Machine Learning (2023) 112:1077–1106
https://doi.org/10.1007/s10994-022-06289-3

# Large scale multi-output multi-class classification using Gaussian processes

Chunchao Ma[1] · Mauricio A. Álvarez[2]

Received: 16 July 2021 / Revised: 16 November 2022 / Accepted: 24 November 2022 /
Published online: 8 February 2023

**Abstract**
Multi-output Gaussian processes (MOGPs) can help to improve predictive performance for some output variables, by leveraging the correlation with other output variables. In this paper, our main motivation is to use multiple-output Gaussian processes to exploit correlations between outputs where each output is a multi-class classification problem. MOGPs have been mostly used for multi-output regression. There are some existing works that use MOGPs for other types of outputs, e.g., multi-output binary classification. However, MOGPs for multi-class classification has been less studied. The reason is twofold: 1) when using a softmax function, it is not clear how to scale it beyond the case of a few outputs; 2) most common type of data in multi-class classification problems consists of image data, and MOGPs are not specifically designed to image data. We thus propose a new MOGPs model called *Multi-output Gaussian Processes with Augment & Reduce (MOGPs-AR)* that can deal with large scale classification and downsized image input data. Large scale classification is achieved by subsampling both training data sets and classes in each output whereas downsized image input data is handled by incorporating a convolutional kernel into the new model. We show empirically that our proposed model outperforms single-output Gaussian processes in terms of different performance metrics and multi-output Gaussian processes in terms of scalability, both in synthetic and in real classification problems. We include an example with the Ommiglot dataset where we showcase the properties of our model.

**Keywords** Gaussian processes · Multi-output Gaussian processes · Image data · Classification · Transfer learning

Editor: Willem Waegeman.

✉ Chunchao Ma
cma15@sheffield.ac.uk

Mauricio A. Álvarez
mauricio.alvarezlopez@manchester.ac.uk

[1] Department of Computer Science, University of Sheffield, Sheffield, UK

[2] Department of Computer Science, University of Manchester, Manchester, UK

## 1 Introduction

Multi-output Gaussian processes (MOGPs), a generalisation of Gaussian processes (GPs), exploit dependencies among outputs to improve joint prediction performances (Bonilla et al., 2008; Álvarez et al., 2012; Dahl & Bonilla, 2019; Nguyen et al., 2018; Wistuba et al., 2018; Alvarez, 2011). For example, in a sensor network, prediction of missing signals from some sensors may be done by exploiting dependencies with signals obtained from nearby sensors (Osborne et al., 2008).

Our main purpose in this paper is to use MOGPs to study the problem of multiple outputs where each output is a multi-class classification problem. The setting considered here goes beyond multi-label classification since we allow each output to potentially have its own inputs moving into the multi-task setting.

MOGPs have mainly been used for multi-output regression to predict continuous variables (Bonilla et al., 2008; Álvarez et al., 2012; Dai et al., 2017). In this setting, the assumption is that each output follows a Gaussian likelihood and the mean of the Gaussian likelihood is given by one output of the MOGP. Due to the properties of the Gaussian distribution, Bayesian inference is tractable in this case.

Beyond the muti-output regression problem, there is some research on other types of outputs in MOGPs. For example, Skolidis and Sanguinetti (2011) use MOGPs to model a setting where each output corresponds to a binary classification problem. Each binary outcome is modelled using a probit likelihood. The MOGP corresponds to the so called intrinsic coregionalisation model (ICM) (Bonilla et al., 2008). Since Bayesian inference is intractable in this model, the authors approximate posterior distributions using expectation-propagation and variational Bayes.

Several research works have addressed the case of multi-class classification using GPs. Previous works have used the softmax likelihood (Williams & Rasmussen, 2006; Kim & Ghahramani, 2006; Galy-Fajou et al., 2020), the multinomial probit likelihood function (Girolami & Rogers, 2006), the step function (Hernández-Lobato et al., 2011). Recently, Liu et al. (2019) can use all the above likelihoods through additive noise terms. The parameters in these likelihood functions are assumed to follow independent Gaussian processes. Another strand of works generalise this setting by allowing correlated Gaussian processes for the latent parameters of the likelihood functions, typically using MOGPs. Both Dezfouli and Bonilla (2015) and Chai (2012) use an ICM for a single-output multi-class classification problem modelled through a multinomial logistic likelihood, i.e. the softmax likelihood. In terms of Bayesian inference, Chai (2012) proposes a variational sparse approximation for the posterior distribution, and based on scalable automated variational inference, Dezfouli and Bonilla (2015) approximates the posterior distribution by a mixture of Gaussians. Moreno-Muñoz et al. (2018) build a heterogeneous multi-output Gaussian process, where each output has its own likelihood, through a linear model of coregionalisation (LMC) (Álvarez et al., 2012). Moreno-Muñoz et al. (2018) use an stochastic variational approach for Bayesian inference.

The approaches for single-output multi-class classification described above are restricted to the case where the number of classes is small. They scale poorly when the number of classes go beyond a few tens. Scalability is also poorly handled by the more general model of Moreno-Muñoz et al. (2018) for the multi-output multi-class classification case, where once again, problems that go beyond a few tens of classes are out of reach.

Our main contribution in this paper is that we introduce a new extension of multi-output GPs able to handle large scale multi-output multi-class classification problems, typically in the range of hundreds and even thousands of classes. We achieve scalability by subsampling both training input data and classes in each output, by using stochastic variational inference (Hensman et al., 2013; Moreno-Muñoz et al., 2018), and by choosing a softmax likelihood function via Gumbel noise error for all outputs. We refer to this model as *Multi-output Gaussian Processes with Augment & Reduce (MOGPs-AR)*.

We also enable our MOGPs-AR to allow downsized images as input data. To efficiently deal with downsized images, we employ convolutional kernels (Van der Wilk et al., 2017), computing the entries of the kernel matrices using kernels over patches of the images and integrating these kernels within a MOGP. Since our model is able to capture both intra- and inter-output dependencies, it also provides a means to perform transfer learning in the multi-task setting. We show an example of this multi-task learning ability of our model in the Ommiglot dataset. To the best of our knowledge, this is the first time that a multi-task multi-class Gaussian process model is used over such dataset.

## 2 Related work

As we mentioned early, the multi-class classification problem has been mainly studied using single-output GPs (Williams & Rasmussen, 2006; Kim & Ghahramani, 2006; Hernández-Lobato et al., 2011; Girolami & Rogers, 2006; Liu et al., 2019). The model introduced in this paper, MOGPs-AR, uses the softmax likelihood through additive noise errors, which is the same as Liu et al. (2019). However, MOGPs-AR solves multiple outputs problems together while the model in Liu et al. (2019), like all single-output GPs, only solves single output problems. Regarding single output problems, MOGPs-AR can also improve prediction using a correlation between all latent parameter functions whereas single-output GPs cannot capture the correlation.

The works more relevant to ours are Chai (2012); Dezfouli and Bonilla (2015); Skolidis and Sanguinetti (2011); Moreno-Muñoz et al. (2018). Both Chai (2012) and Dezfouli and Bonilla (2015) can only handle a single output multi-class classification problem even if they use MOGPs. Nevertheless, our model can tackle multiple outputs where each output is a multi-class classification problem. Skolidis and Sanguinetti (2011) only solve multi-output binary classification problems, which is different to ours. Compared with Skolidis and Sanguinetti (2011), our inference method is also suited to large scale data sets. Moreno-Muñoz et al. (2018) can tackle multi-output multi-class classification problems and develop a similar stochastic variational inference method as us. However, we are different to Moreno-Muñoz et al. (2018) since we can cope with a large number of classes by subsampling classes and also can deal with downsized images through convolutional kernels (Van der Wilk et al., 2017).

The work by Panos et al. (2021) is much related to us since we use a similar subsampling method. Panos et al. (2021) extend a semiparametric latent model, a special case of LMC, to address the multi-label problem by using sigmoidal/Bernoulli likelihood for each latent parameter function. Panos et al. (2021) can doubly subsample data points and classes to reduce computational complexity based on stochastic variational inference, which is analogous to us. However, we are different in other aspects. First, we solve multi-class classification problems using the softmax likelihood instead of multi-label problems using sigmoidal/Bernoulli likelihood. Further, we can apply a convolutional kernel to handle downsized image data. Finally, our model can deal with multi-output problems instead of only tackling single output problems.

**Table 1** Nomenclature

| Notation | Description |
|---|---|
| $v$ | dimension of the input space |
| $D$ | number of outputs |
| $N$ | number of data points per output |
| $C$ | number of classes for all outputs $C = \sum_{d=1}^{D} C_d$ where $D \geq 1$ |
| $M$ | number of inducing points or inducing patches |
| $Q$ | number of latent functions $u_q(\mathbf{x})$ |
| $u_q(\mathbf{x})$ | $q$-th latent function evaluated at $\mathbf{x}$ |
| $u_q^i(\mathbf{x})$ | $i$-th sample of $u_q(\mathbf{x})$ drawn independent and identically distributed |
| $R_q$ | number of latent functions $u_q^i(\mathbf{x})$ |
| $f_d^c(\mathbf{x})$ | $c$-th latent parameter function in the $d$-th output evaluated at $\mathbf{x}$ |
| $k_q(\mathbf{x}, \mathbf{x}')$ | Gaussian process covariance function of $u_q(\mathbf{x})$ |
| $k_{f_d^c f_{d'}^{c'}}(\mathbf{x}, \mathbf{x}')$ | covariance between latent functions $f_d^c(\mathbf{x})$ and $f_{d'}^{c'}(\mathbf{x}')$ |
| $\mathbf{K}_{\mathbf{f}_d^c \mathbf{f}_{d'}^{c'}}$ | covariance matrix with entries given by $k_{f_d^c f_{d'}^{c'}}(\mathbf{x}_n, \mathbf{x}_m)$ with $\mathbf{x}_n, \mathbf{x}_m \in \mathbf{X}$ |

## 3 Methodology

In this section, we will derive the MOGPs-AR model. We first develop the LMC model with a convolutional kernel. We then define the softmax likelihood through augmenting noise data. We finally describe stochastic variational inference and the approximated predictive distribution for our model.

We assume there are $D$ different outputs (Table 1 shows the description of our notation). The vector $\mathbf{y} \in \mathbf{R}^D$ groups all the $D$ different outputs:

$$\mathbf{y}(\mathbf{x}) = \left[y_1(\mathbf{x}), y_2(\mathbf{x}), \cdots, y_D(\mathbf{x})\right]^\top, \tag{1}$$

where $\mathbf{x} \in \mathbf{R}^v$. Each output $y_d(\mathbf{x}) \in \{1, ..., C_d\}(C_d \geq 2$ and $d \in \{1, ..., D\})$ is a categorical variable and $C_d$ is the number of classes in the $d$-th output. Like Moreno-Muñoz et al. (2018), we also assume that those outputs are conditionally independent given parameters $\boldsymbol{\theta}(\mathbf{x}) = \left[\theta_1(\mathbf{x}), \theta_2(\mathbf{x}), \cdots, \theta_D(\mathbf{x})\right]^\top$, where $\boldsymbol{\theta}(\mathbf{x})$ is defined by latent parameter functions:

$$\mathbf{f}(\mathbf{x}) = \left[f_1^1(\mathbf{x}), f_1^2(\mathbf{x}), \cdots f_1^{C_1}(\mathbf{x}), f_2^1(\mathbf{x}), f_2^2(\mathbf{x}), \cdots, f_D^{C_D}(\mathbf{x})\right]^\top \in \mathbf{R}^{C \times 1}, \tag{2}$$

where $C = \sum_{d=1}^{D} C_d$ and $f_d^c(\mathbf{x})$ is $c$-th latent parameter function in the $d$-th output evaluated at $\mathbf{x}$. We then obtain:

$$p(\mathbf{y}(\mathbf{x})|\boldsymbol{\theta}(\mathbf{x})) = p(\mathbf{y}(\mathbf{x})|\mathbf{f}(\mathbf{x})) \tag{3}$$

$$= \prod_{d=1}^{D} p\left(y_d(\mathbf{x})|\boldsymbol{\theta}_d(\mathbf{x})\right) \tag{4}$$

$$= \prod_{d=1}^{D} p\left(y_d(\mathbf{x})|\widetilde{\mathbf{f}}_d(\mathbf{x})\right), \tag{5}$$

where $\widetilde{\mathbf{f}}_d(\mathbf{x}) = \left[f_d^1(\mathbf{x}), \cdots, f_d^{C_d}(\mathbf{x})\right]^\top \in \mathbf{R}^{C_d \times 1}$ is a group of latent parameter functions defining the parameters in $\boldsymbol{\theta}_d(\mathbf{x})$.

## 3.1 Combining with convolutional kernel

We use the linear model of coregionalisation (LMC) and combine it with the convolutional kernel. The LMC is a popular model in MOGPs, where each output is expressed as a linear combination of a collection of Gaussian processes (Álvarez et al., 2012). The convolutional kernel (Van der Wilk et al., 2017) can effectively exploit features in images.

We construct a convolutional structure for mutually independent latent functions $\mathcal{U} = \left\{u_q(\mathbf{x})\right\}_{q=1}^{Q}$ where $u_q$ follows a Gaussian process, $Q$ is the number of the latent functions and each latent parameter function $f_d^c(\mathbf{x})$ is a linear combination of the latent functions $\mathcal{U}$. Here, we assume $\mathbf{x} \in \mathbf{R}^{W \times H}$ is an image data point that has a $v = W \times H$ size where $W$ and $H$ are the width and height of the image separately. We also assume $\mathbf{x}^{[p]}$ is the $p$th patch of $\mathbf{x}$ with patches of size $E = w \times h$ where $w$ and $h$ are the width and height of each patch, respectively.

After dividing an image into patches, we get a total of $P = (W - w + 1) \times (H - h + 1)$ patches. We begin with a patch response function $u_q\left(\mathbf{x}^{[p]}\right) : \mathbf{R}^{w \times h} \to \mathbf{R}$, which maps a patch of size $E = w \times h$ to a real number in $\mathbf{R}$. Then we add a weight for each patch response function and get a latent function $u_q(\mathbf{x}) : \mathbf{R}^{W \times H} \to \mathbf{R}$, where $u_q(\mathbf{x})$ is the sum of all patch responses with weights: $u_q(\mathbf{x}) = \sum_p w_p u_q\left(\mathbf{x}^{[p]}\right)$. Each function $u_q$ is drawn from an independent GP prior: $u_q(\cdot) \sim \mathcal{GP}\left(0, k_q(\cdot,\cdot)\right)$, where $k_q(\cdot,\cdot)$ can be any kernel function. In this paper, we use the radial basis function kernel with automatic relevance determination (RBF-ARD) (Williams & Rasmussen, 2006):

$$k_{\mathrm{ard}}\left(\mathbf{x}^{[p]}, \mathbf{x}^{[p']}\right) = \sigma_{\mathrm{ard}}^2 \exp\left(-\frac{1}{2}\sum_{j=1}^{E} \frac{\left(x_j^{[p]} - x_j^{[p']}\right)^2}{l_j^2}\right), \tag{6}$$

where $x_j^{[p]}$ is the $j$-th dimension of $\mathbf{x}^{[p]}$, $\sigma_{\mathrm{ard}}^2$ is a variance parameter and $l_j$ is the length scale for the $j$-th input dimension. Therefore $k_q(\cdot,\cdot)$ is RBF-ARD. When all length scales are the same, the kernel is called radial basis function kernel (RBF) (Lawrence & Hyvärinen, 2005). Hence, each $f_d^c(\mathbf{x})$ is defined as

$$f_d^c(\mathbf{x}) = \sum_{q=1}^{Q}\sum_{i=1}^{R_q} a_{d,c,q}^i u_q^i(\mathbf{x}) = \sum_{q=1}^{Q}\sum_{i=1}^{R_q} a_{d,c,q}^i \left(\sum_{p=1}^{P} w_p u_q^i(\mathbf{x}^{[p]})\right), \tag{7}$$

where $a_{d,c,q}^i \in \mathbf{R}$ can be considered as a weight on $\mathcal{U}$ and we assume $\{\phi_q\}_{q=1}^{Q}$ are the hyperparameters for $\{k_q(\cdot,\cdot)\}_{q=1}^{Q}$, with $\phi_q$ being the hyperparameters for the kernel $k_q(\cdot,\cdot)$. $R_q$ represents the number of latent functions $u_q^i(\mathbf{x})$ that are sampled independently and identically from the Gaussian processes $u_q(\cdot) \sim \mathcal{GP}\left(0, k_q(\cdot,\cdot)\right)$. The difference between the convolutional kernel model and a more classic kernel, e.g., RBF, is that we use the convolutional structure term $\sum_{p=1}^{P} w_p u_q^i(\mathbf{x}^{[p]})$ instead of solely $u_q^i(\mathbf{x})$, where Fig. 1 shows an example of two images and how they are handled through the convolutional kernel. With $q = 1, ..., Q$

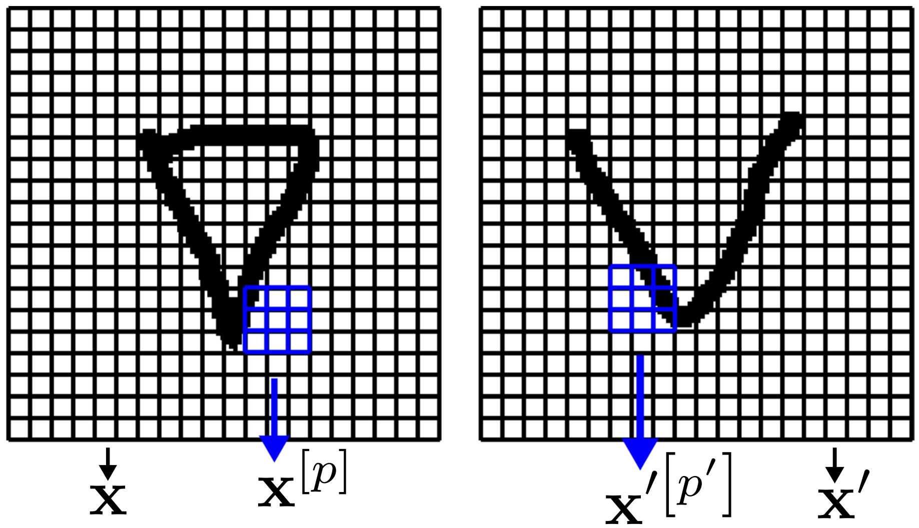


**Fig. 1** An example of two images for the convolutional kernel inputs. The two images are two characters in the Ojibwe alphabet (please see Sect. 4.4.1 for more detail). We consider two characters as two classes. The two images are one data point for each class separately. Left: The whole image is considered as an input data point $\mathbf{x}$ and the blue grid represents the $p$-th patch $\mathbf{x}^{[p]}$. Right: The whole image is considered as an input data point $\mathbf{x}'$ and the blue grid represents the $p'$-th patch $\mathbf{x}'^{[p']}$ (Color figure online)

and $i = 1, ..., R_q$, the function $u_q^i(\mathbf{x})$ have a zero mean and covariance $\text{cov}\left[u_q^i(\mathbf{x}), u_{q'}^{i'}(\mathbf{x}')\right] = \sum_{p=1}^{P} \sum_{p'=1}^{P} w_p w_{p'} k_q\left(\mathbf{x}^{[p]}, \mathbf{x}'^{[p']}\right)$ if $i = i'$ and $q = q'$. Let the mean function of $f_d^c(\mathbf{x})$ be zero and the cross-covariance function of $f_d^c(\mathbf{x})$ be

$$k_{f_d^c f_{d'}^{c'}}\left(\mathbf{x}, \mathbf{x}'\right) = \text{cov}\left[f_d^c(\mathbf{x}), f_{d'}^{c'}\left(\mathbf{x}'\right)\right] \tag{8}$$

$$= \text{cov}\left[\sum_{q=1}^{Q} \sum_{i=1}^{R_q} a_{d,c,q}^i u_q^i(\mathbf{x}), \sum_{q'=1}^{Q} \sum_{i'=1}^{R_q} a_{d',c',q'}^{i'} u_{q'}^{i'}(\mathbf{x}')\right] \tag{9}$$

$$= \sum_{q=1}^{Q} \sum_{q'=1}^{Q} \sum_{i=1}^{R_q} \sum_{i'=1}^{R_q} a_{d,c,q}^i a_{d',c',q'}^{i'} \text{cov}\left[u_q^{i'}(\mathbf{x}), u_{q'}^{i'}(\mathbf{x}')\right]. \tag{10}$$

Because $u_q^i(\cdot)$ is independently and identically drawn from $u_q(\cdot)$ and $\mathcal{U}(\cdot)$ are mutually independent

$$k_{f_d^c f_{d'}^{c'}}\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{q=1}^{Q} b_{(d,c),(d',c')}^q \left[\sum_{p=1}^{P} \sum_{p'=1}^{P} w_p w_{p'} k_q\left(\mathbf{x}^{[p]}, \mathbf{x}'^{[p']}\right)\right], \tag{11}$$

where $b_{(d,c),(c',c')}^q = \sum_{i=1}^{R_q} a_{d,c,q}^i a_{d',c',q}^i$. For simplicity in the presentation, we assume that all outputs $y_d(\mathbf{x})$ have a collection of the same input vectors $\mathbf{X} = \left\{\mathbf{x}_n\right\}_{n=1}^{N} \in \mathbf{R}^{N \times v}$. Our model also works for each output with a different input data set. For notation simplicity, we define

$$\mathbf{f}_d^c = \left[f_d^c(\mathbf{x}_1), \cdots, f_d^c(\mathbf{x}_N)\right]^\top \in \mathbf{R}^{N\times 1} \tag{12}$$

$$\widetilde{\mathbf{f}}_d = \left[\left(\mathbf{f}_d^1\right)^\top, \cdots, \left(\mathbf{f}_d^{C_d}\right)^\top\right]^\top \in \mathbf{R}^{C_d N\times 1} \tag{13}$$

$$\mathbf{f} = \left[\widetilde{\mathbf{f}}_1^\top, \cdots, \widetilde{\mathbf{f}}_D^\top\right]^\top \in \mathbf{R}^{CN\times 1} \tag{14}$$

The prior distribution of $\mathbf{f}$ is given by $\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$, where $\mathbf{K}$ is a block-wise matrix based on $\left\{\mathbf{K}_{\mathbf{f}_d^c \mathbf{f}_{d'}^{c'}}\right\}_{d=1,d'=1,c=1,c'=1}^{D,D,C_d,C_{d'}}$ as each block and $\mathbf{K}_{\mathbf{f}_d^c \mathbf{f}_{d'}^{c'}}$ has entries computed using $k_{f_d^c f_{d'}^{c'}}\left(\mathbf{x}_n, \mathbf{x}_m\right)$ with $\mathbf{x}_n, \mathbf{x}_m \in \mathbf{X}$. $\mathbf{K}$ can be formulated as a sum of Kronecker products $\mathbf{K} = \sum_{q=1}^{Q} \mathbf{A}_q \mathbf{A}_q^\top \otimes \mathbf{K}_q = \sum_{q=1}^{Q} \mathbf{B}_q \otimes \mathbf{K}_q$ as well, where $\mathbf{A}_q \in \mathbf{R}^{C\times R_q}$ and $\mathbf{B}_q$ have entries $\left\{a_{d,c,q}^i\right\}_{d=1,c=1,i=1}^{D,C_d,R_q}$ and $\left\{b_{(d,c),(d',c')}^q\right\}_{d=1,d'=1,c=1,c'=1}^{D,D,C_d,C_{d'}}$, respectively. $\mathbf{K}_q \in \mathbf{R}^{N\times N}$ has entries computed using $\sum_{p=1}^{P} \sum_{p'=1}^{P} w_p w_{p'} k_q\left(\mathbf{x}_n^{[p]}, \mathbf{x}_m^{[p']}\right)$ for $\mathbf{x}_n, \mathbf{x}_m \in \mathbf{X}$. Each matrix $\mathbf{B}_q \in \mathbf{R}^{C\times C}$ is known as a *coregionalisation matrix* and it controls the correlation between each latent parameter function.

## 3.2 Augmenting model by noise data

In this section, we generalise the model in the last subsection to cope with the multi-output multi-class classification problem using the softmax likelihood. We derive a softmax likelihood function through Gumbel noise error for a generic output $\mathbf{y}_d$.

We take the $d$-th output $y_d(\mathbf{x})$ with the latent parameter function $\widetilde{\mathbf{f}}_d(\mathbf{x}) = \left[f_d^1(\mathbf{x}), \cdots, f_d^{C_d}(\mathbf{x})\right]^\top$. We first add a Gumbel noise error to each of latent parameter functions $\widetilde{\mathbf{f}}_d(\mathbf{x})$ to get a new vector function $\mathbf{h}_d(\mathbf{x}) = \left\{h_d^c(\mathbf{x})\right\}_{c=1}^{C_d}$ for each of the classes in the $d$-th output. We thus obtain:

$$h_d^c(\mathbf{x}) = f_d^c(\mathbf{x}) + \epsilon_{d,i}^c, \tag{15}$$

$$y_d(\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, h_d^c(\mathbf{x}), \tag{16}$$

where $\epsilon_{d,i}^c$ is the $i$-th i.i.d. Gumbel noise error for class $c$ in the $d$-th output. We define $\mathbf{h}_d(\mathbf{x}_i) = \left(h_d^1(\mathbf{x}_i), \cdots, h_d^{C_d}(\mathbf{x}_i)\right)^\top \in \mathbf{R}^{C_d\times 1}$. We then employ the internal step likelihood (Liu et al., 2019):

$$p\left(y_d(\mathbf{x}_i) | \mathbf{h}_d(\mathbf{x}_i)\right) = \prod_{c\neq y_d(\mathbf{x}_i)} H\left(h_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) - h_d^c(\mathbf{x}_i)\right) \tag{17}$$

$$= \prod_{c\neq y_d(\mathbf{x}_i)} H\left(f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) + \epsilon_{d,i}^{y_d(\mathbf{x}_i)} - f_d^c(\mathbf{x}_i) - \epsilon_{d,i}^c\right), \tag{18}$$

where $H(z) = 1$ when $z > 0$; otherwise, $H(z) = 0$; $y_d(\mathbf{x}_i)$ is $i$-th point in $d$-th output. By integrating $\mathbf{h}_d(\mathbf{x}_i)$ out, we get

$$p\left(y_d(\mathbf{x}_i)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) = \int p(y_d(\mathbf{x}_i)|\mathbf{h}_d(\mathbf{x}_i))p\left(\mathbf{h}_d(\mathbf{x}_i)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)d\mathbf{h}_d(\mathbf{x}_i) \tag{19}$$

$$= \int \phi_{\mathcal{G}}(\epsilon_{d,i}) \prod_{c\neq y_d(\mathbf{x}_i)} \Phi_{\mathcal{G}}\left(\epsilon_{d,i} + f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) - f_d^c(\mathbf{x}_i)\right)d\epsilon_{d,i}, \tag{20}$$

where $\phi_{\mathcal{G}}$ and $\Phi_{\mathcal{G}}$ are the probability density function and the cumulative distribution function of the Gumbel distribution, respectively. (NB: We drop out the c in $\epsilon^c_{d,i}$ for convenience since all the $\epsilon^c_{d,i}$ are from the same Gumbel error distribution). Now, we assume the Gumbel error $\epsilon_{d,i} \sim \phi_{\mathcal{G}}(\epsilon_{d,i}|0,1)$ so we obtain $\phi_{\mathcal{G}}(\epsilon_{d,i}) = \exp\left(-\epsilon_{d,i} - e^{-\epsilon_{d,i}}\right)$ and $\Phi_{\mathcal{G}}(\epsilon_{d,i}) = \exp(-e^{-\epsilon_{d,i}})$. From expression (20), we recover a softmax likelihood (Liu et al., 2019; Ruiz et al., 2018):

$$p\left(y_d(\mathbf{x}_i)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) = \frac{\exp\left(f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)\right)}{\sum_{c=1}^{C_d}\exp\left(f_d^c(\mathbf{x}_i)\right)}. \tag{21}$$

The softmax likelihood is a common likelihood used in multi-class classification with Gaussian processes (Williams & Rasmussen, 2006). As we mentioned in expression (5), all outputs are conditional independent given the corresponding latent parameter functions so each output has its own likelihood expression (20).

## 3.3 Scalable variational inference

We have derived the LMC model with a convolutional kernel and used the softmax likelihood. However, there exists a computational challenge if there are a very large number of classes and training instances in each output. We thus use scalable variational inference to reduce the computational complexity by the techniques of inducing patches and subsampling, where we refer our model to *Multi-output Gaussian Processes with Augment & Reduce (MOGPs-AR)*. Inducing patches can ease the computational complexity of the inversion of a kernel matrix from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$, where $N$ is the number of data points per output and $M$ is the number of inducing patches ($M \ll N$). Subsampling reduces the computational complexity of our model using a subset of both training data and classes for each output during hyperparameters and parameters optimisation.

### 3.3.1 Inducing patches for MOGPs-AR

We assume we use image data sets in this section. We define the inducing patches (Van der Wilk et al., 2017) at the latent functions $\mathcal{U}$. If our input data sets are not image data sets, we use inducing points (Hensman et al., 2013). The difference between the inducing points and the inducing patches is the dimension size. The dimensions of the inducing points are the same as the input data, whereas the dimensions of the inducing patches match the patch of the images.

We first define a group of $M$ inducing patches (Van der Wilk et al., 2017) $\mathbf{Z} = \{\mathbf{z}_m\}_{m=1}^M \in \mathbf{R}^{M\times E}$ for each latent function $u_q$. We then obtain $\mathbf{u}_q = \left[u_q(\mathbf{z}_1), \cdots, u_q(\mathbf{z}_M)\right]^\top$ evaluated at $\mathbf{Z}$. All latent functions $\mathcal{U} = \{u_q(\mathbf{x})\}_{q=1}^Q$ have different inducing patches and $\mathbf{u} = \left[\mathbf{u}_1^\top, \cdots, \mathbf{u}_Q^\top\right]^\top \in \mathbf{R}^{QM\times 1}$. Since the latent functions $\mathcal{U}$ are mutually independent, the distribution $p(\mathbf{u})$ factorises as $p(\mathbf{u}) = \prod_{q=1}^{Q} p(\mathbf{u}_q)$, with $\mathbf{u}_q \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_q)$, where $\mathbf{K}_q \in \mathbf{R}^{M\times M}$ has entries given by $k_q(\mathbf{z}_i, \mathbf{z}_j)$ with $\mathbf{z}_i, \mathbf{z}_j \in \mathbf{Z}$. The latent parameter functions $\mathbf{f}_d^c$ are conditionally independent given $\mathbf{u}$. We therefore obtain the conditional distribution $p(\mathbf{f}|\mathbf{u})$:

$$p(\mathbf{f}|\mathbf{u}) = \prod_{d=1}^{D}\prod_{c=1}^{C_d} p\left(\mathbf{f}_d^c|\mathbf{u}\right) \tag{22}$$

$$= \prod_{d=1}^{D}\prod_{c=1}^{C_d} \mathcal{N}\left(\mathbf{f}_d^c \middle| \mathbf{K}_{\mathbf{f}_d^c\mathbf{u}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{u}, \mathbf{K}_{\mathbf{f}_d^c\mathbf{f}_d^c} - \mathbf{K}_{\mathbf{f}_d^c\mathbf{u}}\mathbf{K}_{\mathbf{uu}}^{-1}\mathbf{K}_{\mathbf{f}_d^c\mathbf{u}}^\top\right), \tag{23}$$

where $\mathbf{K}_{\mathbf{uu}} \in \mathbf{R}^{QM\times QM}$ is a block-diagonal matrix based on $\mathbf{K}_q$ as each block.

Based on Moreno-Muñoz et al. (2018); Liu et al. (2019), we obtain the lower bound $\mathcal{L}$ for $\log p(\mathbf{y})$:

$$\log p(\mathbf{y}) = \log \int\int\int p(\mathbf{y}, \epsilon, \mathbf{f}, \mathbf{u}) d\mathbf{f} d\epsilon d\mathbf{u} \tag{24}$$

$$= \log \int\int\int \frac{p(\mathbf{y}, \epsilon, \mathbf{f}, \mathbf{u}) q(\mathbf{f}, \mathbf{u}, \epsilon)}{q(\mathbf{f}, \mathbf{u}, \epsilon)} d\mathbf{f} d\epsilon d\mathbf{u} \tag{25}$$

$$\geq \int\int\int q(\mathbf{f}, \mathbf{u}, \epsilon) \log \frac{p(\mathbf{y}, \epsilon, \mathbf{f}, \mathbf{u})}{q(\mathbf{f}, \mathbf{u}, \epsilon)} d\mathbf{f} d\epsilon d\mathbf{u} \tag{26}$$

$$= \int\int\int q(\epsilon|\mathbf{f}, \mathbf{u}) q(\mathbf{f}, \mathbf{u}) \log \frac{p(\mathbf{y}, \epsilon, \mathbf{f}, \mathbf{u})}{q(\epsilon|\mathbf{f}, \mathbf{u}) q(\mathbf{f}, \mathbf{u})} d\mathbf{f} d\epsilon d\mathbf{u} \tag{27}$$

$$= \mathcal{L}, \tag{28}$$

where

$$q(\mathbf{f}, \mathbf{u}) = p(\mathbf{f} \mid \mathbf{u}) q(\mathbf{u}) = \prod_{d=1}^{D}\prod_{c=1}^{C_d} p\left(\mathbf{f}_d^c \mid \mathbf{u}\right) \prod_{q=1}^{Q} q\left(\mathbf{u}_q\right),$$

where $q(\mathbf{u}_q) = \mathcal{N}\left(\mathbf{u}_q \mid \boldsymbol{\mu}_{\mathbf{u}_q}, \mathbf{S}_{\mathbf{u}_q}\right)$ and we refer $\left\{\boldsymbol{\mu}_{\mathbf{u}_q}, \mathbf{S}_{\mathbf{u}_q}\right\}_{q=1}^{Q}$ as the variational hyperparameters that need to be optimised. Further, we get (see the Appendix A for detail):

$$\mathcal{L} = \sum_{d}^{D}\sum_{i}^{N} \left\langle \log p\left(y_d(\mathbf{x}_i) | \widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right)\right\rangle_{q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)} \tag{29}$$

$$- \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q) \| p(\mathbf{u}_q)\right) - \sum_{i=d}^{D}\sum_{i=1}^{N} \mathrm{KL}\left(q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \| p\left(\epsilon_{d,i}\right)\right), \tag{30}$$

where $q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$ approximates the posterior $p\left(\epsilon_{d,i}|y_d(\mathbf{x}_i), \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$:

$$p\left(\epsilon_{d,i}|y_d(\mathbf{x}_i), \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \propto p\left(y_d(\mathbf{x}_i)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right)p\left(\epsilon_{d,i}\right), \tag{31}$$

$$p\left(y_d(\mathbf{x}_i)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right) = \prod_{c \neq y_d(\mathbf{x}_i)} \Phi_{\mathcal{G}}\left(\epsilon_{d,i} + f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) - f_d^c(\mathbf{x}_i)\right), \tag{32}$$

$$= \prod_{c \neq y_d(\mathbf{x}_i)} \exp\left(-e^{-\epsilon_{d,i} - f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) + f_d^c(\mathbf{x}_i)}\right). \tag{33}$$

$p(\epsilon_{d,i})$ is a probability density function of the standard Gumbel distribution. $q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$ approximates the marginal posterior for $\widetilde{\mathbf{f}}_d(\mathbf{x}_i)$:

$$q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) = \int p\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)|\mathbf{u}\right)q(\mathbf{u})d\mathbf{u} \tag{34}$$

$$= \mathcal{N}\Big(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)|\mathbf{K}_{\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\mathbf{u}}\mathbf{K}_{\mathbf{u}\,\mathbf{u}}^{-1}\boldsymbol{\mu}_{\mathbf{u}}, \tag{35}$$

$$\mathbf{K}_{\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\widetilde{\mathbf{f}}_d(\mathbf{x}_i)} + \mathbf{K}_{\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\mathbf{u}}\mathbf{K}_{\mathbf{u}\,\mathbf{u}}^{-1}\left(\mathbf{S}_{\mathbf{u}} - \mathbf{K}_{\mathbf{u}\,\mathbf{u}}\right)\mathbf{K}_{\mathbf{u}\,\mathbf{u}}^{-1}\mathbf{K}_{\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\mathbf{u}}^{\top}\Big), \tag{36}$$

where $\boldsymbol{\mu}_{\mathbf{u}} = \left[\boldsymbol{\mu}_{\mathbf{u}_1}^{\top}, \cdots, \boldsymbol{\mu}_{\mathbf{u}_Q}^{\top}\right]^{\top}$ and $\mathbf{S}_{\mathbf{u}}$ is a block diagonal matrix with blocks given as $\mathbf{S}_{\mathbf{u}_q}$. After calculation (NB: detail is in the Appendix A), we obtain:

$$\mathcal{L} = -\sum_{d=1}^{D}\sum_{i=1}^{N} \log\left(\mathcal{P}_{d,i} + 1\right) - \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q)\|p(\mathbf{u}_q)\right), \tag{37}$$

where

$$\mathcal{P}_{d,i} = \exp\left(\frac{v_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)}{2} - \mu_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)\right)\sum_{c \neq y_d(\mathbf{x}_i)} \exp\left(\frac{v_{f_d^c}(\mathbf{x}_i)}{2} + \mu_{f_d^c}(\mathbf{x}_i)\right), \tag{38}$$

where $\mu_{f_d^c}(\mathbf{x}_i)$ and $v_{f_d^c}(\mathbf{x}_i)$ are the mean and variance of $q\left(f_d^c(\mathbf{x}_i)\right)$, respectively.

### 3.3.2 Reducing computational complexity by subsampling

To reduce the computational complexity of our model, we use only a subset of the data observations and a subset of the classes to estimate the parameters and hyperparameters. The optimal parameters and hyperparameters are chosen by maximising an unbiased estimator of $\mathcal{L}$ (37), where the unbiased estimator is obtained through a subset of both training data points and classes in each output.

In our model, the hyperparameters are $\mathbf{Z}$, $\left\{a_{c,q}\right\}_{c=1,q=1}^{C,Q}$, $\{\phi_q\}_{q=1}^{Q}$, $\{w_p\}_{p=1}^{P}$ and the variational parameters are $\left\{\boldsymbol{\mu}_{\mathbf{u}_q}, \mathbf{S}_{\mathbf{u}_q}\right\}_{q=1}^{Q}$ for $\{q(\mathbf{u}_q)\}_{q=1}^{q=Q}$. We refer all those parameters as $\Theta$. To obtain the optimal values of the parameters $\Theta$, we use gradient descent to maximise $\mathcal{L}$ with respect to $\Theta$:

$$\nabla_\Theta \mathcal{L} = -\nabla_\Theta \sum_{d=1}^{D} \sum_{i=1}^{N} \log\left(\mathcal{P}_{d,i} + 1\right) - \nabla_\Theta \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q) \| p(\mathbf{u}_q)\right) \tag{39}$$

$$= -\nabla_\Theta \sum_{d=1}^{D} \sum_{i=1}^{N} \log\left(\left(\psi_{y_d(\mathbf{x}_i)} \sum_{c \neq y_d(\mathbf{x}_i)} \gamma_{d,c(\mathbf{x}_i)}\right) + 1\right) \tag{40}$$

$$- \nabla_\Theta \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q) \| p(\mathbf{u}_q)\right), \tag{41}$$

where, for notation simplicity, we define

$$\psi_{y_d(\mathbf{x}_i)} = \exp\left(\frac{v_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)}{2} - \mu_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)\right), \tag{42}$$

$$\gamma_{d,c(\mathbf{x}_i)} = \exp\left(\frac{v_{f_d^c}(\mathbf{x}_i)}{2} + \mu_{f_d^c}(\mathbf{x}_i)\right). \tag{43}$$

We then estimate $\nabla_\Theta \mathcal{L}$ by randomly sampling a subset of data points $\mathcal{B} = \{\mathbf{b}_1, ..., \mathbf{b}_{|\mathcal{B}|}\} \subseteq \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ of size $|\mathcal{B}|$ and a subset of classes $\mathcal{S} = \{s_1, ..., s_{|\mathcal{S}|}\} \subseteq \{1, \dots, C_d\} \backslash \{y_d(\mathbf{x})\}$ with size $|\mathcal{S}|$ ("\" means $\{y_d(\mathbf{x})\}$ is excluded from $\{1, \dots, C_d\}$) for each multi-class classification output:

$$\nabla_\Theta \widetilde{\mathcal{L}} = -\nabla_\Theta \sum_{d=1}^{D} \sum_{\mathbf{x}_i \in \mathcal{B}} \frac{N}{|\mathcal{B}|} \log\left(\frac{C_d - 1}{|\mathcal{S}|}\left(\psi_{y_d(\mathbf{x}_i)} \sum_{c \in \mathcal{S}} \gamma_{d,c(\mathbf{x}_i)}\right) + 1\right) \tag{44}$$

$$- \nabla_\Theta \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q) \| p(\mathbf{u}_q)\right). \tag{45}$$

$\nabla_\Theta \widetilde{\mathcal{L}}$ is an unbiased estimator for $\nabla_\Theta \mathcal{L}$ where the computational complexity of MOGPs-AR is dominated by optimising the parameters through maximising $\mathcal{L}$.

Our sampling strategy is in Algorithm 1. The computational complexity of MOGPs-AR mainly depends on the inversion of $\mathbf{K}_{\mathbf{u}\,\mathbf{u}}$ with complexity $\mathcal{O}(QM^3)$ and products like $\mathbf{K}_{\tilde{\mathbf{f}}\mathbf{u}}$ with complexity $\mathcal{O}\left(D|\mathcal{S}||\mathcal{B}|QM^2\right)$; If we do not use the subsampling of classes, we have to calculate products like $\mathbf{K}_{\bar{\mathbf{f}}\mathbf{u}}$ with a cost of $\mathcal{O}\left(C|\mathcal{B}|QM^2\right)$, where the notations are defined as below:

$$\widetilde{\mathbf{f}} = \left[\widetilde{\mathbf{f}}_{1,\mathcal{B}}^{\top}, \cdots, \widetilde{\mathbf{f}}_{D,\mathcal{B}}^{\top}\right]^{\top} \in \mathbf{R}^{D|\mathcal{S}||\mathcal{B}|\times 1} \tag{46}$$

$$\bar{\mathbf{f}} = \left[\bar{\mathbf{f}}_{1,\mathcal{B}}^{\top}, \cdots, \bar{\mathbf{f}}_{D,\mathcal{B}}^{\top}\right]^{\top} \in \mathbf{R}^{C|\mathcal{B}|\times 1} \tag{47}$$

$$\widetilde{\mathbf{f}}_{d,\mathcal{B}} = \left[\mathbf{f}_{d,\mathcal{B}}^{\mathcal{S}_1}, \cdots, \mathbf{f}_{d,\mathcal{B}}^{\mathcal{S}_{|\mathcal{S}|}}\right]^{\top} \in \mathbf{R}^{|\mathcal{S}||\mathcal{B}|\times 1} \tag{48}$$

$$\bar{\mathbf{f}}_{d,\mathcal{B}} = \left[\mathbf{f}_{d,\mathcal{B}}^{1}, \cdots, \mathbf{f}_{d,\mathcal{B}}^{C_d}\right]^{\top} \in \mathbf{R}^{C_d|\mathcal{B}|\times 1} \tag{49}$$

$$\mathbf{f}_{d,\mathcal{B}}^{c} = \left[f_d^c(\mathbf{b}_1), \cdots, f_d^c(\mathbf{b}_{|\mathcal{B}|})\right]^{\top} \in \mathbf{R}^{|\mathcal{B}|\times 1} \tag{50}$$

We notice $D|\mathcal{S}| \ll C$ ($C = \sum_{d=1}^{D} C_d$) so MOGPs-AR alleviates the computational complexity of the product $\mathbf{K}_{\widetilde{\mathbf{f}}\mathbf{u}}$ from $\mathcal{O}\left(C|\mathcal{B}|QM^2\right)$ to $\mathcal{O}\left(D|\mathcal{S}||\mathcal{B}|QM^2\right)$.

**Algorithm 1** Sampling strategy

```
1: Input: data $(\mathbf{X}, \mathbf{y})$, minibatch sizes $|\mathcal{B}|$ and $|\mathcal{S}|$
2: Output: parameters $\Theta$
3: Initialise all parameters and hyperparameters
4: for iteration $t = 1, 2, 3,...,$until convergence do
5:     # Sampling minibatches
6:     Sample a minibatch of data, $\mathcal{B} \subseteq \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
7:     for $i \in \mathcal{B}$ do
8:         # Sampling Classes
9:         Sample a set of classes, $\mathcal{S}_{d,i} \subseteq \{1, \ldots, C_d\} \setminus \{y_d(\mathbf{x}_i)\}$, $d \in \{1, ..., D\}$
10:        Compute $q\left(f_d^c(\mathbf{x}_i)\right)$ , where $c \in \mathcal{S}_{d,i} \cup \{y_d(\mathbf{x}_i)\}$, $d \in \{1, ..., D\}$ and $|\mathcal{S}| = |\mathcal{S}_{d,i}| + 1$
11:    Gradient step on parameters: $\Theta \leftarrow \Theta + \alpha \nabla_{\Theta} \widetilde{\mathcal{L}}$
```

## 3.4 Prediction

In this subsection, we derive the predictive distribution of MOGPs-AR. Considering a new test input $\mathbf{x}_*$ in the $d$-th output, we assume $p(\mathbf{u}|\mathbf{y}) \approx q(\mathbf{u})$ and approximate the predictive distribution $p\left(y_d(\mathbf{x}_*)\right)$ by

$$p\left(y_d(\mathbf{x}_*)|\mathbf{y}\right) \approx \int p\left(y_d(\mathbf{x}_*) \mid \widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right) q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right) d\widetilde{\mathbf{f}}_d(\mathbf{x}_*), \tag{51}$$

where $q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right) = \int p\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_*) \mid \mathbf{u}\right) q(\mathbf{u}) d\mathbf{u} = \prod_{c=1}^{C_d} q\left(f_d^c(\mathbf{x}_*)\right)$. The approximated latent parameter functions $q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right)$ are mutually independent, so we obtain

$$p\left(y_d(\mathbf{x}_*)|\mathbf{y}\right) \approx \int p\left(y_d(\mathbf{x}_*) \mid \widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right) q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_*)\right) d\widetilde{\mathbf{f}}_d(\mathbf{x}_*) \tag{52}$$

$$= \int \frac{\exp\left(f_d^{y_d(\mathbf{x}_*)}(\mathbf{x}_*)\right)}{\sum_{c=1}^{C_d} \exp\left(f_d^c(\mathbf{x}_*)\right)} \prod_{c=1}^{C_d} \mathcal{N}\left(f_d^c(\mathbf{x}_*) \mid \mu_{f_d}^c(\mathbf{x}_*), v_{f_d}^c(\mathbf{x}_*)\right) d\widetilde{\mathbf{f}}_d(\mathbf{x}_*). \tag{53}$$

We can use Monte Carlo to approximate the integral in the same way as Liu et al. (2019).

## 4 Experiments

In this section, we evaluate MOGPs-AR in various data sets. We apply MOGPs-AR in a synthetic data set to show its scalability in the number of classes compared to multi-output Gaussian processes. We also compare MOGPs-AR to other models in different real data sets. Further, to test MOGPs-AR the capacity in dealing with an image data set, we compare the performance of a convolutional kernel and RBF-ARD in our model.

*Baselines.* We compare the MOGPs-AR with the following two single-output and one multi-output Gaussian process models: 1) A Gaussian process for multi-class classification model (**G-M**), an independent Gaussian process using the softmax likelihood. 2) A Gaussian process multi-class classification with additive noise model (**G-A**), an independent Gaussian process using the softmax likelihood via Gumbel noise. 3) A multi-output Gaussian process model for multi-class classification problems (**MG-M**), a standard linear model of coregionalisation for MOGPs using the softmax likelihood. For all the different models in this paper, we use RBF-ARD, unless otherwise stated. For all models, we use traditional inducing points (Hensman et al., 2013) unless mentioned otherwise. All models are trained using the Adam optimiser with 0.01 learning rate and trained by 4000 iterations (Kingma & Ba, 2014). We use the same 80% training and 20% validation data set to choose the optimal number $Q$ of latent functions $\mathcal{U}$, where we optimise again all hyperparameters during cross-validation, for MOGPs-AR and MG-M.

*Evaluation Metrics.* There are three different evaluation metrics in this paper:

$$\text{Precision-Weighted} = \frac{1}{\sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right|} \sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right| P\left(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}\right), \tag{54}$$

$$\text{Recall-Weighted} = \frac{1}{\sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right|} \sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right| R\left(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}\right), \tag{55}$$

$$\text{F1-Weighted} = \frac{1}{\sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right|} \sum_{l\in\mathbb{L}} \left|\mathbb{O}^l_{true}\right| F_1\left(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}\right), \tag{56}$$

$$P(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) = \frac{|\mathbb{O}^l_{prediction} \cap \mathbb{O}^l_{true}|}{|\mathbb{O}^l_{prediction}|}, \tag{57}$$

$$R(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) = \frac{|\mathbb{O}^l_{prediction} \cap \mathbb{O}^l_{true}|}{|\mathbb{O}^l_{true}|}, \tag{58}$$

$$F_1(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) = 2\frac{P(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) \times R(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true})}{P(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) + R(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true})}, \tag{59}$$

where $\mathbb{O}_{true}$ and $\mathbb{O}_{prediction}$ are sets of true and predicted pairs (input data point, class) separately (e.g., $\mathbb{O}_{true,n} = \left(\mathbf{x}_n, y_{true,n}\right)$ where $\mathbf{x}_n$ is the $n$-th input data point and $y_{true,n}$ is the

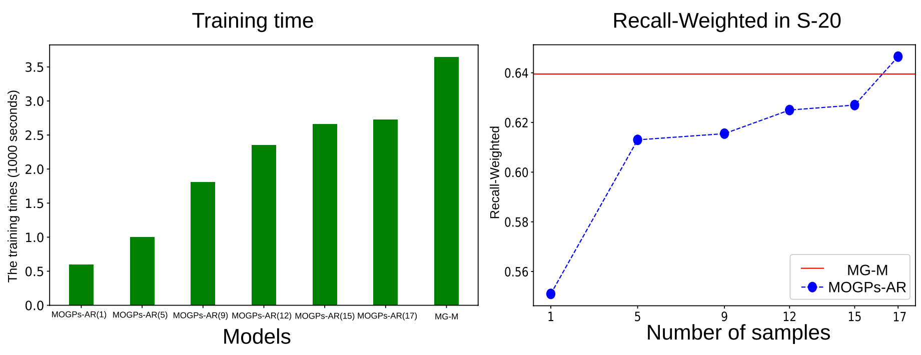


**Fig. 2** Left: The training time in MG-M and MOGPs-AR model in S-20 (MOGPs-AR(5) means that MOGPs-AR with a subset of classes $\mathcal{S}_d \subseteq \{1, \dots, C_d\} \backslash \{y_d(\mathbf{x})\}$ with size $|\mathcal{S}_d| = 5$ ($|\mathcal{S}| = |\mathcal{S}_d| + 1$) and we use $y_d(\mathbf{x})$ and $C_d$ for notation consistency but here is only one output); Right: The Recall weight between MG-M and MOGPs-AR with different number of samples (e.g., 5 means $|\mathcal{S}_d| = 5$)

corresponding class for $\mathbf{x}_n$. The $\mathbb{O}^l_{true}$ and $\mathbb{O}^l_{prediction}$ are subsets of $\mathbb{O}_{true}$ and $\mathbb{O}_{prediction}$ separately (e.g., $\mathbb{O}^l_{true} = \{ \left(\mathbf{x}_n, y_{true,n}\right) \in \mathbb{O}_{true} \mid y_{true,n} = l, n \in \mathbb{N}\}$). The $\mathbb{L}$ and $\mathbb{N}$ are the sets of classes and input data points, respectively. The formulas use $P(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) = 0$ or $R(\mathbb{O}^l_{prediction}, \mathbb{O}^l_{true}) = 0$ if $\mathbb{O}^l_{prediction} = \emptyset$ or $\mathbb{O}^l_{true} = \emptyset$.

The synthetic data experiment was performed on a Dell PowerEdge C6320 with an Intel Xeon E5-2630 v3 at 2.40 GHz and 64GB of RAM. All real data experiments were performed on a PowerEdge R740XD Server with NVIDIA Tesla v100 32GB GDDR.[1]

## 4.1 Synthetic data

In this subsection, we compare the performance of MOGPs-AR with MG-M on synthetic data where we generate a single output classification synthetic data set.[2] We create a 20 class data set by assigning a cluster of 100 points normally distributed, where each data point has five features, to each class. In total, there are 2000 samples. Since the synthetic data has 20 classes, we refer to it as S-20. We use 20 classes to compare MOGPs-AR with MG-M in terms of scalability. MOGPs-AR and MG-M use the same parameter setting (see Table 3) exclude that MOGPs-AR used a different number of subset classes.

We compare MOGPs-AR with MG-M in terms of training time and Recall-Weighted performance. Figure 2 shows the mean training time for MOGPs-AR is less than MG-M in five folds cross-validation. This is because the computational complexity of MOGPs-AR is less than MG-M. As we mentioned in 3.3.2, compared to MG-M, MOGPs-AR reduce the computational complexity of the product $\mathbf{K}_{\tilde{\mathbf{f}}\mathbf{u}}$ from $\mathcal{O}\left(C|\mathcal{B}|QM^2\right)$ to $\mathcal{O}\left(D|\mathcal{S}||\mathcal{B}|QM^2\right)$ where $D|\mathcal{S}| \ll C$. Figure 2 empirically shows the mean training time of MOGPs-AR (1) with 596s is nearly one-sixth of MG-M with 3641s. The mean training time in MOGPs-AR increases as the number of $|\mathcal{S}_d|$ increases but it is still less than MG-M. While MOGPs-AR has less training time than MG-M, it has a similar performance in Recall-Weighted with MG-M for S-20. Even if we use a small subset of classes, e.g., five classes, MOGPs-AR also has a close performance to MG-M (see Fig. 2 right panel). The Recall-Weighted of MOGPs-AR slightly increases as the number

[1] All codes are available online at https://github.com/ChunchaoPeter/MOGPs-AR.

[2] This data is generated from scikit-learn (Pedregosa et al., 2011)

of samples increase. Further, we notice that MOGPs-AR (17) has a better performance than MG-M. In theory, MOGPs-AR should have the same performance as MG-M. However, we can not perform convex optimisation for both MG-M and MOGPs-AR, so MOGPs-AR may outperform MG-M in various performance metrics in practice.

### 4.2 Single-output GP classification: four real data sets

We will use the following four real data sets to test the performance of the different GP classifiers: 1) **Balance** (Dua & Graff, 2017) is a data set for the results of psychology experiments. There are 625 data points with four discrete variables: Left-Weight, Left-Distance, Right-Weight and Right-Distance. The value of all four discrete variables ranges from one to five. The data set consists of three classes: the balance scale tipped to the right (R), tipped to the left (L) or be balanced (B). 2) **CANE9** (Dua & Graff, 2017) contains 1080 documents of free text business descriptions of Brazilian companies. Those documents are divided into nine different categories. Each document has 856 integer variables (word frequency). 3) **Mediamill** (Snoek et al., 2006) is a multi-label data set for generic video indexing. To apply multi-classification, we only maintained one label, which is the first label to appear, for each data point. Further, we only use part of this data set since the original data set is highly imbalanced. We then obtain the number of data points for each class ranged from 31 to 545. In total, we have 6689 data points with 120 numeric features and 35 classes. 4) **Bibtex** data set (Katakis et al., 2008) is also a multi-label data that contains 7395 Bibtex entries with 1836 variables. Similarly, we only maintained one label, which is the first label to appear, obtaining 148 classes.

In all three performance measures, MOGPs-AR outperforms G-A and G-M on all four data sets (see Figure. 3). This is because MOGPs-AR can use each of latent parameter functions $\mathbf{f}$, which is a linear combination of latent functions $\mathcal{U}$, to predict each class. The underlying function of the latent functions $\mathcal{U}$ and $\mathbf{B}_q$ can transfer knowledge between each class in the same output. However, G-A and G-M only have independent Gaussian processes that can not capture the similarity between each class. Further, Fig. 3 indicates that using a small subset of classes (e.g., MOGPs-AR(1) or MOGPs-AR(5)), MOGPs-AR obtains a similar result as MG-M for Balance, CANE9, Mediamill data sets as we have discussed as in 4.1.

Compared with single output Gaussian processes, MOGPs-AR can achieve around 10% improvement in terms of three performance metrics on Balance and CANE9 data set (Fig. 3 upper panel). The optimal number ($Q$) of latent functions $\mathcal{U}$ is two and nine for the Balance and CANE9 data sets separately. Those latent functions share the knowledge between each class and help to improve the performance. There is also a connection between single output and multi-output Gaussian processes. Considering an extreme case, we assume there is only one class, $Q$=1 and $\mathbf{B}_q = 1$, MOGPs-AR and MG-M have the same structure as G-A and G-M in theory, respectively.

Regarding both Mediallmill (35 classes) and Bibtex (148 classes), MOGPs-AR has excellent performance compared to the single-output Gaussian processes and MG-M. For the Mediamill dataset, based on capturing dependency between each class, MOGPs-AR is nearly 6 times better than G-A and 4 times better than G-M in terms of F1-Weighted, where the mean of F1-Weighted is 0.04 for G-A, 0.08 for G-M and 0.25 for MOGPs-AR. Further, we cannot apply MG-M in the Bibtex data set since it is not able to compute $\mathbf{K}_{\tilde{\mathbf{f}}\mathbf{u}}$ (out of memory). However, MOGPs-AR scales well since it only uses a subset of classes (MOGPs-AR (20)) for prediction.

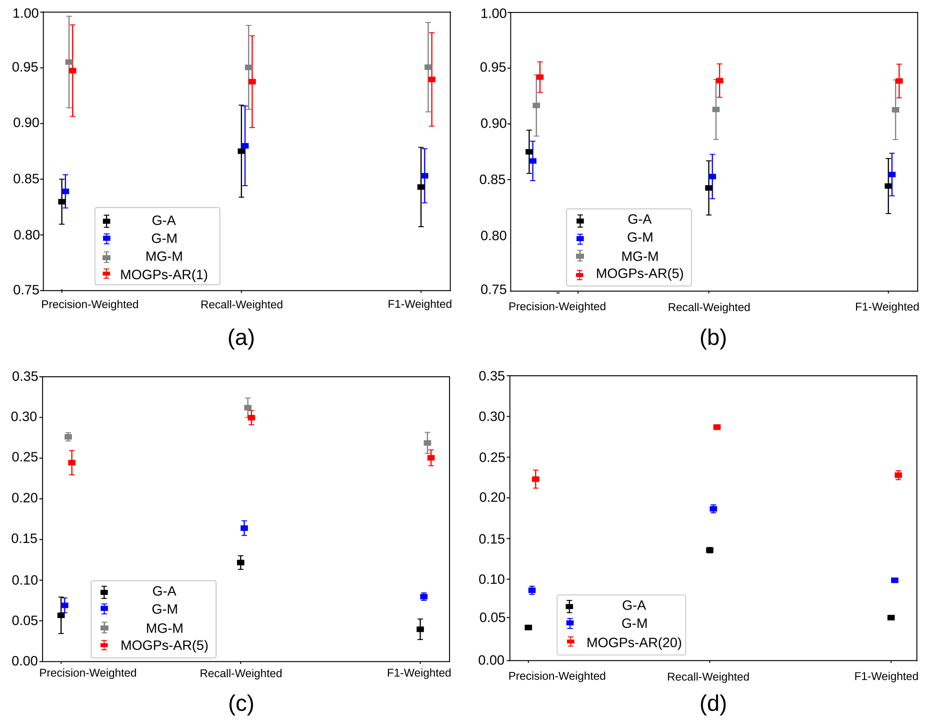


**Fig. 3** Performance in cross-validation (mean ± standard deviation) in Balance, CANE9 and Mediamill data sets. **a** Balance Data Results; **b** CANE9 Data Results; **c** Mediamill Data Results; **d** Bibtex Data Results

### 4.3 Multi-output GPs classifications: UJIIndoorLoc

To compare the performance of MOGPs-AR in multi-output multi-class classification problems, we apply MOGPs-AR to UJIIndoorLoc Data Set (Torres-Sospedra et al., 2014). There are 21048 instances that rely on WIFI fingerprint for three buildings of Universitat Jaume I where Building I and Building II have four floors respectively and Building III has five floors. Each instance has 520 features based on signal strength intensity. We randomly sample 200 data points from each floor so there are 800 data points for Building I and Building II respectively and 1000 data points for Building III. Further, we standardise the data set for each Building. We make predictions for each floor depending on the 520 features. Since there are three buildings of Universitat Jaume I, we assume there is a strong correlation between each building. We regard each building as each output and different floors as different classes in our model. The UJIIndoorLoc is considered as a multi-output multi-class classification problem. In this experiment and following, we do not apply the MG-M model due to its computational complexity. MOGPs-AR can be an alternative model for MG-M so we only consider MOGPs-AR and two single output GP models.

Figure 4 shows that MOGPs-AR outperforms single-output Gaussian processes in both Building I, II and III in all three performance measures. For example, MOGPs-AR can achieve around 50% improvement in terms of Recall-Weighted on Building I compared with single output Gaussian processes. The reason is that MOGPs-AR can capture dependencies of intra- and inter- three buildings. The dependencies can help improve the prediction for all buildings.

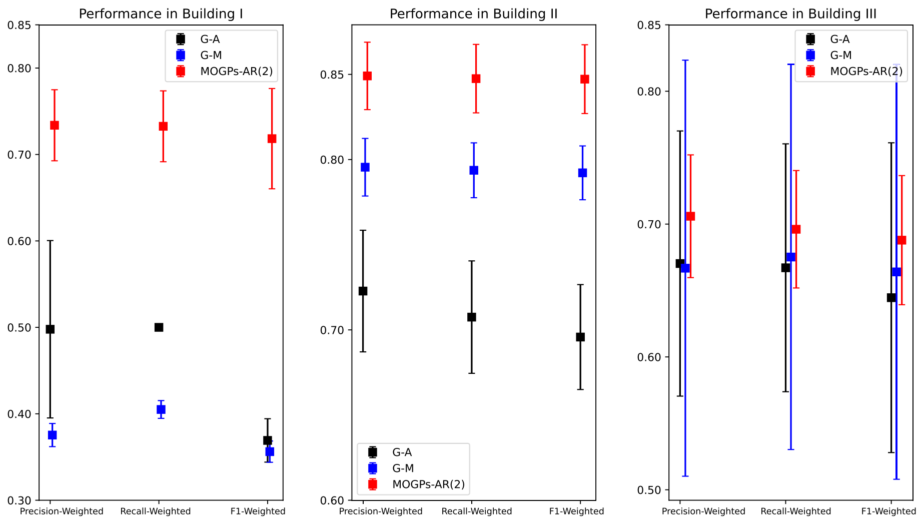


**Fig. 4** Performance in cross-validation (mean ± standard deviation)

The single-output Gaussian processes cannot use the dependency so the single output Gaussian process does not perform well in UJIIndoorLoc.

To investigate the correlation between intra- and inter output, we create a *global absolute coregionalisation matrix*. First, we create absolute coregionalisation matrices $\left\{\mathbf{B}_q^{abs}\right\}_{q=1}^{Q}$, where $\mathbf{B}_q^{abs} \in \mathbf{R}^{C\times C}$, by taking the absolute value of each entry in $\mathbf{B}_q$. Second, we obtain the mean of those absolute coregionalisation matrices: $\overline{\boldsymbol{B}} = \frac{1}{Q}\sum_{q=1}^{Q}\mathbf{B}_q^{abs}$ and $\overline{\boldsymbol{B}} \in \mathbf{R}^{C\times C}$. Since we are performing $K$-fold cross-validation, we have $K$ different mean absolute coregionalisation matrices: $\left\{\overline{\boldsymbol{B}}^i\right\}_{i=1}^{K}$, where $\overline{\boldsymbol{B}}^i \in \mathbf{R}^{C\times C}$ refers to the mean absolute coregionalisation matrices during the $i$-th fold cross-validation. Further, we calculate the mean of $\left\{\overline{\boldsymbol{B}}^i\right\}_{i=1}^{K}$ for all $K$-fold cross-validation so $\tilde{\mathbf{B}} \in \mathbf{R}^{C\times C} = \frac{1}{K}\sum_{i=1}^{K}\overline{\boldsymbol{B}}^i$:

$$\tilde{\mathbf{B}} = \begin{bmatrix} \left(\tilde{\mathbf{B}}\right)_{1,1} & \cdots & \left(\tilde{\mathbf{B}}\right)_{1,D} \\ \left(\tilde{\mathbf{B}}\right)_{2,1} & \cdots & \left(\tilde{\mathbf{B}}\right)_{2,D} \\ \vdots & \ddots & \vdots \\ \left(\tilde{\mathbf{B}}\right)_{D,1} & \cdots & \left(\tilde{\mathbf{B}}\right)_{D,D} \end{bmatrix}, \tag{60}$$

where $\left(\tilde{\mathbf{B}}\right)_{i,j} \in \mathbf{R}^{C_i\times C_j}$ indicates the correlations for all latent parameter functions between $i$-th output and $j$-th output. At the end, in order to find the correlation for outputs independently, we calculate a scalar $\tilde{B}_{i,j} = \frac{1}{C_iC_j}\sum_m\sum_n\left\{\left(\tilde{\mathbf{B}}\right)_{i,j}\right\}_{m,n}$, which represents dependence between $i$-th output and $j$ output. We therefore define a global absolute coregionalisation matrix (GACM $\in \mathbf{R}^{D\times D}$) as the following:

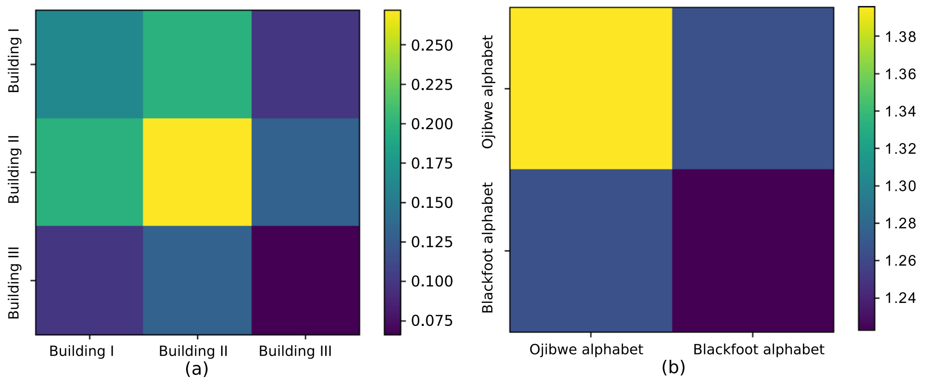


**Fig. 5** Global absolute coregionalisation matrix. **a** The global absolute coregionalisation matrix of UJIIndoorLoc data set. **b** The global absolute coregionalisation matrix of Ojibwe and Blackfoot alphabets

$$\mathrm{GACM} = \begin{bmatrix} \tilde{B}_{1,1} & \dots & \tilde{B}_{1,D} \\ \tilde{B}_{2,1} & \dots & \tilde{B}_{2,D} \\ \vdots & \ddots & \vdots \\ \tilde{B}_{D,1} & \dots & \tilde{B}_{D,D} \end{bmatrix}. \tag{61}$$

Figure 5a shows the correlation between each building captured by our model. We can notice there is a strong correlation between the different buildings. Building I and Building II have a relatively strong correlation compared to Building I and Building III, Building II and Building III. Building II has the strongest intra-output correlation while Building III has the smallest intra-output correlation among those three buildings.

### 4.4 Multi-output GPs classifications: Omniglot-dataset

We apply MOGPs-AR to Omniglot image data set (Lake et al., 2015). The Omniglot data set includes 1623 various handwritten characters from 50 distinct alphabets. Each of the 1623 characters was drawn by 20 different people (the total number of images is 32460). Although traditional MOGPs are not specifically designed to deal with image data, MOGPs-AR can handle image data by incorporating a convolutional kernel (Van der Wilk et al., 2017). The size of each image is $105 \times 105$ pixels. To help speeding up the computation and reducing the computational complexity in the covolutional kernel, we resize the images from $105 \times 105$ to $20 \times 20$ as Santoro et al. (2016) did. We regard each alphabet as an output in our model. Each alphabet has different characters which are considered as different classes. Therefore, we consider the Omniglot data set as multi-output multi-class classification problems.

#### 4.4.1 Ojibwe and blackfoot alphabets

To compare the performance of MOGPs-AR in multi-output multi-class classification problems and image input data, we first consider Ojibwe and Blackfoot alphabets as two different multi-class classification problems (see Fig. 6). Since the two alphabets are from Canadian Aboriginal Syllabics, we assume there is a strong correlation between them. Our

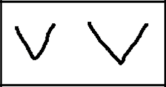
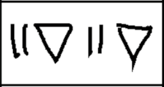
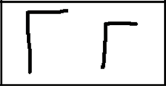
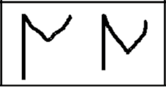
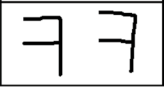
| Character | One | Two | Three | ··· | Fourteen |
|---|---|---|---|---|---|
| Ojibwe alphabet | |  | | ··· |  |
| Blackfoot alphabet | |  |  | ··· |  |

**Fig. 6** Both the Ojibwe and the Blackfoot alphabets have 14 characters each. We show two data points for each character

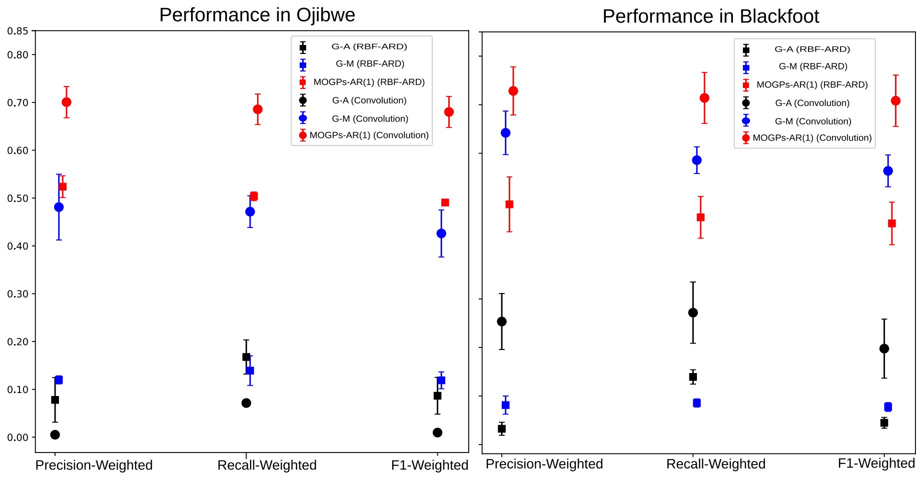


**Fig. 7** Image: Performance in cross-validation (mean ± standard deviation). We compare both **RBF-ARD** and **the convolutional kernel** for all the model

model can capture the correlation through joint modelling of the two alphabets to improve predictive performance for each multi-class classification problem. There are 14 different characters in each output so there are 14 classes, and each class has 20 data points. We compare both the RBF-ARD kernel and the convolutional kernel. Table 4 shows the parameter setting in Omniglot data set.

In Fig. 7 we show that MOGPs-AR outperforms single-output Gaussian processes in both alphabets in terms of the convolutional kernel or RBF-ARD. The reason is that MOGPs-AR can capture the dependency between the two alphabets. The dependency can help improve the prediction for both alphabets. The single output Gaussian processes cannot use the dependency so the single output Gaussian process with either the convolutional kernel or RBF-ARD does not perform well in both Ojibwe and Blackfoot. The size of the mini-batch is too small that has also a negative influence on the single output Gaussian processes (Fig. 8). Especially, the values of the three performance metrics are closed to 0.05 for G-A with the convolutional kernel on Ojibwe.

Since both alphabets are from Canadian Aboriginal Syllabic we expect they have a strong correlation. Figure 5b indeed shows there is a similar global correlation between intra- and inter- output for both alphabets, which indicates that our model has the capacity of capturing the underline correlation among those correlated data sets.

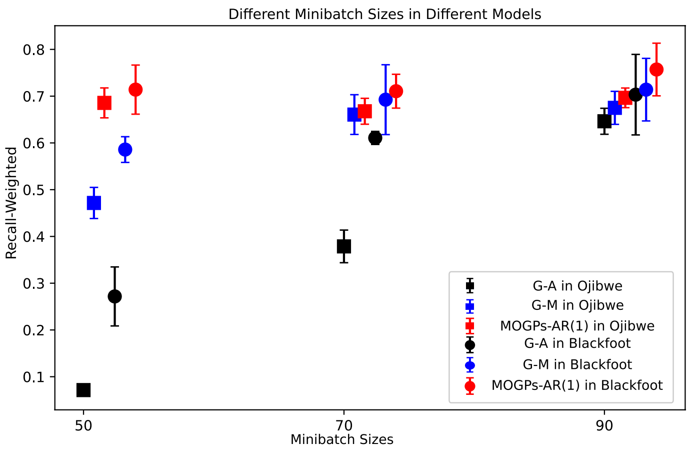


**Fig. 8** **Recall-Weighted** Performance in cross-validation (mean ± standard deviation)

MOGPs-AR with the convolutional kernel outperforms MOGPs-AR with RBF-ARD in both alphabets in terms of three performance metrics (see Fig. 7). For example, MOGPs-AR improves the Recall-Weighted from 0.468 to 0.714 by changing RBF-ARD to the convolutional kernel on the Blackfoot alphabet. Moreover, we also combine G-M and G-A with the convolutional kernel and they also have stronger performance compared with RBF-ARD. In particular, G-M with the convolutional kernel obtains 0.5858 compared with 0.0857 using RBF-ARD in terms of Recall-Weighted on the Blackfoot alphabet. The performance of G-M with the convolutional kernel (0.5858) is better than MOGPs-AR with RBF-ARD (0.468) on the Blackfoot alphabet. The reason is that the convolutional kernel is more effectively capturing image-level features than the RBF-ARD kernel.

To investigate the effects of mini-batch size, we set up another experiment. We train again the exact same models with the parameters initialised in the same way as the experiment above but using different mini-batch sizes (e.g., 50, 70, 90). Since the convolution kernel provided better results in the previous experiments, we only show the results using the convolution kernel and the Recall-Weighted performance measure for both alphabets. Figure 8 shows that the size of the mini-batch has more influence on single output Gaussian processes than MOGPs-AR. A small size number for the mini-batch, e.g., 50, has a negative impact on G-M and G-A. However, MOGPs-AR has a slight increase in performance or keeps a similar result with the mini-batch size increasing. G-A and G-M improve the performance as the mini-batch size grows up from 50 to 90. When the size of the mini-batch is 90, G-M has a similar performance with MOGPs-AR. However, when we consider the mini-batch of size 50, MOGPs-AR still can get good performance compared to single GPs.

### 4.4.2 All alphabets

In our final experiment, we apply MOGPs-AR in 50 alphabets in the original data set. There are 50 outputs with different number of classes in each output (for more details on the number of classes in each output see Table 2). The total number of classes in the 50 outputs are 1623. We follow Lake et al. (2015) and split the 50 alphabets into two sets: a background set and an evaluation set, where the background set has 30 alphabets (with a

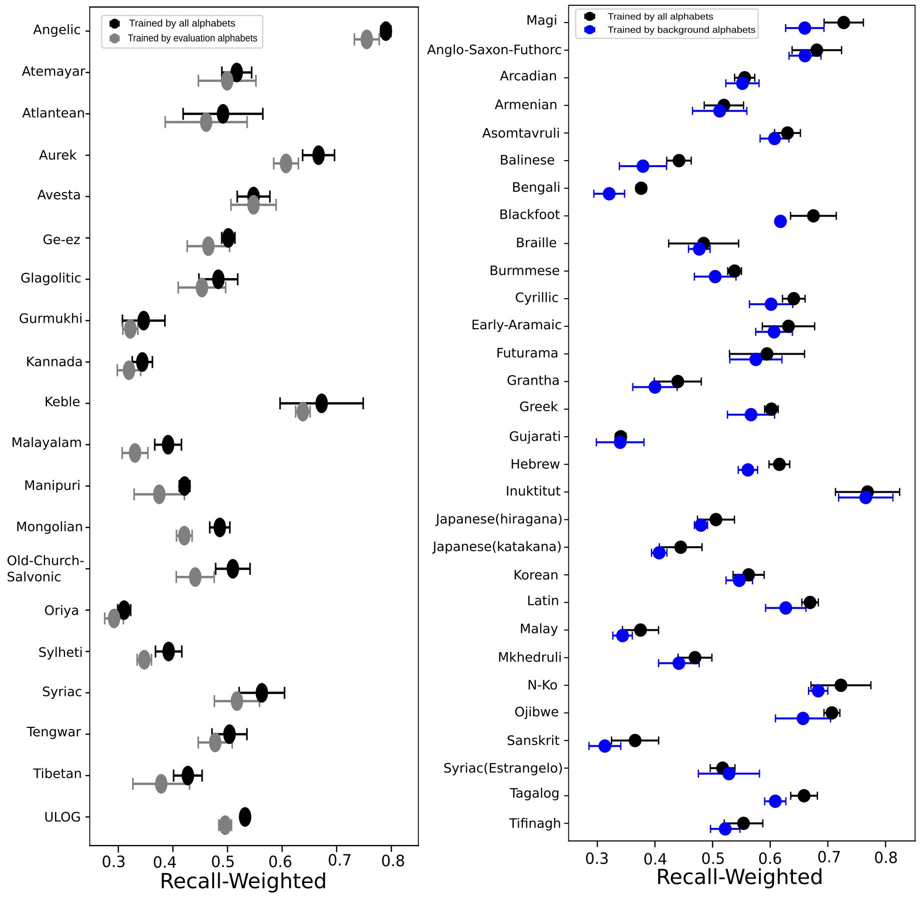


**Fig. 9** Performance in cross-validation in Evaluation alphabets (Left) and Background alphabets (Right). In the both Figures, each circle is the mean of Recall-Weighted; the error bar is the standard deviation of Recall-Weighted

total of 964 classes) and the evaluation set has 20 alphabets (with a total of 659 classes). In order to apply MOGPs-AR in all 50 alphabets, we use a mini-batch size of nine data points for each output to train our model. The small mini-batch size has a negative impact on G-M and G-A so we only apply MOGPs-AR in this experiments. We apply MOGPs-AR for three different sets of alphabets: all alphabets, the Background alphabets and the Evaluation alphabets.

In Fig. 9, we empirically (50 different outputs and a total of 1623 classes of image data) show that MOGPs-AR has better scalability than traditional multi-output Gaussian processes. MOGPs-AR reduces the computational complexity by subsampling both training data points and classes in each output. Figure 9 also indicates that MOGPs-AR obtains good performance even if we choose a small size of mini-batch (nine) and only a small number of classes (one) in each output since it captures both intra- and inter-output correlation.

In most predictions, our model trained with the data of all alphabets could outperform one trained with the data of part of the alphabets. For example, our model trained using all alphabets improves the Recall-Weighted from 0.6096 to 0.6692 for the Aurek alphabet, compared with one using evaluation alphabets for training. The extra alphabets can help our model improve its performance.

However, there are exceptions to the scenario in the last paragraph. For example, for the Syriac (Estrangelo) alphabet, the values of the Recall-Weighted 0.5174 is less than 0.5283 where only use background alphabets for training our model. One likely reason is that our model assumes a correlation with all alphabets. However, the correlation with those alphabets may not exist or the correlation may hinder the predictive performance.

## 5 Conclusion

In this paper, we have introduced MOGPs-AR, a novel framework that allows the use of multi-output Gaussian processes for multi-output multi-class classification. MOGPs-AR can tackle large scale data sets and a large number of classes in each output. Further, when combined with the convolutional kernel, it is suited for downsized image data.

We experimentally show that MOGPs-AR has a similar result to MG-M that is a linear model of coregionalization and uses a similar stochastic variational inference method as us. However, the training time of MOGPs-AR is less than MG-M. Experimental results in various data sets also indicate that MOGPs-AR significantly improves the performance compared to single output Gaussian processes.

MOGPs-AR has good performance in extreme classification using a softmax function which is only suited to each instance associated with a single class. Because of the softmax function, MOGPs-AR can not deal with a multi-label problem where each data point belongs to multiple classes. It would be an interesting work for future research if we can generalise MOGPs-AR for the multi-label problem that has a strong correlation with extreme classification problems.

A practical application of Gaussian process models to realistic image recognition tasks is still an open research problem. For example, in terms of accuracy performance in a realistic RGB set CIFAR-10 (Krizhevsky & Hinton, 2009), the accuracy performance of Gaussian processes (Van der Wilk et al., 2017; Blomqvist et al., 2019) is not as high as the state-of-the-art like deep learning. A potential extension would be to consider integrating the structural properties of deep learning architectures into our model by using deep kernel learning (Wilson et al., 2016).

## Appendix A Complete derivation of the lower bound $\mathcal{L}$

To compute the derivation of the lower bound $\mathcal{L}$, we begin with the following:

$$\mathcal{L} = \left\langle \log \frac{p(\mathbf{y}, \mathbf{f}, \mathbf{u}, \epsilon)}{q(\mathbf{f}, \mathbf{u}, \epsilon)} \right\rangle_{q(\mathbf{f},\mathbf{u},\epsilon)} \tag{62}$$

$$= \int\int\int q(\epsilon|\mathbf{f})p(\mathbf{f}|\mathbf{u})q(\mathbf{u})\log\frac{p(\mathbf{y}|\mathbf{f}, \mathbf{u}, \epsilon)p(\mathbf{u})p(\epsilon)}{q(\epsilon|\mathbf{f})q(\mathbf{u})} d\mathbf{f}d\epsilon d\mathbf{u} \tag{63}$$

$$= \int\int\int \prod_{d=1}^{D}\prod_{i=1}^{N} q(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)) \prod_{c=1}^{C_d} p(\mathbf{f}_d^c|\mathbf{u}) \prod_{q=1}^{Q} q(\mathbf{u}_q) \tag{64}$$

$$\times \log \frac{\prod_{d=1}^{D}\prod_{i=1}^{N} p\left(y_d(\mathbf{x}_i) \mid \widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right) \prod_{q=1}^{Q} p(\mathbf{u}_q) \prod_{d=1}^{D}\prod_{i=1}^{N} p(\epsilon_{d,i})}{\prod_{d=1}^{D}\prod_{i=1}^{N} q(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)) \prod_{q=1}^{Q} q(\mathbf{u}_q)} \tag{65}$$

$$d\mathbf{f}d\epsilon d\mathbf{u} \tag{66}$$

$$= \int\int\int \prod_{d=1}^{D}\prod_{i=1}^{N} q(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)) \prod_{c=1}^{C_d} p(\mathbf{f}_d^c|\mathbf{u}) \prod_{q=1}^{Q} q(\mathbf{u}_q) \tag{67}$$

$$\times \log \prod_{d=1}^{D}\prod_{i=1}^{N} p\left(y_d(\mathbf{x}_i) \mid \widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right) d\mathbf{f}d\epsilon d\mathbf{u} \tag{68}$$

$$- \sum_{q=1}^{Q} \mathrm{KL}\left(q(\mathbf{u}_q) \| p(\mathbf{u}_q)\right) - \sum_{i=d}^{D}\sum_{i=1}^{N} \mathrm{KL}\left(q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \| p(\epsilon_{d,i})\right), \tag{69}$$

where $q(\mathbf{f}, \mathbf{u}, \epsilon) = p(\mathbf{f}|\mathbf{u})q(\mathbf{u})q(\epsilon|\mathbf{f})$. We assume $q(\mathbf{f}) \approx p(\mathbf{f} \mid \mathbf{y})$ so we obtain

$$q(\mathbf{f}) \approx p(\mathbf{f} \mid \mathbf{y}) \tag{70}$$

$$= \int p(\mathbf{f} \mid \mathbf{u}) q(\mathbf{u}) d\mathbf{u} \tag{71}$$

$$= \int \prod_{d=1}^{D}\prod_{c=1}^{C_d} p\left(\mathbf{f}_d^c \mid \mathbf{u}\right) \prod_{q=1}^{Q} q(\mathbf{u}_q) d\mathbf{u} \tag{72}$$

$$= \prod_{d=1}^{D}\prod_{c=1}^{C_d} \mathbf{E}_{q(\mathbf{u})}\left\{p\left(\mathbf{f}_d^c \mid \{\mathbf{u}_q\}_{q=1}^{Q}\right)\right\} \tag{73}$$

$$= \prod_{d=1}^{D} q\left(\widetilde{\mathbf{f}}_d\right) = \prod_{d=1}^{D}\prod_{c=1}^{C_d} q(\mathbf{f}_d^c) = \prod_{i=1}^{N}\prod_{d=1}^{D}\prod_{c=1}^{C_d} q(f_d^c(\mathbf{x}_i)) \tag{74}$$

The above function means the latent parameter functions are mutually independent in $q(\mathbf{f})$. Then, we obtain:

$$\mathcal{L} = \int\int \prod_{d=1}^{D}\prod_{i=1}^{N} q(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)) q(\mathbf{f}) \log \prod_{d=1}^{D}\prod_{i=1}^{N} p\left(y_d(\mathbf{x}_i) \mid \widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right) d\mathbf{f}d\epsilon \tag{75}$$

$$-\sum_{q=1}^{Q}\mathrm{KL}\left(q\left(\mathbf{u}_q\right)\|p\left(\mathbf{u}_q\right)\right)-\sum_{i=d}^{D}\sum_{i=1}^{N}\mathrm{KL}\left(q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)\|p\left(\epsilon_{d,i}\right)\right) \tag{76}$$

$$=\sum_{d}^{D}\sum_{i}^{N}\left\langle\log p\left(y_d\left(\mathbf{x}_i\right)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i),\epsilon_{d,i}\right)\right\rangle_{q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)} \tag{77}$$

$$-\sum_{q=1}^{Q}\mathrm{KL}\left(q\left(\mathbf{u}_q\right)\|p\left(\mathbf{u}_q\right)\right)-\sum_{i=d}^{D}\sum_{i=1}^{N}\mathrm{KL}\left(q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)\|p\left(\epsilon_{d,i}\right)\right) \tag{78}$$

The $q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$ approximates the posterior $p\left(\epsilon_{d,i}|y_d\left(\mathbf{x}_i\right),\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$ (similar to Liu et al. (2019)):

$$p\left(\epsilon_{d,i}|y_d\left(\mathbf{x}_i\right),\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)\propto p\left(y_d\left(\mathbf{x}_i\right)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i),\epsilon_{d,i}\right)p\left(\epsilon_{d,i}\right) \tag{79}$$

$$=\phi_{\mathcal{G}}\left(\epsilon_{d,i}\right)\prod_{c\neq y_d(\mathbf{x}_i)}\Phi_{\mathcal{G}}\left(\epsilon_{d,i}+f_d^{y_d(\mathbf{x}_i)}\left(\mathbf{x}_i\right)-f_d^{c}\left(\mathbf{x}_i\right)\right) \tag{80}$$

$$=\exp\left(-\epsilon_{d,i}-\left(1+\sum_{c\neq y_d(\mathbf{x}_i)}e^{f_d^c(\mathbf{x}_i)-f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)}\right)e^{-\epsilon_{d,i}}\right) \tag{81}$$

$$\overset{c}{=}\mathrm{Gumbel}\left(\epsilon_{d,i}|\log\theta^*_{d,i},1\right) \tag{82}$$

where $\theta^*_{d,i}=1+\sum_{c\neq y_i}e^{f_d^c(\mathbf{x}_i)-f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)}=\sum_{c=1}^{C_d}e^{f_d^c(\mathbf{x}_i)-f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)}$. The optimal $p\left(\epsilon_{d,i}|y_d(\mathbf{x}_i),\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)$ does have exact analytic form. However, $\mathcal{L}$ will be intractable by using an analytic form. We thus take a more general distribution $q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)=\mathrm{Gumbel}\left(\epsilon_{d,i}|\log\theta_{d,i},1\right)$, which satisfies $\theta_{d,i}>1$, also including the optimal distribution. Then the $\mathcal{L}$ is:

$$\mathcal{L}=\sum_{d}^{D}\sum_{i}^{N}\left\langle\log p\left(y_d\left(\mathbf{x}_i\right)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i),\epsilon_{d,i}\right)\right\rangle_{q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)} \tag{83}$$

$$-\sum_{q=1}^{Q}\mathrm{KL}\left(q\left(\mathbf{u}_q\right)\|p\left(\mathbf{u}_q\right)\right)-\sum_{i=d}^{D}\sum_{i=1}^{N}\mathrm{KL}\left(q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)\|p\left(\epsilon_{d,i}\right)\right). \tag{84}$$

We first consider the inner expectation in the double-expectation term in $\mathcal{L}$:

$$\left\langle\log p\left(y_d\left(\mathbf{x}_i\right)|\widetilde{\mathbf{f}}_d(\mathbf{x}_i),\epsilon_{d,i}\right)\right\rangle_{q\left(\epsilon_{d,i}|\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)} \tag{85}$$

$$= \sum_{c \neq y_d(\mathbf{x}_i)} \int_{-\infty}^{+\infty} q\left(\epsilon_{d,i} | \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \log \Phi_{\mathcal{G}}\left(\epsilon_{d,i} + f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) - f_d^c(\mathbf{x}_i)\right) d\epsilon_{d,i} \tag{86}$$

$$= - \sum_{c \neq y_d(\mathbf{x}_i)} \int_{-\infty}^{+\infty} e^{\left(-(\epsilon_{d,i} - \log\theta_{d,i}) - e^{-\left(\epsilon_{d,i} - \log\theta_{d,i}\right)}\right)} \tag{87}$$

$$e^{-\left(\epsilon_{d,i} + f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) - f_d^c(\mathbf{x}_i)\right)} d\epsilon_{d,i} \tag{88}$$

$$= \sum_{c \neq y_d(\mathbf{x}_i)} \theta_{d,i} e^{f_d^c(\mathbf{x}_i) - f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)} \frac{(1 + \theta_{d,i} u) e^{-\theta_{d,i} u}}{\theta_{d,i}^2} \Bigg|_0^{+\infty} \tag{89}$$

$$= - \frac{1}{\theta_{d,i}} \sum_{c \neq y_i} e^{f_d^c(\mathbf{x}_i) - f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)}, \tag{90}$$

where $u = e^{-\epsilon_{d,i}}$. We second consider the outside expectation. Because of

$$q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) = \prod_{c=1}^{C_d} q\left(f_d^c(\mathbf{x}_i)\right) = \prod_{c=1}^{C_d} \mathcal{N}\left(f_d^c(\mathbf{x}_i) | \mu_{f_d^c}(\mathbf{x}_i), \nu_{f_d^c}(\mathbf{x}_i)\right), \tag{91}$$

we take expression 91 and expression 90 to obtain

$$\left\langle \log p\left(y_d(\mathbf{x}_i) | \widetilde{\mathbf{f}}_d(\mathbf{x}_i), \epsilon_{d,i}\right) \right\rangle_{q\left(\widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) q\left(\epsilon_{d,i} | \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right)} \tag{92}$$

$$= - \frac{1}{\theta_{d,i}} \int e^{-f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)} q\left(f_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i)\right) df_d^{y_d(\mathbf{x}_i)}(\mathbf{x}_i) \tag{93}$$

$$\times \sum_{c \neq y_d(\mathbf{x}_i)} \int e^{f_d^c(\mathbf{x}_i)} q\left(f_d^c(\mathbf{x}_i)\right) df_d^c(\mathbf{x}_i) \tag{94}$$

$$= - \frac{1}{\theta_{d,i}} \exp\left( \frac{\nu_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)}{2} - \mu_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i) \right) \sum_{c \neq y_d(\mathbf{x}_i)} \exp\left( \frac{\nu_{f_d^c}(\mathbf{x}_i)}{2} + \mu_{f_d^c}(\mathbf{x}_i) \right) \tag{95}$$

Calculating the KL divergence $\sum_{i=d}^{D} \sum_{i=1}^{N} \mathrm{KL}\left(q\left(\epsilon_{d,i} | \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \| p\left(\epsilon_{d,i}\right)\right)$ term, we have

$$\sum_{i=d}^{D} \sum_{i=1}^{N} \mathrm{KL}\left(q\left(\epsilon_{d,i} | \widetilde{\mathbf{f}}_d(\mathbf{x}_i)\right) \| p\left(\epsilon_{d,i}\right)\right) = \sum_{i=d}^{D} \sum_{i=1}^{N} \left( \log\theta_{d,i} + \frac{1}{\theta_{d,i}} - 1 \right). \tag{96}$$

Then, the closed-form $\mathcal{L}$ is reorganized as,

$$\mathcal{L} = \sum_{d=1}^{D}\sum_{i=1}^{N}\left\{-\frac{1}{\theta_{d,i}}\mathcal{P}_{d,i} - \log\theta_{d,i} - \frac{1}{\theta_{d,i}} + 1\right\} - \sum_{q=1}^{Q}\mathrm{KL}\big(q(\mathbf{u}_q)\,\|\,p(\mathbf{u}_q)\big),$$

where $\mathcal{P}_{d,i} = \exp\left(\frac{v_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)}{2} - \mu_{f_d^{y_d(\mathbf{x}_i)}}(\mathbf{x}_i)\right)\sum_{c\neq y_d(\mathbf{x}_i)}\exp\left(\frac{v_{f_d^c}(\mathbf{x}_i)}{2} + \mu_{f_d^c}(\mathbf{x}_i)\right)$. To get a tight bound, we derivative $\mathcal{L}$ with respect to $\theta_{d,i}$,

$$\frac{\partial\mathcal{L}}{\partial\theta_{d,i}} = \sum_{d=1}^{D}\sum_{i=1}^{N}\frac{1}{\theta_{d,i}^2}\big(\mathcal{P}_{d,i} + 1\big) - \frac{1}{\theta_{d,i}} = 0 \tag{97}$$

We thus obtain the optimal value $\theta^*_{d,i} = \mathcal{P}_{d,i} + 1$. After substitution of $\theta_{d,i}$ by $\theta^*_{d,i}$, there is

$$\mathcal{L} = -\sum_{d=1}^{D}\sum_{i=1}^{N}\log\big(\mathcal{P}_{d,i} + 1\big) - \sum_{q=1}^{Q}\mathrm{KL}\big(q(\mathbf{u}_q)\,\|\,p(\mathbf{u}_q)\big) \tag{98}$$

## Appendix B Omniglot data

Table 2 shows the number of data points and classes for each alphabet in the Omniglot data set. The columns of Background set and Evaluation set have shown 30 and 20 alphabets separately.

**Table 2** Omniglot Data

| Omniglot-evaluation | $N_{data}$ | classes | Omniglot-background | $N_{data}$ | classes |
|---|---|---|---|---|---|
| Angelic | 400 | 20 | Alphabet-of-the-Magi | 400 | 20 |
| Atemayar-Qelisayer | 520 | 26 | Anglo-Saxon-Futhorc | 580 | 29 |
| Atlantean | 520 | 26 | Arcadian | 520 | 26 |
| Aurek-Besh | 520 | 26 | Armenian | 820 | 41 |
| Avesta | 520 | 26 | Asomtavruli-(Georgian) | 800 | 40 |
| Ge-ez | 520 | 26 | Balinese | 480 | 24 |
| Glagolitic | 900 | 45 | Bengali | 920 | 46 |
| Gurmukhi | 900 | 45 | Blackfoot (Canadian-Aboriginal-Syllabics) | 280 | 14 |
| Kannada | 820 | 41 | Braille | 520 | 26 |
| Keble | 520 | 26 | Burmese-(Myanmar) | 680 | 34 |
| Malayalam | 940 | 47 | Cyrillic | 660 | 33 |
| Manipuri | 800 | 40 | Early-Aramaic | 440 | 22 |
| Mongolian | 600 | 30 | Futurama | 520 | 26 |
| Old-Church-Slavonic (Cyrillic) | 900 | 45 | Grantha | 860 | 43 |
| Oriya | 920 | 46 | Greek | 480 | 24 |
| Sylheti | 560 | 28 | Gujarati | 960 | 48 |
| Syriac-(Serto) | 460 | 23 | Hebrew | 440 | 22 |
| Tengwar | 500 | 25 | Inuktitut-(Canadian-Aboriginal-Syllabics) | 320 | 16 |
| Tibetan | 840 | 42 | Japanese-(hiragana) | 1040 | 52 |
| ULOG | 520 | 26 | Japanese-(katakana) | 940 | 47 |
| | | | Korean | 800 | 40 |
| | | | Latin | 520 | 26 |
| | | | Malay-(Jawi-Arabic) | 800 | 40 |
| | | | Mkhedruli-(Georgian) | 820 | 41 |
| | | | N-Ko | 660 | 33 |
| | | | Ojibwe-(Canadian-Aboriginal-Syllabics) | 280 | 14 |
| | | | Sanskrit | 840 | 42 |
| | | | Syriac-(Estrangelo) | 460 | 23 |
| | | | Tagalog | 340 | 17 |
| | | | Tifinagh | 1100 | 55 |

## Appendix C Parameters setting

Tables 3 and 4 show the parameters setting in non-image data set and Omniglot data set respectively.

**Table 3** Setting and parameters of different GP models in non-image data sets. All the models for all data sets use 100 inducing variables and 200 mini-batch sizes. "# of Folds" indicates the number of folds for cross-validation. "Q" refers to the optimal number $Q$ of latent functions $\mathcal{U}$

| Data set | Model | Q | # of Folds |
|---|---|---|---|
| S-20 | MOGPs-AR | [5, 10, 15, 20] | 5 |
| S-20 | MG-M | [5, 10, 15, 20] | 5 |
| Balance | MOGPs-AR (1) | [1, 2, 3] | 5 |
| Balance | MG-M | [1, 2, 3] | 5 |
| Balance | G-A | None | 5 |
| Balance | G-M | None | 5 |
| CANE | MOGPs-AR (5) | [6, 9] | 5 |
| CANE | MG-M | [6, 9] | 5 |
| CANE | G-A | None | 5 |
| CANE | G-M | None | 5 |
| Mediamill | MOGPs-AR (5) | [10, 15, 20] | 5 |
| Mediamill | MG-M | [10, 15, 20] | 5 |
| Mediamill | G-A | None | 5 |
| Mediamill | G-M | None | 5 |
| Bibtex | MOGPs-AR (20) | [5, 10, 15, 20] | 3 |
| Bibtex | G-A | None | 3 |
| Bibtex | G-M | None | 3 |
| UJIIndoorLoc | MOGPs-AR (2) | [4, 8, 12] | 3 |
| UJIIndoorLoc | G-A | None | 3 |
| UJIIndoorLoc | G-M | None | 3 |

**Table 4** Setting and parameters of different GP models in Omniglot. There are three cross-validation for all models and the optimal number $Q \in [10, 15, 20]$ of latent functions $\mathcal{U}$ for MOGPs-AR. "M" indicates the number of inducing variables or inducing patches; "B" refers to the size of mini-batch. "Con-K" means convolutional kernel

| Data set | Model | Kernel | patch-size | M | B |
|---|---|---|---|---|---|
| Ojibwe & Blackfoot | MOGP-AR (1) | Con-K | 3 * 3 | 100 | 50/70/90 |
| Ojibwe & Blackfoot | G-A | Con-K | 3 * 3 | 100 | 50/70/90 |
| Ojibwe & Blackfoot | G-M | Con-K | 3 * 3 | 100 | 50/70/90 |
| Ojibwe & Blackfoot | MOGP-AR (1) | RBF-ARD | None | 40 | 50/70/90 |
| Ojibwe & Blackfoot | G-A | RBF-ARD | None | 40 | 50/70/90 |
| Ojibwe & Blackfoot | G-M | RBF-ARD | None | 40 | 50/70/90 |
| All alphabets | MOGP-AR (1) | Con-K | 8 * 8 | 200 | 9 |
| Background alphabets | MOGP-AR (1) | Con-K | 8 * 8 | 200 | 9 |
| Evaluation alphabets | MOGP-AR (1) | Con-K | 8 * 8 | 200 | 9 |

**Acknowledgements** Chunchao Ma would like to thank Chao Han, Yan Ge, Shuo Zhou and Lawrence Schobs for their feedback on the initial draft of the manuscript. Chunchao Ma also would like to thank Senee Kitimoon for his help in refactoring the code in GPflow.

**Author Contributions** (Please see submission guidelines for the format) Chunchao Ma was in charge of writing the code, performing the experimental evaluations and writing the first draft of the manuscript.

Mauricio Álvarez supervised the development of the research and provided feedback at all the stages of the process including editing the final manuscript. Chunchao Ma and Mauricio Álvarez contributed equally to the conception of the main research ideas developed in the manuscript.

**Funding** Mauricio A. Álvarez has been financed by the EPSRC Research Projects EP/R034303/1, EP/T00343X/2, EP/V029045/1, and the Wellcome Trust project 217068/Z/19/Z.

**Data availability** Our code is publicly available in the repository https://github.com/ChunchaoPeter/MOGPs-AR. All real data sets are available online at https://github.com/ChunchaoPeter/MOGPs-AR/tree/main/Data_set. The code of the synthetic data set (S-20) is available online at https://github.com/ChunchaoPeter/MOGPs-AR/blob/main/Experiment_demo/Single-output-multi-class-classifications/S-20dataset/MOGP-S20.py.

## Declarations

**Conflict of interest** The authors declare that they have no conflict of interest.

**Ethics approval and Consent to participate** The authors declare that this research did not require Ethics approval or Consent to participate since it does not concern human participants or human or animal datasets.

**Consent for publication** The authors of this manuscript consent to its publication.

## References

Alvarez, M.A. (2011). Convolved Gaussian process priors for multivariate regression with applications to dynamical systems. In PhD thesis, The University of Manchester (United Kingdom).

Álvarez, M. A., Rosasco, L., Lawrence, N. D., et al. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning, 4*(3), 195–266.

Blomqvist, K., Kaski, S., & Heinonen, M. (2019). Deep convolutional gaussian processes. *Joint european conference on machine learning and knowledge discovery in databases* (pp. 582–597). Cham: Springer.

Bonilla, E. V., Chai, K. M., & Williams, C. (2008). Multi-task Gaussian process prediction. *Advances in neural information processing systems* (pp. 153–160). New York: PMLR.

Chai, K. M. A. (2012). Variational multinomial logit Gaussian process. *The Journal of Machine Learning Research, 13*, 1745–1808.

Dahl, A., & Bonilla, E. V. (2019). Grouped Gaussian processes for solar power prediction. *Machine Learning, 108*(8–9), 1287–1306.

Dai, Z., Álvarez, M.A., & Lawrence, N.D (2017). Efficient modeling of latent information in supervised learning using Gaussian processes. arXiv preprint arXiv:1705.09862

Dezfouli, A., & Bonilla, E. V. (2015). Scalable inference for Gaussian process models with black-box likelihoods. *Advances in Neural Information Processing Systems, 28*, 1414–1422.

Dua, D., & Graff, C. (2017). UCI machine learning repository. http://archive.ics.uci.edu/ml

Galy-Fajou, T., Wenzel, F., Donner, C., & Opper, M. (2020). Multi-class Gaussian process classification made conjugate: Efficient inference via data augmentation. *Uncertainty in artificial intelligence* (pp. 755–765). New York: PMLR.

Girolami, M., & Rogers, S. (2006). Variational bayesian multinomial probit regression with gaussian process priors. *Neural Computation, 18*(8), 1790–1817.

Hensman, J., Fusi, N., & Lawrence, N.D. (2013). Gaussian processes for big data. arXiv preprint arXiv:1309.6835

Hernández-Lobato, D., Hernández-lobato, J., & Dupont, P. (2011). Robust multi-class Gaussian process classification. *Advances in Neural Information Processing Systems, 24,* 280–288.

Katakis, I., Tsoumakas, G., & Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In Proceedings of the ECML/PKDD, (pp. 5). Citeseer.

Kim, H. C., & Ghahramani, Z. (2006). Bayesian Gaussian process classification with the EM-EP algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(12), 1948–1959.

Kingma, D.P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980

Krizhevsky, A., Hinton, G., et al. (2009). *Learning multiple layers of features from tiny images*. Citeseer: Technical report.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science, 350*(6266), 1332–1338.

Lawrence, N., & Hyvärinen, A. (2005). Probabilistic non-linear principal component analysis with gaussian process latent variable models. *Journal of Machine Learning Research, 6*(11), 1783–1816.

Liu, H., Ong, Y.S., Yu, Z., Cai, J., & Shen, X. (2019). Scalable Gaussian process classification with additive noise for various likelihoods. arXiv preprint arXiv:1909.06541

Moreno-Muñoz, P., Artés, A., & Álvarez, M. (2018). Heterogeneous multi-output Gaussian process prediction. *Advances in neural information processing systems* (pp. 6712–6721). New York: PMLR.

Nguyen, T. N. A., Bouzerdoum, A., & Phung, S. L. (2018). Stochastic variational hierarchical mixture of sparse Gaussian processes for regression. *Machine Learning, 107*(12), 1947–1986.

Osborne, M.A., Roberts, S.J., Rogers, A., Ramchurn, S.D., & Jennings, N.R. (2008). Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In 2008 International Conference on information processing in sensor networks (ipsn 2008), (pp. 109–120). IEEE.

Panos, A., Dellaportas, P., & Titsias, M. (2021). Large scale multi-label learning using gaussian processes. *Machine Learning*. https://doi.org/10.1007/s10994-021-05952-5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research, 12,* 2825–2830.

Ruiz, F.J., Titsias, M.K., Dieng, A.B., & Blei, D.M. (2018). Augment and reduce: Stochastic inference for large categorical distributions. arXiv preprint arXiv:1802.04220

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In International conference on machine learning, (pp. 1842–1850).

Skolidis, G., & Sanguinetti, G. (2011). Bayesian multitask classification with Gaussian process priors. *IEEE Transactions on Neural Networks*. https://doi.org/10.1109/TNN.2011.2168568.

Snoek, C.G., Worring, M., Van Gemert, J.C., Geusebroek, J.M., & Smeulders, A.W. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia. In Proceedings of the 14th ACM international conference on Multimedia, (pp. 421–430).

Torres-Sospedra, J., Montoliu R, Martínez-Usó A, Avariento JP, Arnau TJ, Benedito-Bordonau M, Huerta J (2014) Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In 2014 international conference on indoor positioning and indoor navigation (IPIN), (pp. 261–270). IEEE.

Van der Wilk, M., Rasmussen, C. E., & Hensman, J. (2017). Convolutional Gaussian processes. *Advances in neural information processing systems* (pp. 2849–2858). New York: PMLR.

Williams, C. K., & Rasmussen, C. E. (2006). *Gaussian processes for machine learning* (Vol. 2). MA: MIT press Cambridge.

Wilson, A. G., Hu, Z., Salakhutdinov, R., & Xing, E. P. (2016). Deep kernel learning. *Artificial intelligence and statistics* (pp. 370–378). New York: PMLR.

Wistuba, M., Schilling, N., & Schmidt-Thieme, L. (2018). Scalable Gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning, 107*(1), 43–78.