GP-Tree: A Gaussian Process Classifier for Few-Shot Incremental Learning

Idan Achituve [1] **Aviv Navon** [1] **Yochai Yemini** [1] **Gal Chechik** [1 2] **Ethan Fetaya** [1]

Abstract

Gaussian processes (GPs) are non-parametric, flexible, models that work well in many tasks. Combining GPs with deep learning methods via deep kernel learning (DKL) is especially compelling due to the strong representational power induced by the network. However, inference in GPs, whether with or without DKL, can be computationally challenging on large datasets. Here, we propose *GP-Tree*, a novel method for multi-class classification with Gaussian processes and DKL. We develop a tree-based hierarchical model in which each internal node of the tree fits a GP to the data using the Pòlya-Gamma augmentation scheme. As a result, our method scales well with both the number of classes and data size. We demonstrate the effectiveness of our method against other Gaussian process training baselines, and we show how our *general* GP approach achieves improved accuracy on standard incremental few-shot learning benchmarks.

1. Introduction

Gaussian processes (GPs) are a popular Bayesian non-parametric approach that enjoys a closed-form marginal likelihood, thus avoiding one of the major computational difficulties of many Bayesian approaches. However, due to several obstacles, successfully applying GPs to certain problems may be challenging. First, the performance of GP models heavily depends on the kernel function being used. In domains such as images, where common kernels are not a good measure of semantic similarity, this can hinder the performance. This problem is commonly addressed by deep kernel learning (DKL) where GPs are combined with modern neural networks to learn a useful kernel function from the data (Calandra et al., 2016; Wilson et al., 2016a).

Second, computing the marginal likelihood involves storing and inverting an $n \times n$ kernel matrix, where n is the number of training examples. This can limit exact inference on large datasets, especially when combined with DKL where the kernel needs to be re-computed at each iteration. A common approach to handle large datasets is to learn a small set of inducing points, which act as a proxy for the training data (Silverman, 1985; Quinonero-Candela & Rasmussen, 2005; Snelson & Ghahramani, 2006).

While extensive work has been done on handling these challenges for Gaussian process regression, comparatively little has been done on how to scale Gaussian process classification (see Liu et al., 2020a, for a recent review). This is partly because the categorical distribution on the target variable results in non-Gaussian posteriors and we no longer have a closed-form marginal likelihood. One appealing approach to address this obstacle is the Pólya-Gamma augmentation (Polson et al., 2013). In the Pólya-Gamma augmentation, the posterior becomes Gaussian when conditioned on the augmented Pólya-Gamma variable. However, this augmentation was designed for binary classification tasks. Since then, several extensions to multi-class classification have been proposed (Linderman et al., 2015; Galy-Fajou et al., 2020; Snell & Zemel, 2021). However, as we will show empirically their performance degrades as the number of target classes increases.

In this study, we present a novel method for Gaussian process classification (GPC) that is designed to handle both small and large datasets. Importantly, it is also designed to handle a large number of classes. We develop a tree-based model in which each node solves a binary classification task using a Gaussian process and the Pólya-Gamma augmentation scheme. We term our method *GP-Tree*. GP-Tree has great flexibility as it allows us to take samples from the posterior with Gibbs sampling, or apply variational inference and use the inducing points approximation. To train GP-Tree on large-scale image classification tasks, we further combine it with DKL and we show how in this setup as well it is superior to popular GPC methods.

Finally, we apply GP-Tree to incremental few-shot learning challenges (Tao et al., 2020). In incremental few-shot learning, we assume that the data come sequentially. Initially, the learner is presented with many samples from some base classes. Then, at each new iteration, it has access only to a new, small, dataset that originated from a set of novel classes

arXiv:2102.07868v4 [cs.LG] 13 Jul 2021

not seen during previous iterations. Here the challenges are two-fold, generalizing from a small number of training points for the novel classes and avoiding catastrophic forgetting of the previously encountered classes. We claim that GPs are a natural fit for this problem. The inducing points, which summarize the training data, can help mitigate the catastrophic forgetting problem, and GPs generalize well from small datasets due to their Bayesian nature. Indeed, we show that once we get GPs to scale and successfully train on the base classes, our GP approach achieves performance gains over baseline methods on incremental few-shot benchmarks.

Thus, we make the following novel contributions: (1) we show how current Gaussian process classification methods struggle when the number of classes to be learned is large; (2) we present a novel method for Gaussian process classification that is designed to handle a large number of classes and large datasets based on the Pólya-Gamma augmentation; (3) we present GPs as a promising new research direction for few-shot incremental learning; (4) we demonstrate competitive classification accuracy on two benchmark datasets designed for few-shot incremental learning. Our code is publicly available at https://github.com/IdanAchituve/GP-Tree.

2. Background

2.1. Notations

We denote vectors with bold lower-case font, e.g. $\boldsymbol{x}$, and matrices with capital bold font, e.g. $\boldsymbol{X}$. Given a dataset $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_n, y_n)$, we denote by $\boldsymbol{y} = [y_1, ..., y_n]^T$ the vector of labels, and by $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ the design matrix whose i^{th} row is $\boldsymbol{x}_i$. In the classification case, each y_i takes a value from $\{1, ..., C\}$ class labels.

2.2. Gaussian Processes

In Gaussian process learning we assume the mapping from the input points to the target values is via a latent function f. The target values are assumed to be independent when conditioned on the latent function, i.e., $p(\boldsymbol{y}|\boldsymbol{X}, f) = \prod_{i=1}^{n} p(y_i|f(\boldsymbol{x}_i))$. The latent function is assumed to follow a Gaussian process prior $f \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$, where the evaluation vector of f on $\boldsymbol{X}$, $\boldsymbol{f} = [f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_n)]^T$, has a Gaussian distribution $\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{K})$, where $\boldsymbol{\mu}_i = m(\boldsymbol{x}_i)$ and $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The mean $m(\boldsymbol{x})$ is commonly taken to be the constant zero function, and the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ is a positive semi-definite function.

Let $\boldsymbol{X}, \boldsymbol{y}$ be the training data, and let f_* be the evaluation of f on a novel point $\boldsymbol{x}_*$. In the regression case, we assume $p(y|\boldsymbol{x}, f) = \mathcal{N}(f(\boldsymbol{x}), \sigma^2)$. Therefore, the predictive distributions, $p(f_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X})$ and $p(y_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X}) = \int p(f_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X}) p(y_*|f_*) df_*$, are Gaussians with known

parameters. Specifically,

$$
\begin{aligned}
p(f_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X}) &= \mathcal{N}(\mu_*, \sigma_*), \\
\mu_* &= \boldsymbol{k}_*^T (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}, \\
\sigma_* &= k_{**} - \boldsymbol{k}_*^T (\boldsymbol{K} + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{k}_*.
\end{aligned}
\tag{1}
$$

Where, $k_{**} = k(x_*, x_*)$, and $\boldsymbol{k}_*[i] = k(\boldsymbol{x}_i, \boldsymbol{x}_*)$. This closed-form solution allows us to avoid the costly marginalization step; however, it entails the inversion of an $n \times n$ matrix which can be expensive to compute for large datasets.

DKL (Wilson et al., 2016a) is a popular choice to apply a kernel on structured data such as images. The kernel over the input data points is commonly in the form of a fixed kernel on an embedding learned by a deep neural network g_θ, e.g., $k_\theta(\boldsymbol{x}, \boldsymbol{x}') = \exp(-||g_\theta(\boldsymbol{x}) - g_\theta(\boldsymbol{x}')||^2 / 2\ell^2)$. Therefore, the closed-form inference is of even greater importance as it allows to easily backpropagate through the GP inference.

2.3. The Pólya-Gamma Augmentation

When applying GPs to classification tasks, the likelihood $p(y|f(\boldsymbol{x}_i))$ is no longer Gaussian. The predictive distributions are also no longer Gaussian and we do not have a closed-form solution for them. To overcome this limitation, several methods were offered based on the Pólya-Gamma augmentation (Polson et al., 2013) to model the discrete likelihoods in GPs (Linderman et al., 2015; Wenzel et al., 2019; Galy-Fajou et al., 2020; Snell & Zemel, 2021). The Pólya-Gamma augmentation hinges on the following identity

$$
\frac{(e^\psi)^a}{(1 + e^\psi)^b} = 2^{-b} e^{\kappa \psi} \mathbb{E}_\omega[e^{-\omega \psi^2 / 2}],
\tag{2}
$$

where $\kappa = a - b/2$, and ω has the Pólya-Gamma distribution $\omega \sim PG(b, 0)$.

Suppose we have a binary classification task with $y \in \{0, 1\}$, and we are given a vector of latent function values $\boldsymbol{f}$, the likelihood can be written as,

$$
p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{j=1}^{n} \sigma(f_j)^{y_j} (1 - \sigma(f_j))^{1 - y_j} = \prod_{j=1}^{n} \frac{e^{y_j f_j}}{1 + e^{f_j}},
\tag{3}
$$

where $\sigma(\cdot)$ is the sigmoid function. We can now use Eq. 2 to introduce auxiliary Pólya-Gamma variables (one per sample) such that we recover Eq. 3 by marginalizing them out. If instead we sample the Pólya-Gamma variables $\boldsymbol{\omega}$ we get the following posteriors:

$$
\begin{aligned}
p(\boldsymbol{f}|\boldsymbol{y}, \boldsymbol{\omega}) &= \mathcal{N}(\boldsymbol{f}|\boldsymbol{\Sigma}(\boldsymbol{K}^{-1}\boldsymbol{\mu} + \boldsymbol{\kappa}), \boldsymbol{\Sigma}), \\
p(\boldsymbol{\omega}|\boldsymbol{y}, \boldsymbol{f}) &= PG(\boldsymbol{1}, \boldsymbol{f}).
\end{aligned}
\tag{4}
$$

Where $\kappa_j = y_j - 1/2$, $\boldsymbol{\Sigma} = (\boldsymbol{K}^{-1} + \boldsymbol{\Omega})^{-1}$, and $\boldsymbol{\Omega} = diag(\boldsymbol{\omega})$. We can now sample from $p(\boldsymbol{f}, \boldsymbol{\omega}|\boldsymbol{y}, \boldsymbol{X})$ using block Gibbs sampling and get Monte-Carlo estimations of the marginal and predictive distributions.

2.4. Inducing Points

Exact inference with GPs compels us to store the entire training set and invert an $n \times n$ matrix $\boldsymbol{K}_{nn}$, which greatly limits their use. A common solution to this problem is to use inducing points (Silverman, 1985; Quinonero-Candela & Rasmussen, 2005; Snelson & Ghahramani, 2006). With inducing points, we usually define $m \ll n$ pseudo-inputs whose locations are often trainable parameters. Then we only need to invert an $m \times m$ matrix, thus allowing us to control the computational cost.

Our approach closely follows the inducing points method for binary classification with the Pólya-Gamma augmentation presented in (Wenzel et al., 2019). We define $\bar{\boldsymbol{X}}$ as the inducing locations/inputs and $\bar{\boldsymbol{f}}$ as the latent function value evaluated on $\bar{\boldsymbol{X}}$. We have,

$$p(\bar{\boldsymbol{f}}) = \mathcal{N}(0, \ \boldsymbol{K}_{mm}), \tag{5}$$

$$p(\boldsymbol{f}|\bar{\boldsymbol{f}}) = \mathcal{N}(\boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\bar{\boldsymbol{f}}, \ \boldsymbol{Q}_{nn}),$$
$$\boldsymbol{Q}_{nn} = \boldsymbol{K}_{nn} - \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}. \tag{6}$$

$$p(\boldsymbol{y}, \boldsymbol{\omega}, \boldsymbol{f}, \bar{\boldsymbol{f}}) = p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{\omega})p(\boldsymbol{f}|\bar{\boldsymbol{f}})p(\bar{\boldsymbol{f}})p(\boldsymbol{\omega}), \tag{7}$$

where $\boldsymbol{K}_{mm}$ is the kernel matrix on the inducing locations, and $\boldsymbol{K}_{nm}$ is the cross-kernel matrix between the inputs and the inducing locations. While $p(\bar{\boldsymbol{f}}|\boldsymbol{y}, \boldsymbol{\omega})$ is Gaussian due to the Pólya-Gamma augmentation, the distribution after marginalizing over $\boldsymbol{\omega}$, $p(\bar{\boldsymbol{f}}|\boldsymbol{y})$, is not Gaussian. Following (Hensman et al., 2015), Wenzel et al. (2019) proposed a variational inference approach based on the following assumption $q(\boldsymbol{\omega}, \bar{\boldsymbol{f}}) = q(\boldsymbol{\omega})q(\bar{\boldsymbol{f}})$, where $q(\boldsymbol{\omega})$ is a Pólya-Gamma density and $q(\bar{\boldsymbol{f}})$ is a Gaussian density. Then, the inducing locations and the variational parameters are learned with the evidence lower bound.

3. Method

In Sections 3.1-3.3 we describe our method to train GP classifiers with DKL that scale to large training sets and a large number of classes. Afterward, in Section 3.4, we will show how our model, with minor modifications, can be adjusted to the few-shot class-incremental learning setup.

3.1. Hierarchical Classification

Consider the case presented in (Linderman et al., 2015) aimed at modeling categorical and multinomial data. To derive a Pólya-Gamma augmentation Linderman et al. (2015) utilized the stick-breaking representation for the multinomial density. They turn the multi-class classification task into a sequence of $C - 1$ binary classification tasks where $y^j = 1$ if the original label is j and $y^j = 0$ if the original label is larger than j. Then, $C - 1$ independent Gaussian

processes can be learned, one for each of the binary classification tasks. Denoting by $\boldsymbol{f}_i = (f_i^1, ..., f_i^{C-1})$ the vector of latent processes values at the ith example, the stick-break likelihood is:

$$p(y_i = c|\boldsymbol{f}_i) = \sigma(f^c)\prod_{k<c}(1 - \sigma(f^k)), \tag{8}$$

where the remaining probability mass is assigned to the Cth class. While the stick-breaking formulation allows to break the multi-class classification problem into a sequence of binary classification tasks, as we will show in Section 5.1, its performance degrades with the number of classes. Intuitively, the stick-breaking process can be viewed as a hierarchical classification with an extremely unbalanced tree (see illustration in Figure 1(a)). This sequential structure can be severely sub-optimal for two reasons: (1) the number of binary classification tasks needed to classify a data point grows linearly with the number of classes instead of logarithmic for a perfectly balanced tree; (2) not all label splits result in equally hard binary classification tasks, yet the stick-breaking process uses the default label ordering.

Therefore, we propose to use a tree-structured hierarchical classification instead of the sequential alternative. This construction allows finding a tree structure that results in easy-to-learn binary tasks. Conceptually, we create a tree by splitting the data recursively by classes until we get to single class leaves. More formally, starting at the root node, we partition the classes $\{1, ..., C\}$ into two disjoint sets C_l and C_r, such that $C_l \cup C_r = \{1, ..., C\}$. Let D_l and D_r denote the data points associated with the classes in C_l and the classes in C_r respectively. We assign D_l to the left child of the root node and D_r to the right one. We recursively apply the same operation at each node until we are left with single class leaf nodes. Thus, a binary tree is formed (not necessarily a perfectly balanced one), see Figure 1(b) for an example. We then fit a binary GP classifier to each internal node of the tree that makes a binary decision to either go left or go right at that node.

The model quality depends on how we partition the data, namely the tree construction processes described above. A naive approach would be to use a random balanced binary tree; however, this strategy does not take advantage of the semantic meaning of the classes. We propose the following procedure, we first compute a representative prototype for each class by taking the mean of all samples (or their representation obtained by a NN) belonging to the same class coordinate-wise. We then normalize the vectors to have unit length and apply divisive hierarchical clustering. We recursively split each node by using k-means++ clustering (Arthur & Vassilvitskii, 2007) with $k = 2$ on the class prototype vectors, until we are left with single class leaves. We note that partitioning the data in this manner may be sub-optimal when working on the input space directly; however,
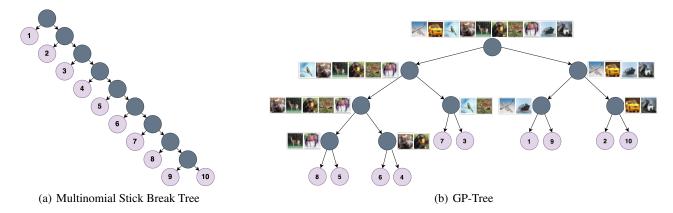
(a) Multinomial Stick Break Tree

(b) GP-Tree

Figure 1. The trees corresponding to the multinomial stick break model (left) and the GP-Tree model (right) for CIFAR-10. The multinomial stick break generates an unbalanced tree in which the order of the classes is arbitrary. GP-Tree, on the other hand, generates a more balanced tree that is divided by the semantic meaning of the classes. For example, motorized vehicles are on the right subtree of the root node while animals are on the left one. This semantic partition is pronounced at all tree levels.

when applied on features extracted by a NN it is very sensible since NNs tend to generate points that cluster around a single prototype for each class (Snell et al., 2017). Then, we fit a GP to each internal node which makes a binary decision based on the data associated with that node. We denote the GP associated with node v_i by $f_{v_i} \sim \mathcal{GP}(m_{v_i}, k_{v_i})$, and all of the GPs in the tree with $\mathcal{F}$. The likelihood of a data point having the class c is given by the unique path P^c from the root to the leaf node corresponding to that class:

$$p(y = c|\mathcal{F}) = \prod_{v_i \in P^c} \sigma(f_{v_i})^{y_{v_i}} (1 - \sigma(f_{v_i}))^{1-y_{v_i}}, \quad (9)$$

where $y_{v_i} = 1$ if the path goes left at v_i and zero otherwise.

3.2. Inference at the Node Level

Since the likelihood in Eq. 9 factorizes over the nodes, we may look at the individual components separately. In the following we omit the subscript v_i for clarity; however, all datum and quantities are those that belong to a specific node v_i. In general, we can perform inference on each tree node with either Gibbs sampling or variational inference (VI). For training on large datasets with deep kernel learning and inducing points, we found the variational approach to scale better, as the inducing points posterior depends on the entire dataset. However, when modeling the novel classes in incremental few-shot learning, the Gibbs sampling procedure is more suitable, as no new parameters are required.

For Gibbs sampling, we use the posterior probabilities introduced in section 2.3. At each node, we can use block-Gibbs sampling to sample $\boldsymbol{\omega}$ and $\boldsymbol{f}$. Then we can obtain the augmented marginal and predictive distributions described next.

The augmented marginal likelihood:

$$p(\boldsymbol{y}|\boldsymbol{\omega}, \boldsymbol{X}) = \int p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{\omega}, \boldsymbol{X})p(\boldsymbol{f})d\boldsymbol{f} \\ \propto \mathcal{N}(\boldsymbol{\Omega}^{-1}\boldsymbol{\kappa}|\boldsymbol{0}, \ \boldsymbol{K} + \boldsymbol{\Omega}^{-1}), \quad (10)$$

and the augmented predictive likelihood on a new data point $\boldsymbol{x}_*$:

$$p(y_*|\boldsymbol{x}_*, \boldsymbol{\omega}, \boldsymbol{y}, \boldsymbol{X}) = \int p(y_*|f_*)p(f_*|\boldsymbol{x}_*, \boldsymbol{\omega}, \boldsymbol{y}, \boldsymbol{X})df_*, \quad (11)$$

where,

$$p(f_*|\boldsymbol{x}_*, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{\omega}) = \mathcal{N}(f_*|\mu_*, \ \Sigma_*), \\ \mu_* = \boldsymbol{k}_*^T(\boldsymbol{\Omega}^{-1} + \boldsymbol{K})^{-1}\boldsymbol{\Omega}^{-1}\boldsymbol{\kappa}, \quad (12) \\ \Sigma_* = k_{**} - \boldsymbol{k}_*^T(\boldsymbol{\Omega}^{-1} + \boldsymbol{K})^{-1}\boldsymbol{k}_*.$$

Where we assumed a zero mean prior. The integral in Eq. 11 is intractable but can be computed numerically with 1D Gaussian-Hermite quadrature.

Alternatively to the Gibbs sampling, we may apply variational inference at each node. We define $\bar{\boldsymbol{X}}$ as the learned pseudo-inputs and $\bar{\boldsymbol{y}}$ as their associated class labels. $\bar{\boldsymbol{X}}$ are defined at the tree level and are shared by all relevant nodes. For each node we define the variational distributions $q(\boldsymbol{\omega}) = PG(\boldsymbol{1}, \boldsymbol{c})$ and $q(\bar{\boldsymbol{f}}) = \mathcal{N}(\bar{\boldsymbol{f}}|\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}})$, where $\boldsymbol{c}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$ are learnable parameters. The variational lower bound to the log marginal likelihood is:

$$\mathcal{C}(\boldsymbol{c}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) = \mathbb{E}_{p(\boldsymbol{f}|\bar{\boldsymbol{f}})q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega})}[log \ p(\boldsymbol{y}|\boldsymbol{\omega}, \boldsymbol{f})] - \\ KL(q(\bar{\boldsymbol{f}}, \boldsymbol{\omega})||p(\bar{\boldsymbol{f}}, \boldsymbol{\omega})), \quad (13)$$

which has a closed-form expression. We can then learn the model parameters and the variational parameters by

maximizing the variational lower bound in Eq. 13. The explicit form of Eq. 13 and the update rules of the variational parameters were adapted from (Wenzel et al., 2019) and are presented in supplementary Section A.

To make predictions we can plug in the approximate posterior in the predictive posterior calculation:

$$
\begin{aligned}
p(f_*|\boldsymbol{x}_*, \bar{\boldsymbol{X}}, \bar{\boldsymbol{y}}) &\approx \int p(f_*|\bar{\boldsymbol{f}}, \boldsymbol{x}_*) q(\bar{\boldsymbol{f}}) d\bar{\boldsymbol{f}} \\
&= \mathcal{N}(f_*|\mu_*, \Sigma_*), \\
\mu_* &= \boldsymbol{k}_{m*}^T \boldsymbol{K}_{mm}^{-1} \tilde{\boldsymbol{\mu}}, \\
\Sigma_* &= k_{**} - \boldsymbol{k}_{m*}^T (\tilde{\boldsymbol{\Sigma}} \boldsymbol{K}_{mm}^{-1} - \boldsymbol{I}) \boldsymbol{k}_{m*},
\end{aligned} \tag{14}
$$

where $\boldsymbol{k}_{m*}$ denotes the kernel vector between the inducing points and the test point, and k_{**} denotes the kernel value at the test point. Similarly to the Gibbs sampling case, we can obtain $p(y_*|x_*, \bar{\boldsymbol{y}}, \bar{\boldsymbol{X}})$ by taking an integral over f_* and compute it numerically with 1D Gaussian-Hermite quadrature.

3.3. Full Learning of the Tree

We discussed how to learn and perform inference on a single node, we will now describe how to train the full tree model. For the full tree we need to learn the joint pseudo-inputs and the per-node variational parameters. Optimizing the full tree splits into the separate marginal likelihood of all examples and all the nodes on the path from the root to the leaf nodes:

$$
\begin{aligned}
\mathcal{L} &= \sum_{j=1}^{n} \log \ p(y_j|\boldsymbol{x}_j, \bar{\boldsymbol{y}}, \bar{\boldsymbol{X}}) \\
&= \sum_{j=1}^{n} \sum_{v_i \in P^{y_j}} \log \ p(y_{v_i}|\boldsymbol{x}_j, \bar{\boldsymbol{y}}, \bar{\boldsymbol{X}}).
\end{aligned} \tag{15}
$$

Since we cannot directly optimize this loss, we optimize the lower bound of it given in Eq. 13. We summarize the learning algorithm of GP-Tree with VI in supplementary Section B. To apply predictions in the original multi-class problem, the prediction for a new data point x_* is given as the product of predictive distributions from the root node to leaf node corresponding to each class.

Our method can be easily integrated with DKL. We simply superimpose the tree-based GP on an embedding layer of a neural network and learn the network parameters θ as well. In this case, we may define the inducing inputs in the input space or the feature space. We found that setting them in the feature space yields better performance, and requires less memory. Their location was initialized at the beginning of training using k-means++ applied on the embedding of data samples for each class separately. Also, we empirically found it beneficial to start the training process with a few epochs of standard training using the cross-entropy loss
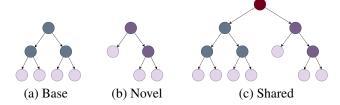

Figure 2. Tree expansion for novel classes: (a) A tree that was learned on the base classes; (b) On a novel session, we first build a tree from the novel classes representations; (c) Next we connect the trees using a shared root node.

before building the GP tree and transitioning to learning with it.

3.4. GP-Tree for Few-Shot Class-Incremental Learning

In class-incremental learning, we are given a sequence of labeled datasets $D_1, D_2, ..., D_T$, each sampled from a disjoint set of classes $C_1, ..., C_T$. At each timestamp t, the model has access to dataset D_t and the previous model, and it is tasked with classifying all the previously seen classes $\cup_{i=1}^{t} C_i$. In few-shot incremental learning (Ren et al., 2019; Tao et al., 2020), the base classes dataset, D_1, has a large number of samples, but all of the following datasets (D_2 onwards) have a limited amount of labeled data, e.g., 5 samples per class. Thus, after the model learned the previous classes it is tasked with learning new classes from few examples and without impairing the classification of previously seen classes whose data isn't available at that time, also known as catastrophic forgetting (McCloskey & Cohen, 1989). Gaussian processes are naturally well-suited for this challenge. Bayesian models generalize well from few examples (Snell & Zemel, 2021) and the inducing points can be used as a compact representation of the base classes, allowing us to avoid catastrophic forgetting. As Gaussian processes are non-parametric learners, we can classify new classes without fitting any new variables or tuning the NN parameters, which may also mitigate the catastrophic forgetting problem.

At the initial phase of learning the base classes, we employ GP-Tree (with DKL) to learn this dataset using VI as described earlier. We can view the inducing inputs, which are learned per class, as 'exemplars' of the base classes, a commonly used practice in incremental learning studies (He et al., 2018). We then use them in future learning sessions as described next. At the end of this stage, we freeze the NN backbone to avoid any parameter learning.

At each new session, we are given data from novel classes. We use the (fixed) NN backbone to obtain representations for the new samples that will be used for modeling the novel classes with GP-Tree. In this case, unlike the learning of base classes, the datasets are small. Therefore, we do not

need to use inducing points. We just need to restructure the tree to account for the new classes and fit a GP to each new node. There are several alternatives for restructuring the tree, here we propose the following. We retain the original tree that was learned on the base classes intact, and at each novel session, we build a sub-tree from the embedding of samples in all (novel) sessions until the current one, namely $D_2, ..., D_t$. We refer to these examples as novel examples hereafter. We then connect the sub-tree of novel examples with the sub-tree of the base classes with a shared root node. We define a GP classifier for each node in the sub-tree of novel examples and the root node. The GPs for the nodes in the sub-tree of the base classes are left untouched. Fitting the GPs in the sub-tree of novel examples is easily done with the representation of all examples at hand. For the root node, since we no longer have the base classes examples, we may use the inducing inputs of the base classes along with the embeddings of the novel examples. See illustration in Figure 2. We use Gibbs sampling on all new nodes to avoid any parameter tuning after the initial training on the base classes. Supplementary Section D.2 presents other alternatives for constructing the tree at novel sessions.

It is important to note that while we do save the inducing inputs and the network representations of samples from novel sessions for future sessions, this is in line with other incremental learning studies that save a few samples per class (Douillard et al., 2020). Furthermore, we only save the embeddings and not the original images. Therefore, the memory costs are low and in practice are negligible compared to storing the trained network weights. Also, we emphasize that the method described here is a natural way to extend our classification model to new classes. It was not tailored for the class-incremental few-shot scenario specifically. Despite that, we show strong results compared to models designed specifically for this task, especially on later learning sessions (see Section 5.3). Finally, GP-Tree can be easily extended to standard incremental learning setups. One immediate option is to build a tree and learn inducing inputs only for the novel classes seen at each new session similarly to the learning applied for base classes with VI. Then this sub-tree can be connected to the previous tree with a shared root node. This strategy can be further improved by taking only the inducing inputs from all classes seen thus far at the end of each novel session to rebuild the entire tree. Then we can use either the VI approach or, preferably, our Gibbs sampling procedure. In this study, we focus on the few-shot case and we leave this extension to future work.

4. Related Work

Incremental learning. Incremental learning (IL) aims at learning new data without forgetting old data, what is known as 'catastrophic forgetting' (McCloskey & Cohen, 1989). Recently, a large body of research was done in that direction, most of which is based on methods that use NNs only. Notable studies that use a similar procedure to ours are Titsias et al. (2019) which regularize subsequent tasks with a set of inducing points stored from previous tasks, and Gidaris & Komodakis (2018) which also freeze the feature extractor after learning the base classes and infer a classification weight vector for novel categories. Methods in this field can be categorized according to three types: (i) *Regularization based* approaches impose regularization methods on the network to maintain past knowledge (Goodfellow et al., 2014; Kirkpatrick et al., 2017; Lee et al., 2017; Chaudhry et al., 2018; Schwarz et al., 2018; Ren et al., 2019). For example, Kirkpatrick et al. (2017) limit the update of the parameters when encountering new data based on the fisher matrix ; (ii) *Architectural based* methods suggest network architectures that are resilient to the catastrophic forgetting issue and can accommodate new tasks (Mallya et al., 2018; Mallya & Lazebnik, 2018; Yoon et al., 2018; Serra et al., 2018; Taitelbaum et al., 2019; jun Liu et al., 2020). For example, Rusu et al. (2016) and following it Yoon et al. (2018) expend the network with each new task. When applying parameters update the former freezes the previous network while the latter retrain part of it; (iii) *Rehearsal based* aims at preventing catastrophic forgetting by storing and replaying information from previous episodes (Rebuffi et al., 2017; Castro et al., 2018; Wu et al., 2018; Hou et al., 2019; Zhai et al., 2019; Liu et al., 2020b; jun Liu et al., 2020). Rebuffi et al. (2017) introduced the class-incremental learning setup. They used exemplars to maintain information of past data and applied the nearest-mean-of-exemplars classification rule. In this paper, we follow the protocol suggested by (Tao et al., 2020) for few-shot class-incremental learning.

Gaussian process classification. In GPs for classification tasks the likelihood is no longer Gaussian and therefore approximation-based approaches or Monte-Carlo-based approaches are needed. Some classic non-augmentation based methods include the Laplace approximation (Williams & Barber, 1998), expectation-propagation (Minka, 2001), and the least square approach (Rifkin & Klautau, 2004). We refer the readers to (Rasmussen & Williams, 2006) for a more thorough review. Recently some approaches emerged that are based on the Pólya-Gamma augmentation (Polson et al., 2013). Linderman et al. (2015) proposed to use the Pólya-Gamma in a stick-breaking process to reparameterize a multinomial distribution as a product of binomial distributions. Wenzel et al. (2019) proposed to use Pólya-Gamma augmentation with variational inference for binary classification tasks. Galy-Fajou et al. (2020) proposed to use the logistic softmax likelihood and derived a conditionally conjugate model based on three augmentation steps. We believe that the quality of the prediction and learning may degrade

Table 1. Test accuracy on CUB-200-2011. Average over 10 runs ($\pm$ SEM). In bold: statistically significant best results (p=0.05).

Method	Number of Classes								
	4	6	8	10	20	30	40	50	60
SBM-GP	96.98±0.5	95.55±0.9	92.11±0.8	91.04±0.7	83.59±0.5	77.63±0.9	67.03±0.8	64.20±1.0	60.69±1.0
OVE	97.33±0.5	96.60±0.9	94.70±0.8	93.05±0.8	–	–	–	–	–
LSM	97.30±0.9	96.95±0.8	95.16±0.8	93.09±0.9	89.23±1.3	84.44±0.6	72.90±1.2	69.82±0.9	65.53±0.9
GP-Tree Rnd. (ours)	97.80±0.8	95.96±0.7	93.49±0.9	92.07±1.1	85.32±1.6	77.45±1.0	68.97±0.8	62.77±1.2	58.49±0.8
GP-Tree (ours)	97.93±0.7	97.15±0.6	94.67±0.9	93.57±0.9	88.77±1.4	83.87±0.8	**75.66±0.8**	**72.87±0.8**	**69.92±0.6**

as a result of the cascade of approximations. Snell & Zemel (2021) proposed to use the one-vs-each likelihood in a few shot setting. Their method does not scale well with the data and classes due to the inversion of a $CN \times CN$ matrix. We use a Gibbs sampling approach for learning novel classes at each node. Several studies suggested alternative, more efficient, posterior sampling techniques when the classes are imbalanced or when using sub-samples of the data (Nemeth & Fearnhead, 2020; Sachs et al., 2020; Sen et al., 2020). Using such techniques to improve the standard Gibbs sampler is out of the current study scope.

Scalable GPs. In recent years some attempts were made to make GP classification more scalable. Inducing points (Silverman, 1985; Quinonero-Candela & Rasmussen, 2005; Snelson & Ghahramani, 2006) are a popular method to handle large datasets. Hoffman et al. (2013) developed a stochastic optimization process for VI. Hensman et al. (2015) introduced a method for GPC within a variational inducing point framework. Izmailov et al. (2018) used tensor train decomposition for variational parameters that allowed increasing dramatically the number of inducing points. Wilson et al. (2016b) proposed to learn multiple GPs on an embedding space and combine them linearly before a Softmax function. Extending this method to the incremental learning setting is not immediate as there are learnable parameters for combining the classes. Bradshaw et al. (2017) used a GP with the Robust-Max likelihood for robustness against adversarial examples. This method doesn't scale well with the number of classes as we will show in Section 5.2.

Hierarchical models. Hierarchical classifiers are a popular design choice. Morin & Bengio (2005), and following it Mnih & Hinton (2008) proposed a tree-based classifier for language modeling. Their method applies to situations where all words are known in advance, while we need the ability to dynamically adapt the tree with new classes. Damianou & Lawrence (2013) stacked multiple GPs to create a hierarchy of GPs. Nguyen et al. (2019) proposed a hierarchical model using a mixture of GPs to learn global and local information. However, it is not clear how to extend this method to incremental learning challenges.

Hierarchical stick break process was used in several con-

texts as well. Adams et al. (2010) proposed a tree-based stick break for clustering as an alternative to the standard sequential stick break. Nassar et al. (2019) proposed to use a tree-structure stick break for linear dynamical systems. Both methods did not include any GP components and do not have the flexibility of our model to apply inference with either VI or Gibbs sampling.

5. Experiments

In our experiments, we first examine several aspects of our method compared to previous common GPC methods (Sections 5.1 & 5.2). Then we evaluate GP-Tree on the setting of class incremental few-shot learning (Section 5.3). In the supplementary material we provide full implementation details (Section C), ablation study, and further analysis (Section D).

5.1. Inference with Gibbs Sampling

As described in Section 3, GP-Tree allows to do inference with Gibbs sampling. This method is preferable when modeling novel classes in incremental learning, as we do not want to optimize any parameters and the size of the datasets are small. We evaluated GP-Tree in this setup on the fine-grained classification dataset, Caltech-UCSD Birds (CUB) 200-2011 (Welinder et al., 2010). The CUB dataset contains 11,788 images of bird species from 200 classes with approximately 30 examples per class in the training set. Here, we did not apply DKL, but rather we used the pre-trained features published by (Xian et al., 2018). This allowed us to only compare the inference part of our model.

We compared GP-Tree against the following baselines that also used the Pólya-Gamma augmentation to get a conditionally conjugate likelihood: **(1) Stick Break Multinomial GP (SBM-GP)** (Linderman et al., 2015): that used the stick-breaking process to convert a multinomial likelihood to a product of binomial likelihoods; **(2) Logistic-Softmax (LSM)** (Galy-Fajou et al., 2020) a recent method for GPC based on the logistic-softmax likelihood; and **(3) One-vs-Each (OVE)** (Snell & Zemel, 2021) a method for GPC proposed recently for few-shot learning. Because this method requires the inversion of a $CN \times CN$ matrix, we were able

to run it with only a few classes.

Table 1 compares GP-Tree against the baseline methods at increasing number of classes starting from 4 to 60 (out of 200). The results are the average test-set classification accuracy along with the standard error of the mean (SEM) over ten seeds which included randomization in the class selection. When the number of classes is small (≤ 30) GP-Tree, LSM, and OVE are comparable. However, as the number of classes increases GP-Tree performs better. Table 1 also shows a variant of GP-Tree in which a balanced tree is built based on a random split of the classes (**GP-Tree Rnd**). This variant performs similarly to the SBM-GP baseline, indicating the importance of the class split algorithm in GP-Tree. To gain a better understanding of that we depict in Figure 1(b) the tree generated by GP-Tree on the CIFAR-10 dataset compared to a (possible) tree that corresponds to the SBM-GP model. The figure shows that: (i) GP-Tree generates a more balanced tree; and (ii) GP-Tree generates a tree that is ordered by the semantic meaning. For example, motorized vehicles are on the right subtree of the root node while animals are on the left subtree.

Finally, empirically we noticed that the number of steps in the Gibbs sampling procedure has a minor effect on the model performance. Therefore, we believe that it indicates that the chains converge quickly. Supplementary Section D.1 further shows improved results for GP-Tree with more parallel Gibbs chains. It also compares the VI approach with the Gibbs sampling procedure, showing a large performance gap in favor of the latter. This result strengthens our choice for using the Gibbs procedure during novel sessions when using GP-Tree for incremental learning.

5.2. GPC with DKL

For evaluating GP-Tree with DKL we used the CIFAR-10 and CIFAR-100 datasets (Krizhevsky et al., 2009). We compared GP-Tree with the following popular baselines that applied GPC with DKL: **(1) Stochastic Variational Deep Kernel Learning (SV-DKL)** (Wilson et al., 2016b) which learned multiple GPs, each on a different subset of the embedding space, and combined them with the Softmax function; and **(2) GPDNN** (Bradshaw et al., 2017) which used the Robust-Max likelihood (Hernández-Lobato et al., 2011). We used ResNet-18 (He et al., 2016) as the backbone NN with an embedding layer of size 1024 and trained the models for 200 epochs.

Table 2 shows the average accuracy across 3 seeds for both datasets. From the table, both GP-Tree and SV-DKL achieve high accuracy; however, GP-Tree prevails on both datasets. We found that GPDNN was extremely sensitive to the learning rate choice and the hyper-parameter controlling the probability of labeling error and we could not get reasonable results for it on the CIFAR-100 dataset.

Table 2. Test accuracy ($\pm$ SEM) of GPs with DKL

Method	CIFAR-10	CIFAR-100
GPDNN	81.16 ± 0.1	–
SV-DKL	$92.73 \pm .05$	70.61 ± 0.2
GP-Tree (ours)	$\mathbf{93.25 \pm 0.1}$	$\mathbf{72.11 \pm .05}$

We note that the SV-DKL method is less suited for incremental learning, as the model includes a linear mapping followed by a softmax to produce a distribution over the classes. Therefore, it needs to learn new parameters with each new session and risks catastrophic forgetting, unlike our approach where no new parameters are tuned.

5.3. Few-Shot Class-Incremental Learning

In this section, we evaluate GP-Tree on the challenging task of few-shot class-incremental learning (FSCIL). We compare with methods that were designed for this learning setup and show comparable, if not superior, results by simply applying GP-Tree. This indicates that Gaussian processes in general and GT-Tree, in particular, are well suited and a natural approach to incremental few-shot learning.

We follow the benchmarks proposed in (Tao et al., 2020), using the CUB 200-2011 dataset and mini-Imagenet, a 100-class subset of the Imagenet (Deng et al., 2009) dataset used in few-shot studies (Vinyals et al., 2016; Finn et al., 2017). We adopt the 10-way 5-shot setting for CUB, choosing 100 base classes, and splitting the remaining 100 classes into ten incremental sessions. For mini-ImageNet, we follow the 5-way 5-shot, with 60 base classes, for a total of nine sessions.

Since the data splits made public by (Tao et al., 2020) did not include a validation set, we pre-allocate a small portion of the base classes dataset for hyper-parameter tuning of GP-Tree, SDC (Yu et al., 2020), and PODNet (Douillard et al., 2020) on both datasets.

We compare GP-Tree with recent and leading FSCIL methods. The results of the following methods were taken from (Tao et al., 2020): **(1) iCaRL** (Rebuffi et al., 2017) that used exemplars of past data and applied the nearest-mean-of-exemplars classification; **(2) EEIL** (Castro et al., 2018) that used a distillation loss for old classes combined with a classification loss on all classes; **(3) NCM** (Hou et al., 2019) which combined a classification loss, a distillation loss over normalized embedding layer, and a margin ranking loss; **(4) TOPIC** (Tao et al., 2020) that optimized a neural gas network with a classification loss, an anchor loss for less forgetting stabilization and a min-max loss to reduce overfitting. We also compare with two additional baselines: **(5) SDC** (Yu et al., 2020) that combined several losses to learn

Table 3. Few-shot class-incremental learning results on CUB-200-2011. Test accuracy averaged over 10 runs.

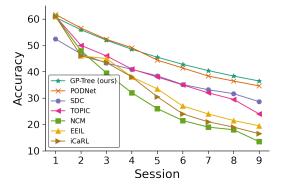| Method	Sessions										
	1	2	3	4	5	6	7	8	9	10	11
iCaRL	68.68	52.65	48.61	44.16	36.62	29.52	27.83	26.26	24.01	23.89	21.16
EEIL	68.68	53.63	47.91	44.20	36.30	27.46	25.93	24.70	23.95	24.13	22.11
NCM	68.68	57.12	44.21	28.78	26.71	25.66	24.62	21.52	20.12	20.06	19.87
TOPIC	68.68	62.49	54.81	49.99	45.25	41.40	38.35	35.36	32.22	28.31	26.28
SDC	64.10	60.58	57.00	53.57	52.09	49.87	48.20	46.38	44.04	43.81	42.39
PODNet	**75.93**	**70.29**	**64.50**	49.00	45.90	43.00	41.33	40.56	40.09	40.59	39.30
GP-Tree (ours)	73.73	68.24	64.22	**59.61**	**56.39**	**53.40**	**51.14**	**49.32**	**47.03**	**45.86**	**44.48**

Figure 3. Few-shot class-incremental learning results on mini-ImageNet. Test set accuracy averaged over 10 runs.

embedding representation and introduced a drift compensation to update previously computed prototypes; and **(6) PODNet** (Douillard et al., 2020) that used a spatial-based distillation-loss and a representation consisted of multiple proxy vectors per class.

The results for CUB are presented in Table 3 and for mini-ImageNet in Figure 3. On both datasets, we found the PyTorch implementation of ResNet-18 to be consistent with the results seen in (Tao et al., 2020). We note that the results on mini-ImageNet could be improved by adapting the NN architecture for smaller images, but we kept the standard ResNet-18 for comparability with (Tao et al., 2020).

The comparison shows that while PODNet outperforms all other methods during the first sessions, our GP-Tree achieves the best accuracy in the remaining sessions (4-11 in CUB and 5-9 in mini-ImageNet), where the challenges of avoiding catastrophic forgetting and learning from few-examples become more difficult. We note that for CUB experiments the average SEM was 0.4 and for mini-ImageNet it was 0.2. These results indicate that using GPs can indeed handle incremental learning challenges better than current procedures, but there is still room for improvement in how it handles the base classes. We also note that when examining the accuracy per session across all sessions at each time step, we noticed that GP-Tree showed a higher accuracy on the base classes compared to novel classes. This is an expected outcome since the number of novel examples is small and the feature extractor is kept fixed during novel sessions. Improving the way GP-Tree handles novel sessions can further boost its performance. In supplementary Section D.2 we perform sensitivity analysis on the kernel function choice and the number of representative samples stored per class. We show that non-linear kernel functions are preferred over a linear kernel function. In supplementary Section D.3 we evaluate GP-Tree against baseline methods according to the average forgetting (Chaudhry et al., 2018). We show that GP-Tree surpasses baseline methods on this aspect as well.

6. Conclusion

In this work, we showed how common Gaussian process classification methods struggle when facing classification tasks with a large number of classes. We presented our method, GP-Tree, that scales with the number of classes and to large datasets. GP-Tree uses the Pólya-Gamma augmentation and allows great flexibility in posterior inference that can be done either with a variational inference approach or a Gibbs sampling procedure. We further showed how GP-Tree can be naturally and successfully combined with DKL. Finally, we demonstrated how GP-Tree can be adjusted to few-shot class-incremental learning challenges and showed how it achieves improved accuracy over current leading baseline methods. This indicates that Gaussian processes are a new and promising approach for this task.

Acknowledgements

References

Adams, R. P., Ghahramani, Z., and Jordan, M. I. Tree-structured stick breaking for hierarchical data. In *24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, 2010.

Arthur, D. and Vassilvitskii, S. k-means++ the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035, 2007.

Bradshaw, J., Matthews, A. G. d. G., and Ghahramani, Z. Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold Gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 3338–3345. IEEE, 2016.

Castro, F. M., Marín-Jiménez, M. J., Guil, N., Schmid, C., and Alahari, K. End-to-end incremental learning. In *Proceedings of the European conference on computer vision*, pp. 233–248, 2018.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision*, pp. 532–547, 2018.

Damianou, A. and Lawrence, N. D. Deep Gaussian processes. In *Artificial intelligence and statistics*, pp. 207–215. PMLR, 2013.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Douillard, A., Cord, M., Ollion, C., Robert, T., and Valle, E. PODNet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, 2020.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135. PMLR, 2017.

Galy-Fajou, T., Wenzel, F., Donner, C., and Opper, M. Multi-class Gaussian process classification made conjugate: Efficient inference via data augmentation. In *Uncertainty in Artificial Intelligence*, pp. 755–765. PMLR, 2020.

Gidaris, S. and Komodakis, N. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4367–4375, 2018.

Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2014.

He, C., Wang, R., Shan, S., and Chen, X. Exemplar-supported generative reproduction for class incremental learning. In *BMVC*, pp. 98, 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*, pp. 770–778, 2016.

Hensman, J., Matthews, A., and Ghahramani, Z. Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pp. 351–360. PMLR, 2015.

Hernández-Lobato, D., Hernández-lobato, J., and Dupont, P. Robust multi-class Gaussian process classification. *Advances in neural information processing systems*, 24: 280–288, 2011.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

Hou, S., Pan, X., Loy, C. C., Wang, Z., and Lin, D. Learning a unified classifier incrementally via rebalancing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 831–839, 2019.

Izmailov, P., Novikov, A., and Kropotov, D. Scalable Gaussian processes with billions of inducing inputs via tensor train decomposition. In *International Conference on Artificial Intelligence and Statistics*, pp. 726–735. PMLR, 2018.

jun Liu, Y., Parisot, S., Slabaugh, G., Jia, X., Leonardis, A., and Tuytelaars, T. More classifiers, less forgetting: A generic multi-classifier paradigm for incremental learning. In *ECCV*, 2020.

Kingma, D. P. and Ba, J. ADAM: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., and Zhang, B.-T. Overcoming catastrophic forgetting by incremental moment matching. *Advances in Neural Information Processing Systems*, 30:4652–4662, 2017.

Linderman, S. W., Johnson, M. J., and Adams, R. P. Dependent multinomial models made easy: stick breaking with the Pólya-Gamma augmentation. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pp. 3456–3464, 2015.

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. When Gaussian process meets big data: A review of scalable GPs. *IEEE transactions on neural networks and learning systems*, 31 (11):4405–4423, 2020a.

Liu, Y., Su, Y., Liu, A.-A., Schiele, B., and Sun, Q. Mnemonics training: Multi-class incremental learning without forgetting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12245–12254, 2020b.

Mallya, A. and Lazebnik, S. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7765–7773, 2018.

Mallya, A., Davis, D., and Lazebnik, S. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision*, pp. 67–82, 2018.

McCloskey, M. and Cohen, N. J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Minka, T. P. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Mnih, A. and Hinton, G. E. A scalable hierarchical distributed language model. *Advances in neural information processing systems*, 21:1081–1088, 2008.

Morin, F. and Bengio, Y. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pp. 246–252. Citeseer, 2005.

Nassar, J., Linderman, S., Bugallo, M., and Park, I. Tree-structured recurrent switching linear dynamical systems for multi-scale modeling. In *International Conference on Learning Representations*, 2019.

Nemeth, C. and Fearnhead, P. Stochastic gradient Markov chain monte carlo. *Journal of the American Statistical Association*, pp. 1–18, 2020.

Nguyen, T. N. A., Bouzerdoum, A., and Phung, S. L. A scalable hierarchical Gaussian process classifier. *IEEE Transactions on Signal Processing*, 67(11):3042–3057, 2019.

Polson, N. G., Scott, J. G., and Windle, J. Bayesian inference for logistic models using Pólya–Gamma latent variables. *Journal of the American Statistical Association*, pp. 1339–1349, 2013.

Quinonero-Candela, J. and Rasmussen, C. E. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6: 1939–1959, 2005.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Ren, M., Liao, R., Fetaya, E., and Zemel, R. S. Incremental few-shot learning with attention attractor networks. In *Advances in Neural Information Processing Systems*, pp. 5276–5286, 2019.

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

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Sachs, M., Sen, D., Lu, J., and Dunson, D. Posterior computation with the Gibbs zig-zag sampler. *arXiv preprint arXiv:2004.04254*, 2020.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pp. 4528–4537. PMLR, 2018.

Sen, D., Sachs, M., Lu, J., and Dunson, D. B. Efficient posterior sampling for high-dimensional imbalanced logistic regression. *Biometrika*, 107(4):1005–1012, 2020.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *International Conference on Machine Learning*, pp. 4548–4557. PMLR, 2018.

Silverman, B. W. Some aspects of the spline smoothing approach to non-parametric regression curve fitting. *Journal*

of the Royal Statistical Society: Series B (Methodological), 47(1):1–21, 1985.

Snell, J. and Zemel, R. Bayesian few-shot classification with one-vs-each Pólya-Gamma augmented Gaussian processes. In *International Conference on Learning Representations*, 2021.

Snell, J., Swersky, K., and Zemel, R. Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4080–4090, 2017.

Snelson, E. and Ghahramani, Z. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pp. 1257–1264. MIT Press, 2006.

Taitelbaum, H., Chechik, G., and Goldberger, J. Network adaptation strategies for learning new classes without forgetting the original ones. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3637–3641. IEEE, 2019.

Tao, X., Hong, X., Chang, X., Dong, S., Wei, X., and Gong, Y. Few-shot class-incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. Functional regularisation for continual learning with Gaussian processes. In *International Conference on Learning Representations*, 2019.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 3637–3645, 2016.

Welinder, P., Branson, S., Mita, T., Wah, C., Schroff, F., Belongie, S., and Perona, P. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

Wenzel, F., Galy-Fajou, T., Donner, C., Kloft, M., and Opper, M. Efficient Gaussian process classification using Pólya-Gamma data augmentation. In *The AAAI Conference on Artificial Intelligence*, pp. 5417–5424. AAAI Press, 2019.

Williams, C. K. and Barber, D. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

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

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Stochastic variational deep kernel learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pp. 2594–2602, 2016b.

Wu, C., Herranz, L., Liu, X., van de Weijer, J., Raducanu, B., et al. Memory replay GANs: Learning to generate new categories without forgetting. *Advances in Neural Information Processing Systems*, 31:5962–5972, 2018.

Xian, Y., Lorenz, T., Schiele, B., and Akata, Z. Feature generating networks for zero-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5542–5551, 2018.

Yoon, J., Yang, E., Lee, J., and Hwang, S. J. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*, 2018.

Yu, L., Twardowski, B., Liu, X., Herranz, L., Wang, K., Cheng, Y., Jui, S., and Weijer, J. v. d. Semantic drift compensation for class-incremental learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6982–6991, 2020.

Zhai, M., Chen, L., Tung, F., He, J., Nawhal, M., and Mori, G. Lifelong GAN: Continual learning for conditional image generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2759–2768, 2019.

Supplementary Material for GP-Tree: A Gaussian Process Classifier for Few-Shot Incremental Learning

A. Variational Bound & Updates

In Section 3.2 we presented the following variational lower bound for the log marginal likelihood at each node:

$$\mathcal{C}(\boldsymbol{c}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) = \mathbb{E}_{p(\boldsymbol{f}|\bar{\boldsymbol{f}})q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega})}[log\, p(\boldsymbol{y}|\boldsymbol{\omega}, \boldsymbol{f})] - \\ KL(q(\bar{\boldsymbol{f}}, \boldsymbol{\omega}) \,||\, p(\bar{\boldsymbol{f}}, \boldsymbol{\omega})).$$

Here, we present the closed-form expression of it and the update rules for the variational parameters $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{\Sigma}}$ and $\boldsymbol{c}$. In the following constants are omitted for conciseness.

A.1. Explicit Form for the Variational Bound

We begin with the expectation term:

$$\mathbb{E}_{p(\boldsymbol{f}|\bar{\boldsymbol{f}})q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega})}[log\, p(\boldsymbol{y}|\boldsymbol{\omega}, \boldsymbol{f})]$$

$$\propto \mathbb{E}_{p(\boldsymbol{f}|\bar{\boldsymbol{f}})q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega})}[(\boldsymbol{y} - 1/2)^T \boldsymbol{f} - \frac{1}{2}\boldsymbol{f}^T \boldsymbol{\Omega} \boldsymbol{f}]$$

$$= \mathbb{E}_{q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega})}[(\boldsymbol{y} - 1/2)^T \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\bar{\boldsymbol{f}} - \frac{1}{2}Tr(\boldsymbol{\Omega}\boldsymbol{Q}_{nn})$$

$$- \frac{1}{2}\bar{\boldsymbol{f}}^T \boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}\boldsymbol{\Omega}\boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\bar{\boldsymbol{f}}]$$

$$= \frac{1}{2}\{2(\boldsymbol{y} - 1/2)^T \boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1} - Tr(\boldsymbol{\Lambda}\boldsymbol{Q}_{nn})$$

$$- Tr(\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}\boldsymbol{\Lambda}\boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\Sigma}})$$

$$- \tilde{\boldsymbol{\mu}}^T \boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}\boldsymbol{\Lambda}\boldsymbol{K}_{nm}\boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\mu}}\},$$

where $\lambda_i = \mathbb{E}_{q(\omega_i)}[\omega_i] = \frac{1}{2c_i}tanh(\frac{c_i}{2})$, $\boldsymbol{\Lambda} = diag(\lambda_i)$.

Now, we move to the KL divergence term. Due to independence between $p(\boldsymbol{\omega})$ and $p(\bar{\boldsymbol{f}})$, and the mean-field as a variational family assumption we have:

$$KL(q(\bar{\boldsymbol{f}}, \boldsymbol{\omega}) \,||\, p(\bar{\boldsymbol{f}}, \boldsymbol{\omega}))$$
$$= KL(q(\bar{\boldsymbol{f}})q(\boldsymbol{\omega}) \,||\, p(\bar{\boldsymbol{f}})p(\boldsymbol{\omega}))$$
$$= KL(q(\bar{\boldsymbol{f}}) \,||\, p(\bar{\boldsymbol{f}})) + KL(q(\boldsymbol{\omega}) \,||\, p(\boldsymbol{\omega})).$$

The first KL term is between two Gaussian distributions and has the following closed-form expression:

$$KL(q(\bar{\boldsymbol{f}}) \,||\, p(\bar{\boldsymbol{f}}))$$
$$= KL(\mathcal{N}(\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) \,||\, \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}_{mm})) \propto$$
$$\frac{1}{2}\{Tr(\boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\Sigma}}) + \tilde{\boldsymbol{\mu}}^T \boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\mu}} - log\,|\tilde{\boldsymbol{\Sigma}}| + log\,|\boldsymbol{K}_{mm}|\}.$$

The second KL term is between two Pólya-Gamma (PG) distributions, each of a mutually independent random variable,

and has a closed-form expression as well:

$$KL(q(\boldsymbol{\omega})||p(\boldsymbol{\omega})) = \sum_{i=1}^{n} KL(q(\omega_i) \,||\, p(\omega_i))$$

$$= \sum_{i=1}^{n} KL(PG(1,\, c_i) \,||\, PG(1,\, 0))$$

$$= \sum_{i=1}^{n} log\, cosh\frac{c_i}{2} - \frac{c_i}{4}tanh(\frac{c_i}{2}).$$

The variational lower bound is obtained by summing all these terms according to Eq. 13.

A.2. Variational Parameters Update

The update rules for the variational parameters are given by taking the derivative of Eq. 13 w.r.t each of $\boldsymbol{c}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$. At each iteration, based on a mini-batch of samples $\mathcal{B}$, we first update the parameters $\boldsymbol{c}^{\mathcal{B}} \subseteq \boldsymbol{c}$ corresponding to the samples seen in the batch using coordinate ascent scheme while holding $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$ fixed. Then, $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$ are updated according to a stochastic natural gradient ascent scheme.

The parameters $\boldsymbol{c}$ have a unique maximum which is given in a closed-form:

$$c_i = (Q_{ii} + \boldsymbol{K}_{im}\boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\Sigma}}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mi} + \\ \tilde{\boldsymbol{\mu}}^T \boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mi}\boldsymbol{K}_{im}\boldsymbol{K}_{mm}^{-1}\tilde{\boldsymbol{\mu}})^{\frac{1}{2}}, \quad (16)$$

where the subscript i denotes a specific row/column corresponding to the ith sample.

For the parameters $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$, the natural parameterization of the variational Gaussian distribution can be used: $\boldsymbol{\eta} = \tilde{\boldsymbol{\Sigma}}^{-1}\tilde{\boldsymbol{\mu}}$ and $\boldsymbol{H} = -\frac{1}{2}\tilde{\boldsymbol{\Sigma}}^{-1}$. The update at each batch then becomes:

$$\tilde{\nabla}_{\boldsymbol{\eta}}\mathcal{C} = \frac{n}{2|\mathcal{B}|}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}^{\mathcal{B}}(\boldsymbol{y}^{\mathcal{B}} - 1/2) - \boldsymbol{\eta},$$

$$\tilde{\nabla}_{\boldsymbol{H}}\mathcal{C} = -\frac{1}{2}(\boldsymbol{K}_{mm}^{-1} + \frac{n}{2|\mathcal{B}|}\boldsymbol{K}_{mm}^{-1}\boldsymbol{K}_{mn}^{\mathcal{B}}\boldsymbol{\Lambda}^{\mathcal{B}}\boldsymbol{K}_{nm}^{\mathcal{B}}\boldsymbol{K}_{mm}^{-1}) - \boldsymbol{H}.$$
$$(17)$$

Where we used the superscript $\mathcal{B}$ to denote only the rows/columns of samples in the batch. Note that the natural gradient updates maintain the positive-definiteness of $\tilde{\boldsymbol{\Sigma}}$.

B. Learning Algorithm with VI

Algorithm 1 summarizes GP-Tree learning with VI and DKL.

Algorithm 1 GP-Tree Inference with VI

Input: Data $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$, I_1 number of training iterations with a NN, I_2 Number of training iterations with GP-Tree
Init: g_θ a NN parameterized by θ
For $i = 1, ..., I_1$:
    - Sample a mini-batch of data from $\mathcal{D}$
    - Learn g_θ with a classification loss
End for
Build GP-Tree $\mathcal{T}$ as described in Section 3.1
Init: GP hyper-parameters ϕ, variational parameters $\boldsymbol{c}, \boldsymbol{\eta}, \boldsymbol{H}$, and inducing locations $\bar{\boldsymbol{X}}$ in the embedded space
For $i = 1, ..., I_2$:
    - Sample a mini-batch of data from $\mathcal{D}$
    - Obtain embedding for $\boldsymbol{X}$ with $g_\theta(\boldsymbol{X})$
    - Traverse the tree (e.g., via in-order traversal)
    For each node in the path:
        - Update $\boldsymbol{c}$ according to Eq. 16
        - Update $\boldsymbol{\eta}, \boldsymbol{H}$ according to Eq. 17
    End for
    - Update θ, ϕ and $\bar{\boldsymbol{X}}$ using Eq. 15 and by replacing the marginal likelihood terms with the variational lower bound $\mathcal{C}(\boldsymbol{c}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}})$ per node
End for
Return $g_\theta, \mathcal{T}, \bar{\boldsymbol{X}}$

C. Experimental Setup

This section provides further details about the experiments shown in Section 5.

C.1. Inference with Gibbs Sampling - Sec. 5.1

Data. We used the pre-trained features extracted by Xian et al. (2018) for the CUB 200-2011 dataset (Welinder et al., 2010). The CUB 200-2011 dataset contains 200 classes of bird species in 11,788 images with approximately 30 examples per class in the training set. Here, since the training set size is limited, we used all 5994 training instances according to the official split and we split the predefined test set to 2897 samples for validation and 2897 for testing.

Hyperparameter tuning. For all baselines, in all experiments, we applied a grid search over the kernel type, either normalized linear kernel or normalized RBF kernel (Snell & Zemel, 2021). We consistently found that under this setting the linear kernel generated better results (this was not true in other settings). The output scale for the linear kernel was chosen based on a grid search in $\{1, 4, 9, 18\}$. We used 20 Gibbs chains for the experiments with $\{4, 6, 8, 10, 20\}$ classes, 10 Gibbs chains for the experiments with 30 classes, and 1 Gibbs chain for the experiments with $\{40, 50, 60\}$ classes. For the OVE baseline, we were able to use only 10 chains for the experiments with 8 classes and 3 chains for the experiments with 10 classes. These experiments were

done on an NVIDIA V100 32GB GPU. We applied 1 Gibbs sampling step before taking $\boldsymbol{\omega}$ for the predictive distribution calculations. In these experiments, we often found it useful to make predictions with a single sample at the expected value location instead of using the 1D Gaussian-Hermite quadrature.

C.2. GPC with DKL - Sec. 5.2

In all experiments we trained from scratch a ResNet-18 (He et al., 2016) adjusted for CIFAR images size with a final embedding layer size of 1024. The Batch size was set to 256. We used SGD with a momentum of 0.9 and a scheduler that decays the learning rate by a factor of 0.1 at epochs 100 and 150. We allocated 10% from the training set for validation using stratified sampling. We applied a grid search over the initial learning rate in $\{0.1, 0.01\}$ for all methods. We found that an initial learning rate of 0.01 was preferred for our method. We used natural gradient descent with a learning rate of 0.05 for learning the variational parameters. In GPDNN experiments we also searched for an initial learning rate in $\{0.001, 0.0005\}$ and experimented with the Adam optimizer (Kingma & Ba, 2014). In all methods, we applied pre-training using a NN with a softmax layer after the last embedding layer and the cross-entropy loss. We searched over the number of pre-training epochs in $\{0, 20, 40, 60, 80\}$. For GP-Tree, 80 epochs yielded the best results. We used 40 inducing points per class in both GP-Tree and GPDNN experiments. For the SV-DKL baseline, we experimented with a grid size of $\{64, 128, 256\}$. In all experiments of all methods, we used the RBF kernel over L2 normalized input vectors. In CIFAR-100 experiments of the GPDNN baseline, we also applied an extensive grid search for the probability of labeling error without any success to achieve reasonable accuracy. In GP-Tree experiments, we found it beneficial to assign a weight to the loss term at each node that is inversely proportional to the amount of data used by that node for inference. It is achieved by dividing the loss at each node by the total number of training samples relevant for that node.

C.3. Few-Shot Class-Incremental Learning - Sec. 5.3

Experimental protocol. The experiments in this part largely followed the protocol suggested in (Tao et al., 2020) for comparability. We adopted the 10- way 5-shot setting for CUB, the first 100 classes were set as base classes, the remaining 100 classes were split into 10 incremental sessions. For mini-ImageNet, we followed the 5-way 5-shot, with 60 base classes, and 40 novel classes for a total of nine sessions. We used the official train/test split published by Tao et al. (2020). We pre-allocated a small portion from the training set of the base classes for a validation set. From the CUB dataset, we took 2 samples per class. From the mini-ImageNet dataset, we allocated 5% using stratified sampling.

Table 4. Class-incremental few-shot learning on CUB-200-2011. Tree construction variants. Test accuracy averaged over 10 runs.

Method	Sessions										
	1	2	3	4	5	6	7	8	9	10	11
Session Tree	73.73	**68.24**	64.07	59.42	56.12	52.80	50.73	48.89	46.34	45.01	43.33
Rebuild Tree	73.73	67.71	63.50	59.03	55.73	52.73	50.49	48.85	46.17	44.84	43.20
GP-Tree	73.73	**68.24**	**64.22**	**59.61**	**56.39**	**53.40**	**51.14**	**49.32**	**47.03**	**45.86**	**44.48**

In CUB experiments we fine-tuned a pre-trained ResNet-18 on ImageNet while in mini-ImageNet experiments we trained it from scratch. The final embedding layer size was set to 512. The mini-batch size at the first session was set to 128 in all experiments and, in later sessions, it included all available samples. We used SGD with a momentum of 0.9.

Hyperparameter tuning. In the experiments of GP-Tree, SDC (Yu et al., 2020) and PODNet (Douillard et al., 2020), we applied a grid search over the initial learning rate at the first session in $\{0.1, 0.01, 0.001\}$. In CUB experiments it was set to 0.01. In mini-ImageNet experiments, it was set to 0.1. At the first session, we trained the models for 100 epochs with a scheduler that decreased the learning rate by a factor of 0.1 at epochs 40 and 60. At later sessions, for our method, there was nothing to set, as it doesn't require any learning. In SDC and PODNet experiments we trained for 100 epochs and followed the training protocols suggested by each. For SDC, we applied a grid search over the embedding network in $\{LwF, MAS\}$, and γ, the hyper-parameter that controls the trade-off between the metric learning loss and the other losses, in $\{5e-5, 5e-4, 5e-3, 5e-2, 5e-1, 1e4, 1e6\}$. For this baseline, we found it beneficial to start with a few epochs of training using a softmax layer and the cross-entropy loss and only afterward train with the triplet loss as advocated in the paper. When training with the triplet loss, we also needed to optimize the ratio of positive and negative examples at each batch to make it work.

GP-Tree configurations. In GP-Tree experiments, at the initial session, we first applied a few epochs of training using a NN only with a softmax layer and the cross-entropy loss. Then, after 20 epochs in CUB experiments, and 40 epochs in mini-ImageNet experiments, we transitioned to learning with GP-Tree as described in Section 3. We used 5 inducing points per class and the RBF kernel on all GPs with an initial length-scale of 1 and an initial output-scale of 4. We learned the variational parameters with natural gradient descent and a learning rate of 0.05. Here, as well, the inputs to the kernel were normalized by their L2 norm. During training, we assigned a weight to the loss term at each node that is inversely proportional to the amount of data used by that node for inference. During novel sessions, we used the RBF kernel with a fixed length-scale of 1 and a fixed output-scale of 8. At the end of each few-shot session,
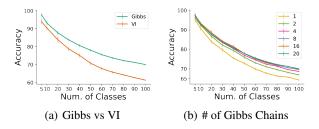

(a) Gibbs vs VI   (b) # of Gibbs Chains

Figure 4. Test accuracy when varying the number of classes. **Left** Gibbs sampling vs variational inference, **Right** varying the number of Gibbs chains. Results are the average over 10 runs ($\pm$ SEM) on pre-trained features of samples from the CUB 200-2011 dataset.

we saved the 512 dimensional representation of the samples for later sessions.

D. Additional Experiments

D.1. GP-Tree Inference

The performance of GP-Tree depends on several factors. Here, we test (1) the effect of using VI against using a Gibbs sampler and, (2) the effect of the number of Gibbs chains. Both analyses were made by observing the test accuracy on the CUB-200-2011 dataset under the setup presented in Section 5.1. The results are shown in Figure 4. Figure 4(a) shows a large performance gap in favor of the Gibbs sampling. This result is not surprising since the Gibbs sampler, asymptotically, samples from the correct posterior while in VI we use an approximate one. The figure also shows that the gap is amplified as the number of classes increases. This result is another justification for using Gibbs sampling when learning novel classes under the incremental learning setup. Figure 4(b) shows that as we increase the number of chains the accuracy increase as well; however, the difference is marginal when using four chains or more.

D.2. Sensitivity Analysis for FSCIL

When we adjusted GP-Tree to the few-shot class-incremental learning setting, we made several design choices. Here, we examine some of them. We will show that GP-Tree is fairly robust to these choices.
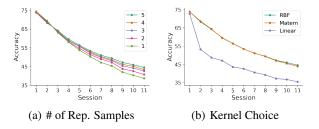
(a) # of Rep. Samples   (b) Kernel Choice

Figure 5. Test accuracy for few-shot class-incremental learning as a function of the number of representative samples per class (left) and the kernel choice (right). Results are the average over 5 runs on the CUB 200-2011 dataset.

Figure 6. Average forgetting across sessions. Results are the average over 5 runs on the CUB 200-2011 dataset.

Tree construction. You may recall that under the FSCIL setup, after learning on the base classes, we retain the original tree intact, and at each novel session t, we build a sub-tree from samples' representations that appeared in sessions $D_2, ..., D_t$. We then connect this sub-tree to the base classes tree with a shared root node. In Table 4 we present two alternatives for the tree construction used for inference during novel sessions ($t > 1$). (1) *Session Tree*, instead of building a tree from samples in all sessions $D_2, ..., D_t$, in each session we build a tree from classes that appeared at the current session only. Then, we connect this sub-tree with the tree that is already built via a shared root; (2) *Rebuild Tree*, building the entire tree at each new session and fitting the Gibbs sampling variant of our approach to each node. To account for base classes we use the inducing inputs. Table 4 shows that both alternatives yield good results; however, the approach chosen for GP-Tree is superior.

Kernel analysis. The results presented in the main paper for few-shot class-incremental learning were with the RBF kernel and 5 representative samples per class. Here we investigate both choices in Figure 5. Figure 5(a) compares between $1 - 5$ representative samples per class. The figure shows that all alternatives achieve high accuracy across all sessions; however, as expected, when using less representative samples there is a slight degradation in performance. The impact becomes more severe in later sessions. Figure 5(b) shows a comparison between the RBF, Matern 5/2 and Linear kernels. Similar to (Snell & Zemel, 2021) we found gain in normalizing the inputs to the kernels by their L2 norm. Therefore, we applied this method in all settings and for all kernel choices. However, unlike in the few-shot learning case presented in (Snell & Zemel, 2021), in which the normalized linear kernel (also referred to as cosine kernel in that study) yielded the best results, in our case either the RBF kernel or the Matern kernel are preferred by a large margin, especially on novel sessions. We hypothesis that on the base classes the NN outputs a representation that is more linearly separable. Therefore, all kernels perform similarly on them. However, the representation of novel classes is
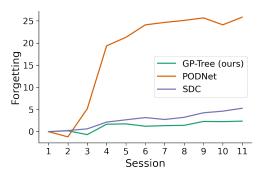
more mixed in the embedding space. Therefore, a stronger kernel that generates non-linear decision boundaries is required.

D.3. Forgetting Across Sessions

In this part, we examine how GP-Tree performance varies across sessions through the *average forgetting* (Chaudhry et al., 2018). The average forgetting was designed to estimate the forgetting of prior tasks. Let α_j^k denote the accuracy of the learner on the jth task at session $k > j$. The forgetting of the jth task is defined as $g_j^k = \max_{l \in \{1,...,k-1\}} \alpha_j^l - \alpha_j^k$. This quantity is measured for every task j seen thus far at each new session. Then, to get an estimate of the average forgetting we may use: $\frac{1}{k-1} \sum_{j=1}^{k-1} g_j^k$. Figure 6 shows the average forgetting across all classes and sessions on the CUB 200-2011 dataset for GP-Tree, SDC (Yu et al., 2020), and PODNet (Douillard et al., 2020). From the figure, we notice a minor forgetting for GP-Tree. When comparing the performance of the three methods across all sessions, we noticed that GP-Tree and SDC showed better performance on the base classes while PODNet was more balanced with a slight advantage to the novel classes.