-task learning}. Multi-task learning is complementary to few-shot learning. In both cases, the aim is to avoid learning a new model for each series of task, however, in multi-task learning there is a limited number of interconnected tasks and a relatively large amount of data for each task. In this context there have been attempts to define an inter-task GP prior. For instance, GPs have been adapted to the multi-task case by defining an index kernel able to represent inter-task covariance (Bonilla et al., 2008). Similarly, a version of informative vector machines has been used to estimate the underlying parameters of a GP (Lawrence and Platt, 2004).

\section{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
Deep Kernel Transfer in Gaussian Processes for Few-shot Learning

Figure 2: A qualitative comparison between different methods on the prediction of unknown periodic functions. We report both 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, and the approximation in red. Uncertainty is given by a red shadow. The 5 support points (blue stars) are uniformly sampled from the available range. The proposed method (GPNet) provides the best fit to the true curve, while providing a measure of uncertainty.

Table 1: Average Mean-Squared Error (MSE) and standard deviation over three runs for few-shot regression of unknown periodic functions, and head pose trajectory estimation (QMUL), with 10 samples for train, and 5 samples for test. We distinguish between test points taken from the same domain at the training points (in-range) and those from an extended unseen domain (out-of-range). The proposed method (GPNet) has the lowest error in all conditions (highlighted in bold).

| Method                        | Unknown Functions | Head trajectories (QMUL) |
|-------------------------------|-------------------|--------------------------|
|                               | in-range          | out-of-range             | in-range     | out-of-range |
| Feature Transfer/1            | 2.94 ± 0.16       | 6.13 ± 0.76              | 0.25 ± 0.04  | 0.20 ± 0.01  |
| Feature Transfer/100          | 2.67 ± 0.15       | 6.94 ± 0.97              | 0.22 ± 0.03  | 0.18 ± 0.01  |
| MAML (Finn et al., 2017)      | 2.76 ± 0.06       | 8.45 ± 0.25              | 0.21 ± 0.01  | 0.18 ± 0.02  |
| GPNet + RBF [ours]            | 1.38 ± 0.03       | 2.61 ± 0.16              | 0.12 ± 0.04  | 0.14 ± 0.03  |
| GPNet + Spectral [ours]       | 0.08 ± 0.06       | 0.10 ± 0.06              | 0.10 ± 0.01  | 0.11 ± 0.02  |

this issue, releasing an open-source benchmark to allow for a uniform comparison between methods. We have integrated our algorithm into this framework using PyTorch and GPyTorch (Gardner et al., 2018); our code is available at https://github.com/BayesWatch/deep-kernel-transfer. 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 GPNet. Training details are reported in Appendix B.

5.1 Regression

We perform a series of 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 ∈ [0.1, 5.0], phase ∈ [0, π], and range ∈ [-5.0, 5.0], and Gaussian noise (µ = 0, σ = 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 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 use the Queen Mary University of London multiview face dataset (QMUL, Gong et al., 1996), which consists of grayscale face images of 37 people (32 train, 5 test). For each person there are 133 facial images covering a viewsphere of ±90° in yaw and ±30° in tilt at 10° incre-
Figure 3: Uncertainty estimation for an outlier when predicting head trajectory. The images of the trajectory are given in the top row with the outlier highlighted by a red frame—95% of the outlier image has been cut out. The trajectory in the left column is random, and on the right, the trajectory has a constant pitch. In both cases the proposed method (GPNet) is able to estimate a mean value (red line) close to the true value (blue circle) while showing larger uncertainty. Feature transfer performs poorly at the same location.

Few methods have tackled few-shot regression, so 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, described in Section 4, 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. Results for the regression experiments are summarized in Table 1, and prediction plots are given in Figure 2. The proposed method (GPNet) 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 ($1.38 \text{ vs } 0.08$, and $2.61 \text{ vs } 0.10$ 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-range (8.45). Figure 2 shows that both feature transfer and MAML are unable to fit the true function, especially out-of-range. We observe similar results for head pose estimation, with GPNet reporting lower MSE over all conditions—this is also reported in Table 1. Qualitative results on the uncertainty quantification experiment are shown in Figure 3. For the corrupted image (highlighted with a red frame), GPNet predicts a value very close to the true one (blue circle), while also indicating a high level of uncertainty (the red shadow). Feature transfer performs poorly, predicting an unrealistic pose.

To understand the reason why the spectral kernel obtains a lower MSE than its RBF counterpart, we analyzed the latent space generated by the two in the head trajectory estimation experiment. We reduced the number of hidden units to $z = \{z_1, z_2\}$ and used an hyperbolic tangent activation to project the values to a Cartesian plane with $z \in [-1, 1]$. Then, we sampled 100 trajectories from the test set and recorded the value of $z$ for the various targets. These are plotted in Figure 4: we can see that the spectral kernel enforce a more compact manifold, clustering the head poses on a linear gradient based on the value of the target.

Figure 4: Latent space representation enforced by an RBF (left) and Spectral (right) kernel on the head trajectory experiments. Pitch $-30^\circ \rightarrow +30^\circ$ are denoted by lighter→darker dots.
Table 2: Average accuracy and standard deviation (percentage) over three runs on the few-shot classification setting (5-ways). All the methods have been trained with the same backbone (a four layer CNN), optimizer (Adam), and learning rate ($10^{-3}$). The test has been performed on novel classes with 3000 randomly generated tasks. The proposed method (GPNet) is competitive across various datasets and conditions. The best results are highlighted in bold for ease of comparison.

| Method                  | CUB       | mini-ImageNet |
|-------------------------|-----------|--------------|
|                         | 1-shot    | 5-shot       | 1-shot    | 5-shot   |
| 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 | **78.51 ± 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 (Sun et al., 2018) | **62.52 ± 0.34** | 78.22 ± 0.07 | **48.76 ± 0.17** | 64.20 ± 0.28 |
| GPNet + Linear [ours]   | **60.23 ± 0.76** | **74.74 ± 0.22** | **48.44 ± 0.36** | **62.88 ± 0.46** |
| GPNet + RBF [ours]      | 55.34 ± 2.56 | 73.20 ± 1.41 | 45.92 ± 1.08 | 61.42 ± 0.74 |
| GPNet + Matérn [ours]   | 58.20 ± 0.63 | 73.21 ± 1.30 | 47.65 ± 0.85 | 62.59 ± 0.12 |
| GPNet + Polynomial ($p = 1$) [ours] | 59.54 ± 1.10 | 74.51 ± 0.98 | 47.78 ± 0.60 | 62.54 ± 0.96 |

5.2 Classification

We perform few-shot classification on two challenging datasets: the Caltech-UCSD Birds (CUB-200, Wah et al., 2011), and mini-ImageNet (Russakovsky et al., 2015; Ravi and Larochelle, 2017). The CUB dataset has been widely used for fine-grained classification, and it consists of 11788 images across 200 classes. We follow standard protocol by dividing 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 a random selection of 64 classes for train, 16 for validations and 20 for test, as is common practice (Ravi and Larochelle, 2017; Chen et al., 2019). All the experiments are 5-way (5 randomly selected 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 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 (Sun et al., 2018). We further compare against feature transfer, and Baseline++ from Chen et al. (2019).

Results. The results are reported in Table 2 (average accuracy as a percentage). In the CUB dataset GPNet is competitive with state-of-the-art algorithms in the 1-shot setting, but less effective in the 5-shot. In mini-ImageNet 1-shot, GPNet with a linear kernel has higher accuracy than any other approach (48.44%), excluding RelationNet which is marginally better (48.76%). The performance of RelationNet can be explained by the expensive relation module used in this architecture. In mini-ImageNet 5-shot the average accuracy of GPNet is in line with other state-of-the-art methods, and similar in value to MatchingNets. Across different kernels, those with first-order covariance functions (e.g. linear and polynomial with $p = 1$) are the best overall. This is most likely due to a low-curvature manifold induced by the neural network in the latent space, which increases the linear separability of data.

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 of 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. The results are given in Table 3.
The authors would like to thank Joseph Mellor for his helpful feedback. This work was supported by a Huawei DDMPLab Innovation Research Grant.

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 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 for other applications. For instance, uncertainty quantification could play a crucial role in few-shot reinforcement learning.

Acknowledgements

The authors would like to thank Joseph Mellor for his helpful feedback. This work was supported by a Huawei DDMPLab Innovation Research Grant.

Table 3: 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 (GPNet) has the best score on most conditions. The best results are highlighted in bold for ease of comparison.

| Method              | Omniglot→EMNIST 1-shot | Omniglot→EMNIST 5-shot | mini-ImageNet→CUB 1-shot | mini-ImageNet→CUB 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             | 57.31 ± 0.11             |
| MatchingNet (Vinyals et al., 2016) | 75.01 ± 2.00           | 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             |
| GPNet + Linear [ours] | 75.97 ± 0.70           | 89.51 ± 0.44           | 38.72 ± 0.42             | 54.20 ± 0.37             |
| GPNet + RBF [ours]   | 74.46 ± 0.41           | 88.38 ± 0.53           | 36.22 ± 0.40             | 51.30 ± 0.52             |
| GPNet + Matérn [ours] | 75.46 ± 0.20           | 88.04 ± 1.81           | 36.98 ± 0.41             | 51.35 ± 0.16             |
| GPNet + Polynomial (p = 1) [ours] | 74.33 ± 0.67           | 90.72 ± 0.47           | 38.24 ± 0.30             | 54.11 ± 0.40             |

Overall, GPNet has the highest accuracy. In Omniglot→EMNIST 1-shot, the best performance is achieved with linear and Matérn kernels (75.97% and 75.46%), and in 5-shot, with linear and polynomial first-order (89.51% and 90.72%). In mini-ImageNet→CUB, GPNet surpasses most methods; only Baseline++ is able to perform better by exploiting the fine-tuning stage on the unseen classes with a large dataset. However, Baseline++ performs very poorly in Omniglot→EMNIST due to heavy overfitting. Note that most competing methods experience difficulties in this setting, as shown by the large standard deviation across runs. MAML seems to be particularly ineffective; this may be due to an exacerbation of the instability of the algorithm in cross-domain classification, as observed recently by Antoniou et al. (2019).

References

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

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

Bonilla, E. V., Chai, K. M., and Williams, C. K. I. (2008). Multi-task Gaussian process prediction. In Advances in Neural Information Processing Systems.

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

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

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

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

Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. (2018). GPYtorch: Black-box matrix-matrix gaussian process inference with gpu acceleration. In Advances in Neural Information Processing Systems.

Gong, S., McKenna, S., and Collins, J. J. (1996). An investigation into face pose distributions. In Proceedings of the International Conference on Automatic Face and Gesture Recognition.
Deep Kernel Transfer in Gaussian Processes for Few-shot Learning

Good, I. J. (1958). Significance tests in parallel and in series. *Journal of the American Statistical Association*, 53(284):799–813.

Hensman, J., Matthews, A., and Ghahramani, Z. (2015). Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*.

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

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

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

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

Lawrence, N. D. and Platt, J. C. (2004). Learning to learn with the informative vector machine. In *International Conference on Machine Learning*.

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

MacKay, D. J. (1992). Bayesian interpolation. *Neural Computation*, 4(3):415–447.

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

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

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

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141.

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. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

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

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

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

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

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

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

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

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

Wilson, A. and Adams, R. (2013). Gaussian process kernels for pattern discovery and extrapolation. In *International Conference on Machine Learning*, pages 1067–1075.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured gaussian processes (KISS-GP). In *International Conference on Machine Learning*.

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

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

\[ k(x, x') = (x^\top x' + c)^p, \]

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

\[ k(x, x') = \exp \left( -\frac{||x - x'||^2}{2l^2} \right), \]

where \( l \) is a lengthscale parameter 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:

\[ k(x, x') = |x - x'|^\nu K_\nu(|x - x'|). \]

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

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

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

where \( \tau = x - x' \), \( w_q \) are weights that specify the contribution of each mixture component, \( \mu_q \) 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.

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 \( 5 \times 10^5 \) 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 \( 100 \times 100 \) 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 GPNet, 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 5.

---

**Figure 5:** 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 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 used as a feature.