CONTINUAL LEARNING WITHOUT KNOWING TASK IDENTITIES: DO SIMPLE MODELS WORK?

Anonymous authors
Paper under double-blind review

ABSTRACT

Due to the catastrophic forgetting phenomenon of deep neural networks (DNNs), models trained in standard ways tend to forget what it has learned from previous tasks, especially when the new task is sufficiently different from the previous ones. To overcome this issue, various *continual learning* techniques have been developed in recent years, which, however, often suffer from a substantially increased model complexity and training time. In this paper, we consider whether properly tailored simple models could perform well for continual learning. By proposing a relatively simple method based on Bayesian neural networks and model selection, we can in many cases outperform several state-of-the-art techniques in terms of accuracy, model size, and running time, especially when each mini-batch of data is known to come from the same task of an unknown identity. This leads to interesting observations suggesting that different continual learning techniques may be beneficial for different types of data and task diversity.

1 INTRODUCTION

Continual learning (also known as incremental learning or lifelong learning) is a paradigm in deep neural network (DNN) training that allows the model to learn new tasks while not forgetting previous tasks (De Lange et al., 2019). This is important in many practical scenarios where the training data of each task arrive in an online or streaming manner and are not stored thereafter, whereas during the inference (testing) phase, the data may come from any task. A major obstacle to continual learning is the catastrophic forgetting phenomenon of DNN training, which causes models to forget knowledge of previous tasks upon learning information from new tasks, especially when the training data of different tasks are significantly different.

Various continual learning approaches have been proposed in recent years, which can overcome catastrophic forgetting to a certain degree. However, these existing techniques have some limitations. Among them, model complexity and training speed are inadequately studied in existing works. Some existing approaches suffer from a significantly increased complexity compared to their non-continual learning counterparts. This is an important issue because, to support continual learning from online/streaming data, model training should be as fast as possible to make the best use of currently available data. In addition, except for a few recent works (Lee et al., 2020; Aljundi et al., 2019a;b), most continual learning methods require knowledge of task identity (i.e., when a new task starts) during either the training or inference phase, which is unrealistic in many practical applications. Furthermore, some existing methods require storing data samples from previous tasks (Chaudhry et al., 2019), which can violate privacy regulations, such as the GDPR (European Union, 2015), and may also be otherwise inefficient.

To resolve the above limitations, we ask the following question in this paper: *is there a simple, efficient, and effective way of continual learning?* We give a positive answer to this question by presenting a low-complexity continual learning method using Bayesian neural networks (BNNs). In our method, we initially train and store a separate BNN for each task. Upon reaching a user-specified capacity, we downselect the stored BNN models so that only the representative models remain stored. In the inference phase, we evaluate each input data sample with all the stored models and obtain the final result from the model that has the lowest degree of uncertainty in its prediction. Our method does not require task identity knowledge (i.e., it works in the *task-free* setting), where task boundaries during the training phase are automatically detected based on changes in the training

loss. We do not store any data sample from previous tasks either. At any point in time, the system only needs to access one mini-batch of data from the current task.

At a conceptual level, our method is simpler and easier to understand than many existing methods. Surprisingly, on several classical datasets and task definitions in continual learning literature, it also outperforms various state-of-the-art task-free continual learning techniques, in terms of accuracy, model size, and running (wall-clock) time. Our experiments for comparison use the code of the original papers that proposed the baseline methods, and also have the same or similar experimentation setups. The code of our method is also provided in the supplementary material.

Our finding suggests that different continual learning techniques may be beneficial for different application scenarios. Some further discussion is also given at the end of this paper.

2 Related Work

Approaches for continual learning mainly used three strategies: *expansion* (Rusu et al., 2016; Yoon et al., 2017), *rehearsal/replay* (Wen et al., 2018; Lopez-Paz et al., 2017; Feldman & Langberg, 2011; Braverman et al., 2016; Rebuffi et al., 2017; Shin et al., 2017), and *regularization* (He & Jaeger, 2018; Lee et al., 2017; Kirkpatrick et al., 2017; Rebuffi et al., 2017; Li et al., 2018; Wen et al., 2018). Our approach shares similarities with the expansion strategy, but we avoid indefinitely growing model size by enforcing a maximum storage capacity. Most of these existing approaches consider the *task-based* setting, where tasks boundaries and identities are assumed to be known in either training or inference phases.

Despite its importance, the *task-free* setting has been largely understudied, except for a few works that we describe next. Most of the existing task-free approaches are based on replay/rehearsal strategies (Aljundi et al., 2019a;b; Chrysakis & Moens, 2020; Chaudhry et al., 2019), where a small buffer of data samples, often referred to as episodic memory, is used for rehearsal purpose. Hybrid replay and expansion methods also exist (Rao et al., 2019), where expansion alone is not enough to avoid catastrophic forgetting and replay is required. Lee et al. (2020) introduced a task-free expansion method governed by Bayesian non-parametric approaches to automatically determine when to expand, which has shown to outperform many other task-free approaches, such as the works by Aljundi et al. (2019b); Chaudhry et al. (2019). However, the downside of this approach is that it is very sensitive to the choice of hyper-parameters. Furthermore, it also requires a short-term memory that collects some training data during the process. Unlike these approaches that employ either a short-term memory or replay buffer to store training samples, the approach we propose in this paper does not require storing any data point, which is a strong advantage as storing raw training samples causes privacy issues as emphasized by De Lange et al. (2019).

Our work also shares some similarities with the work by Ebrahimi et al. (2019), which also uses BNNs to perform continual learning by adapting the learning rate based on the uncertainty of network parameters. However, this approach requires each task to be trained until reaching a "plateau". The learning rate adaptation may also slow down the training process. Finally, most experiments reported in their paper require task identity knowledge during inference. In this paper, we show that our relatively simple approach, which does not require task identity knowledge in either training or inference, is able to outperform existing task-free approaches in performance, storage, and speed.

3 Preliminaries

Task-Free Continual Learning. We assume that there are K tasks. Each task corresponds to a specific distribution of training and testing data, where the data distributions for different tasks are generally different. At any point in time, only one mini-batch of data from the dataset of an arbitrary task $k \in \{1, 2, ..., K\}$ is revealed and available to the system. We consider the task-free setting where we do not know from which task each mini-batch of data comes from, but the algorithm that we present later estimates the task boundary and identity. Our system does not save any data point other than the current mini-batch.

The K tasks arrive in a sequential manner and each task may appear one or multiple times during the training phase. The goal of training is to learn a model that can accurately classify data from any of

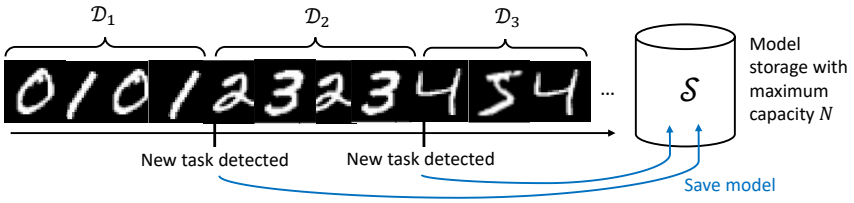

Figure 1: Overview of training procedure.

the K tasks. During the inference phase, the system receives mini-batches of the testing data from arbitrary tasks. The task identity is unknown to the system in both training and inference phases.

Bayesian Neural Networks (BNNs). BNNs have useful characteristics that are beneficial for continual learning. Unlike standard DNNs that aim to find a deterministic value of model parameters to fit the data, BNNs aim to estimate the posterior distribution $p(\mathbf{w}|\mathcal{D})$ over the model parameter vector $\mathbf{w}$, by considering the training data $\mathcal{D}$, which includes the input data and the target output for a supervised classifier, as the evidence. This posterior distribution can be estimated using Bayes' rule as $p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$, where $p(\mathbf{w})$ is an assumed prior distribution and $p(\mathcal{D}|\mathbf{w})$ is the likelihood.

In most cases, the true posterior $p(\mathbf{w}|\mathcal{D})$ is intractable because the term $p(\mathcal{D})$ cannot be efficiently computed in practice. Variational methods are used to find a parametric distribution $q(\mathbf{w};\phi)$ that approximates $p(\mathbf{w}|\mathcal{D})$, i.e., $p(\mathbf{w}|\mathcal{D}) \approx q(\mathbf{w};\phi)$. The parametric distribution q belongs to a "well-behaved" family of distributions, such as normal or exponential distributions that can be represented by a set of parameters ϕ such as mean μ and variance σ^2 (i.e., $\phi = (\mu, \sigma)$). The goal of BNN training is to find the parameter ϕ that minimizes the Kullback-Leibler (KL) divergence between the parametric distribution q and the true posterior distribution: $\phi^* = \arg\min_\phi KL(q(\mathbf{w};\phi)||p(\mathbf{w}|\mathcal{D}))$. The KL divergence is often intractable. However, minimizing the KL divergence is equivalent to minimizing the negative of the tractable evidential lower bound (ELBO) (Blundell et al., 2015):

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\phi)}[\log q(\mathbf{w};\phi) - \log p(\mathbf{w}) - \log p(\mathcal{D}|\mathbf{w})] \tag{1}$$

At the end of training, $q(\mathbf{w};\phi)$ can be used as an approximation of $p(\mathbf{w}|\mathcal{D})$. In practice, $\mathcal{L}(\phi, \mathcal{D})$ is often minimized using stochastic gradient descent (SGD) on ϕ where each iteration is based on an approximate gradient of $\mathcal{L}(\phi, \mathcal{D})$ computed on a mini-batch sampled from $\mathcal{D}$.

Continual Learning with BNNs. In theory, the Bayesian framework has an interesting property that when learning from sequential tasks with training data $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_K$, one can estimate the posterior distribution $p(\mathbf{w}|\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k)$ ($k \in \{2, ..., K\}$) by using the previous posterior as the new prior:

$$p(\mathbf{w}|\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k) \propto p(\mathcal{D}_k|\mathbf{w})p(\mathbf{w}|\mathcal{D}_1, ..., \mathcal{D}_{k-1}). \tag{2}$$

Unfortunately, as mentioned earlier, the true posterior is intractable in most practical scenarios and performing repeated approximations accumulates errors, which causes the algorithm to forget old tasks. This underlines the need for new approaches for efficient and effective continual learning.

4 Our Approach

The main idea of our approach is that we train multiple BNN models, where each model captures one or multiple similar tasks. These models can be regarded as experts for such tasks. We save up to N representative models during training, which are then used for inference, where N is a user-defined positive integer related to the desired aggregated model size and complexity. We emphasize that our system *does not* know true task boundaries or identities during either training or inference.

4.1 Training

As shown in Figure 1, during training, mini-batches of training data arrive in a sequential manner, where we do not know from which task the data comes from. Upon receiving a new mini-batch, the variational parameters ϕ get updated by performing SGD (or its accelerated variants) on $\mathcal{L}(\phi, \mathcal{D})$ defined in (1). Let t denote the iteration/mini-batch index, we use ϕ_t to denote the variational parameters after the t-th iteration.

The task boundaries are detected based on the changes of values of the log-likelihoods across mini-batches. We estimate that a new task has started if the average log-likelihood on the most recent mini-batches is sufficiently smaller than that on some previous mini-batches. Note that we assume that each task stays in the system for a number of consecutive iterations during training, before switching to the next task, which is a common assumption in existing works (Chaudhry et al., 2019; Lee et al., 2020; Ebrahimi et al., 2019). For each (detected) task identity k, we obtain a separate BNN model at the end of this task, i.e., before the algorithm determines that a new task $k+1$ has started. We use ϕ_{t_k} to denote the final variational parameters of the BNN model obtained for task k, in the t_k-th iteration right before the next task $k+1$ starts. Some representative BNN models obtained at the end of detected tasks are stored, where we select the representative models in an online manner so that we save at most N models. The detailed steps are explained as follows.

Detecting Task Boundaries. The variational parameters ϕ are updated after each SGD iteration on a minibatch. Hence, the posterior distribution of $\mathbf{w}$ also gets updated as it is approximated by $q(\mathbf{w};\phi_t)$ (at iteration t). The key idea of task boundary detection is to look at the likelihood of the current mini-batch of data on the posterior of $\mathbf{w}$ estimated from the previous data. To do so, we consider the expected log-likelihood term in (1), $l_t := \mathbb{E}_{\mathbf{w}\sim q(\mathbf{w};\phi_{t-1})}[\log p(\mathcal{D}_{(t)}|\mathbf{w})]$ with $\mathcal{D}_{(t)}$ being the current mini-batch of data, which can be approximated using Monte Carlo by sampling $\mathbf{w}$ from $q(\mathbf{w};\phi_{t-1})$. We compute and save l_t in each iteration t.

When a new task starts, ϕ is gradually adapted to fit the new task. Hence, at the beginning of a new task, ϕ still fits the old task best. To detect whether a new task has started, we define a time window of length T and in any iteration $t = nT$ ($n \in \mathbb{Z}^+$) we compute $L_n := \frac{1}{T}\sum_{\tau=(n-1)T+1}^{nT} l_\tau$. Then, we compare L_n with L_{n-1}. If $L_{n-1} - L_n > \theta$ at iteration $t = nT$, where θ is some threshold, we say that the likelihood has decreased by a significant margin and a new task started at $t = (n-1)T+1$ (and the previous task ended at $t = (n-1)T$). The reason for computing L_n on a time window T is to average out the noise that can exist in a single mini-batch.

Our empirical finding suggests that a proper choice of θ is the standard deviation of L_n over a few recent time windows starting at or after the start of the current task.

Knowledge Transfer Between Tasks. When a task switch is detected, we can use the posterior of $\mathbf{w}$ learned from the previous task as the prior of the current task, as in (2), so that we transfer knowledge from previous tasks into the current model to some degree. Such knowledge transfer is optional and depends on whether there may be correlation across different tasks. Our empirical results show that knowledge transfer may benefit certain scenarios while not in other scenarios.

Model Management. When a task ends at the t_k-th iteration, we obtain ϕ_{t_k} that captures the variational parameters for the model obtained for task k. We save up to N of such models that are jointly used for inference as we will see in Section 4.2. In addition, at any iteration $t = nT$, we save the model ϕ_{nT} in a buffer if a new task is not detected, so that we can revert back to this model later if a new task is detected after the next window $t = (n+1)T$.

Let $\mathcal{S}$ denote an *indexed set* of stored models and we always ensure $|\mathcal{S}| \leq N$. We use the task index k to denote the model ϕ_{t_k} for simplicity. When a new task $k+1$ is detected, if $|\mathcal{S}| < N$, we always save the model for the previous task k into $\mathcal{S}$. When $|\mathcal{S}| = N$, i.e., we have reached the storage (aggregated model size) limit N, we determine which models to keep in $\mathcal{S}$ in the following way.

Every time when a new task $k+1$ is detected, we compute a distance (e.g., KL divergence) between the softmax outputs[1] of the BNN model k and all models in $\mathcal{S}$, on the current mini-batch $t_k + T$, where plus T is because a new task is detected at the end of the time window of length T. Let k_i denote the i-th model currently stored in $\mathcal{S}$, i.e., $k_i \in \mathcal{S}$, where we note that we may have $i < k_i$ because some models before k_i may have been deleted. To avoid confusion, we write the current task (and its corresponding model) as $k_\bullet := k$. Let $d_{k_i,k_j}(t)$ denote the distance between models k_i and k_j computed on the t-th mini-batch. A small (correspondingly, large) value of $d_{k_i,k_j}(t)$ means that models k_i and k_j give similar (correspondingly, different) predictions on mini-batch t.

By computing $d_{k_i,k_\bullet}(t_{k_\bullet} + T)$ for all $i \in \mathcal{S}$ every time when a new task $k_\bullet + 1$ starts, we can progressively obtain all the distances $d_{k_i,k_j}(t_{k_j} + T)$ for all $i \in \{1,...,|\mathcal{S}|\}$, $j \in \{1,...,|\mathcal{S}|,\bullet\}$, and $i < j$ (we assume $\bullet > |\mathcal{S}|$). Hence, at any point in time, the system keeps the following

[1] Similar to many existing works, we consider supervised classification problems in this paper.

$|\mathcal{S}|$-by-$(|\mathcal{S}|+1)$ distance matrix:

$$\mathbf{A} = \begin{bmatrix} d_{k_0,k_1}(t_{k_1}+T) & d_{k_0,k_2}(t_{k_2}+T) & \cdots & d_{k_0,k_{|\mathcal{S}|}}(t_{k_{|\mathcal{S}|}}+T) & d_{k_0,k_\bullet}(t_{k_\bullet}+T) \\ \infty & d_{k_1,k_2}(t_{k_2}+T) & \cdots & d_{k_1,k_{|\mathcal{S}|}}(t_{k_{|\mathcal{S}|}}+T) & d_{k_1,k_\bullet}(t_{k_\bullet}+T) \\ \infty & \infty & \ddots & & \vdots \\ \infty & \infty & \infty & d_{k_{|\mathcal{S}|-1},k_{|\mathcal{S}|}}(t_{k_{|\mathcal{S}|}}+T) & d_{k_{|\mathcal{S}|-1},k_\bullet}(t_{k_\bullet}+T) \\ \infty & \infty & \infty & \infty & d_{k_{|\mathcal{S}|},k_\bullet}(t_{k_\bullet}+T) \end{bmatrix} \tag{3}$$

which is an upper-triangle matrix because we cannot compute newer models on older mini-batches as we do not save data. For convenience of minimization, we set those distances that cannot be computed as infinity. Note that if $k_i \in \mathcal{S}$ at the end of $k_\bullet$, we must have $k_i \in \mathcal{S}$ at the end of k_j for any $j > i$. Hence, the above matrix $\mathbf{A}$ can be computed progressively every time when a new task is detected, without saving any data.

When $|\mathcal{S}| = N$, we have $N+1$ models including $k_\bullet$, and we need to delete a model. To determine which model to delete, we find the smallest distance d_{k_i,k_j} in $\mathbf{A}$ and its corresponding pair of models k_i and k_j. These two models are the most similar to each other. We delete the older model k_i ($i < j$) because newer models can capture some information of older models when knowledge transfer is enabled. After deleting k_i, the distances related to k_i are also deleted. Here, we note that when a model k_m is deleted from $\mathcal{S}$, it will never be added back again and we decrement the indices i and j by one for all $i, j > m$. With this process, we never delete $k_\bullet$ because it is the newest, so we add $k_\bullet$ to $\mathcal{S}$.

4.2 Inference

At inference time, for each stored model $k \in \mathcal{S}$, we use Monte Carlo approximation to estimate the probability distribution of the output (label) $\mathbf{y}$ for a given input data sample $\mathbf{x}$:

$$p_k(\mathbf{y}|\mathbf{x}) = \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\mathcal{D}_k)} p(\mathbf{y}|\mathbf{x},\mathbf{w}) \approx \frac{1}{R}\sum_{r=1}^{R} p(\mathbf{y}|\mathbf{x},\mathbf{w}_k^{(r)}) \tag{4}$$

where $\mathbf{w}_k^{(r)}$ is sampled from $q(\mathbf{w};\phi_{t_k}^*)$ and we take the average over R samples. By computing the above for all $|\mathcal{S}|$ models in $\mathcal{S}$, we obtain the approximate probability distribution $p_k(\mathbf{y}|\mathbf{x})$ from each model $k \in \mathcal{S}$. Assume that the ground-truth label $\mathbf{y}$ includes one-hot encoded labels, our final goal is to find the predicted label $\hat{\mathbf{y}}$ that is equal to the ground-truth label $\mathbf{y}$ with high probability.

Based on the estimated probability distributions $\{p_k(\mathbf{y}|\mathbf{x}) : \forall k \in \mathcal{S}\}$, we find $\hat{\mathbf{y}}$ in the following way. We first compute an uncertainty value defined as $u_k = \sum_d \mathrm{std}_k([p(\mathbf{y}|\mathbf{x},\mathbf{w})]_d)$, where $[p(\mathbf{y}|\mathbf{x},\mathbf{w})]_d$ is the d-th element of the probability distribution $p(\mathbf{y}|\mathbf{x},\mathbf{w})$ and $\mathrm{std}_k(\cdot)$ denotes the standard deviation (estimated over R samples) when sampling $\mathbf{w}$ from $q(\mathbf{w};\phi_{t_k}^*) \approx p(\mathbf{w}|\mathcal{D}_k)$. Then, we first find the model k^* with the smallest uncertainty, i.e., $k^* := \arg\min_k u_k$; and afterwards find a one-hot encoded $\hat{\mathbf{y}}$ that has the highest probability, i.e., $\hat{\mathbf{y}} := \arg\max_{\mathbf{y}:\|\mathbf{y}\|_0=1} p_{k^*}(\mathbf{y}|\mathbf{x})$.

The rationale behind this approach is that, as the uncertainty u_k is defined as the sum standard deviation of each element y_d of $\mathbf{y}$, it captures how certain each model k is about its prediction. The most certain model most likely aligns well with the task that generated the input data.

In practice, u_k is further averaged over the entire mini-batch of data, to reduce noise in individual samples, since similarly to existing works, we assume that data within a mini-batch come from the same task. Instead of the standard deviation, other statistics such as variance may also be used, but we found that standard deviation gives the best performance empirically.

5 Experiments

Continual Learning Scenarios. We experiment our proposed approach in various continual learning scenarios following common practice in the literature, including:

- **5-Split MNIST** (Nguyen et al., 2017), where the MNIST dataset (LeCun et al., 1998) is split into 5 tasks, each containing two digit categories (i.e., 0/1, 2/3, 4/5, 6/7, 8/9), and

2-Split MNIST (Ebrahimi et al., 2019), where the original dataset is split into 2 tasks, each containing five digit categories (i.e., 0-4 and 5-9). For the 5-Split and 2-Split cases, each task has 12,000 and 30,000 training data samples, respectively.

- **Split CIFAR-10** (Lee et al., 2020), where the CIFAR-10 dataset (Krizhevsky et al., 2009) is split into 5 tasks, corresponding to the categories 0/1, 2/3, 4/5, 6/7, 8/9, respectively. Each task includes 10,000 training data samples.
- **MNIST-SVHN** (Shin et al., 2017) consists of two tasks, one task with MNIST data and the other task with SVHN data (Netzer et al., 2011). The two tasks have 60,000 and 73,000 training data samples, respectively.
- **Rotated MNIST** (Lopez-Paz et al., 2017), where each task is represented by digits rotated by a fixed angle. In most of our experiments, we define 4 tasks with angles 0°, 30°, 60°, 90°, which is more challenging than the scenario used by Farajtabar et al. (2020) with 5 tasks and smaller rotation angles 0°, 10°, 20°, 30°, 40° (results for this alternative setting are reported in Appendix A.4). Each task has 60,000 training data samples.
- **Permuted MNIST** (Goodfellow et al., 2013), where each task is a certain random permutation of the input pixels of MNIST images. The distribution of labels remains the same but the distribution of input images is different. Similar as Ebrahimi et al. (2019), we learn a sequence of 10 random permutations. Each task has 60,000 training data samples.

Compared Methods. For our proposed approach, we report two sets of results: with knowledge transfer (**Ours-T**) and with no knowledge transfer (**Ours-NT**). We compare with several other state-of-the-art *task-free* approaches:

- **Reservoir** (Chaudhry et al., 2019), which is a simple experience replay method that has been shown to outperform many other continual learning approaches. The episodic memory can be managed in a task-free setting, which makes Reservoir a strong baseline to compare with, in terms of both accuracy and complexity.
- **CN-DPM** (Lee et al., 2020), which is an expansion-based method that consists of a set of experts that learn different subsets of the data that belong to different tasks. The number of experts is governed by a non-parametric Bayesian framework. We also include an extension of CN-DPM that leverages task-homogeneity within a mini-batch (i.e., data in the same mini-batch come from the same task, see Section 4.2) during inference to select the best expert, which is referred to as CN-DPM-H.
- **UCB** (Ebrahimi et al., 2019), which is a BNN-based method that adapts the learning rate using an uncertainty defined as related to the probability distribution of BNN parameters.
- **Fine-Tuning**, which is a popular baseline used in previous works where the model is naively trained using SGD or its accelerated variants, without paying specific attention to avoiding catastrophic forgetting. The model is an ordinary neural network.
- **Bayesian Fine-Tuning (BBB-FT)**, which is similar to Fine-Tuning but the model is a BNN instead of an ordinary network.

In Appendix A.4, we also compare with task-based (i.e., not task-free) approaches.

Architecture and Other Details. For our approach, Split-MNIST, Rotated MNIST, and Permuted MNIST are trained using a BNN with a single fully-connected layer containing 64 neurons, and Split CIFAR with a single fully-connected layer containing 200 neurons. For MNIST-SVHN, we use a LeNet-5 (LeCun et al., 1998) architecture. As our approach does not assume knowledge of task identities, we employ a single head setting, which is much more challenging than the multi-head setting used by Ebrahimi et al. (2019); Farajtabar et al. (2020); Lopez-Paz et al. (2017). For our method, we use Adam optimizer with a mini-batch size of 64 and learning rate of 0.001 for all the experiments, except for Split CIFAR-10 where we use a learning rate of 0.0002 because the CIFAR-10 dataset is more complex. Apart from the experiment related to model management, the capacity (i.e., N) is equal to the number of tasks. At inference time, the number of Monte Carlo samples is set to either $R = 10$ (referred to as "Ours-*-10") or $R = 2$ (referred to as "Ours-*-2"). Our BNN implementation is based on the code by Shridhar et al. (2019)[2]. Additional details are given in Appendix A.1.

[2] https://github.com/kumar-shridhar/PyTorch-BayesianCNN

Table 1: 5-Split MNIST, Split CIFAR-10, and MNIST-SVHN

	Method	Overall Acc. (%)	#Param.	#Samples	Training time per sample (ms)	Inference time per sample (ms)
5-Split MNIST	Fine-tuning	19.73±0.01	478K	0	0.453±0.008	0.273±0.002
	Reservoir	85.83±0.53	478K	500×5	0.474±0.003	0.257±0.007
	CN-DPM	94.38±0.11	524K	500	2.921±0.040	0.381±0.006
	CN-DPM-H	99.12±0.02	524K	500	(same as above)	0.497±0.030
	Ours-NT-10	98.59±0.52	509K	0	0.228±0.006	2.551±0.101
	Ours-T-10	96.57±1.51	509K	0	0.197±0.003	2.274±0.026
	Ours-NT-2	97.54±0.67	509K	0	(same as above)	0.777±0.062
	Ours-T-2	95.26±0.86	509K	0	(same as above)	0.551±0.002
Split CIFAR-10	Fine-tuning	19.35±0.05	11.2M	0	6.61±0.031	0.929±0.004
	Reservoir	44.53±0.71	11.2M	1000×5	23.275±0.071	1.454±0.005
	CN-DPM	46.46±0.58	4.6M	1000	10.311±0.017	2.717±0.018
	CN-DPM-H	87.33±1.4	4.6M	1000	(same as above)	2.937±0.007
	Ours-NT-10	47.17±7.70	1.58M	0	0.257±0.004	2.625±0.085
	Ours-T-10	64.24±4.64	1.58M	0	0.275±0.005	2.595±0.034
	Ours-NT-2	40.50±5.88	1.58M	0	(same as above)	0.585±0.016
	Ours-T-2	56.4±1.84	1.58M	0	(same as above)	0.629±0.014
MNIST-SVHN	Fine-tuning	85.78±2.83	11.2M	0	6.49±0.031	0.798±0.003
	Reservoir	94.90±0.14	11.2M	1000×2	21.510±0.841	0.808±0.015
	CN-DPM	95.38±0.12	7.8M	1000	10.179±0.073	2.776±0.018
	CN-DPM-H	96.27±0.31	7.8M	1000	(same as above)	2.698±0.011
	Ours-NT-10	93.65±0.19	248K	0	0.342±0.012	2.214±0.033
	Ours-T-10	93.59±0.17	248K	0	0.417±0.014	2.606±0.064
	Ours-NT-2	93.15±0.21	248K	0	(same as above)	0.551±0.015
	Ours-T-2	93.18±0.15	248K	0	(same as above)	0.490±0.036

For baseline approaches, we replicate the same neural network architectures and other settings as Lee et al. (2020). All baselines results were reproduced using the original code[3] from the paper by Lee et al. (2020), except for UCB where we report results from its original paper (Ebrahimi et al., 2019) directly and adapt our setting to theirs in order to make fair comparison. For CN-DPM-H, we use a mini-batch size of 64 during inference, so that it is comparable with our method.

Unless stated otherwise, the system receives data from each task for a consecutive duration of 10 epochs during training as in Lee et al. (2020), before switching to the next task, except for UCB and its comparison where it is assumed that each task is present until training reaches a plateau as this is required for the UCB method.

Performance Evaluation. After training on sequentially arriving tasks, the model is evaluated on the test data from the union of all tasks. We compute the task-wise accuracy a_k, which is the accuracy of task k's test data after having sequentially learned all K tasks, and the overall accuracy $\overline{a} := \frac{1}{K}\sum_{k=1}^{K} a_k$. We also record the number of model parameters (floating point numbers) and the number of stored training data samples for each method. For our approach, the number of parameters is the total number of BNN parameters (mean and standard deviation values) of *all* models stored in $\mathcal{S}$. We further record the per-sample time of training and inference involving all K tasks, measured on a cloud computing instance with 8 CPU cores, 8 GB memory, and a K80 GPU.

We obtain statistics from 5 independent runs with different random seeds for each setting, and report mean and standard deviation values in Tables 1–4 where appropriate.

Results. The results on 5-Split MNIST, Split CIFAR-10, and MNIST-SVHN are shown in Table 1, with comparison to Fine-Tuning, Reservoir, and CN-DPM. We see that our method performs better than the strongest baseline CN-DPM (original version) in many cases, and in other cases we perform similarly to the baselines, in terms of accuracy and number of parameters. For the accuracy, CN-DPM-H gives the best performance here, suggesting that the availability of task-homogeneous mini-batch for inference can also significantly improve CN-DPM's accuracy for these three settings, at a cost of longer training time than our method. In all cases, the training time per sample of our method is faster (usually by an order of magnitude or more) than baseline methods, while our inference time

[3] https://github.com/soochan-lee/CN-DPM

Table 2: Rotated and Permuted MNIST

	Method	Overall Acc. (%)	#Param.	#Samples	Training time per sample (ms)	Inference time per sample (ms)
Rotated MNIST	Fine-tuning	61.93±0.73	478K	0	0.454±0.007	0.272±0.002
	Reservoir	88.49±0.79	478K	500×4	0.507±0.007	0.254±0.004
	CN-DPM	53.15±0.92	709K	500	3.984±0.149	0.373±0.004
	CN-DPM-H	57.73±0.92	709K	500	(same as above)	0.601±0.019
	Ours-NT-10	96.89±0.02	407K	0	0.190±0.004	2.051±0.097
	Ours-T-10	97.01±0.14	407K	0	0.153±0.006	1.722±0.036
	Ours-NT-2	95.91±0.08	407K	0	(same as above)	0.421±0.011
	Ours-T-2	96.51±0.11	407K	0	(same as above)	0.384±0.011
Permuted MNIST	Fine-tuning	42.62±1.54	478K	0	0.448±0.003	0.189±0.006
	Reservoir	87.47±0.60	478K	500×10	0.571±0.032	0.216±0005
	CN-DPM	14.07±3.51	1.19M	500	6.115±0.954	0.408±0.077
	CN-DPM-H	19.05±2.23	1.19M	500	(same as above)	0.661±0.121
	Ours-NT-10	95.61±0.04	1.02M	0	0.166±0.002	0.762±0.010
	Ours-T-10	91.06±0.35	1.02M	0	0.156±0002	0.69±0.017
	Ours-NT-2	94.65±0.05	1.02M	0	(same as above)	0.171±0.012
	Ours-T-2	90.08±0.32	1.02M	0	(same as above)	0.162±0.038

Table 3: Comparison with UCB and BBB-FT, 2-Split MNIST (Left) and Permutation-MNIST (Right)

Method		Overall Acc. (%)	# Param.		Overall Acc. (%)	# Param.
UCB		98.7	1.9M		92.5	1.9M
BBB-FT	2-Split	98.1	1.9M	Perm.	86.1	1.9M
Ours-NT	MNIST	98.72±0.13	204K	MNIST	96.31±0.21	1.02M
Ours-T		98.66±0.05	204K		93.23±0.32	1.02M

per sample remains similar to baseline methods and also similar to our training time. This makes our approach useful for settings where both training and evaluation data arrive to the system in a streaming manner, which are deleted after the current mini-batch. Furthermore, our approach does not require storing any training data samples, unlike Reservoir and CN-DPM. This is an advantage in terms of not only privacy but also storage. To illustrate, for the CIFAR-10 dataset, saving 1,000 samples for CN-DPM corresponds to storing 3M floating point numbers (i.e., $32 \times 32 \times 3 \times 1000$), and 5 times more for Reservoir.

Table 2 shows the results for rotated and permuted MNIST. Our approach generally outperforms the baselines with a large margin. The bad performance of CN-DPM is possibly related to hyper-parameter tuning. Although we use the same hyper-parameters as in the original paper for MNIST dataset, it does not seem to perform well for rotated and permuted MNIST. As the approach is very slow (especially since each task has 60,000 training samples), it is impractical to try a wide range of hyper-parameters. This is a shortcoming of CN-DPM that requires detailed hyper-parameter tuning which is often impractical. Note that unlike Table 1, even CN-DPM-H gives low accuracy in Table 2, suggesting that homogeneous mini-batch does not help CN-DPM much for these two settings. As CN-DPM does not seem to be a strong baseline for rotated and permuted MNIST, we also compare with additional task-based baselines in Appendix A.4 that require task identity knowledge during training, where we see that our task-free method even outperforms many state-of-the-art task-based methods.

The task-wise accuracies of the above are given in Appendix A.2.

An interesting question that arises is whether task-homogeneity within a mini-batch at inference is essential for our method. We give the accuracy results for different mini-batch sizes (during inference) in Appendix A.3. The general observation is that while the homogeneous mini-batch assumption is essential for obtaining good accuracy for class incremental settings (5-Split MNIST and Split CIFAR-10), it appears not critical for non-class incremental settings. For example, our method still gives higher accuracies for rotated and permuted MNIST than the baselines even if the mini-batch size is 1 at inference time. This suggests an interesting phenomenon that our method works well for continual learning scenarios with certain characteristics.

Table 3 compares our approach to UCB and BBB-FT, where the accuracy and number of parameters are obtained directly from the paper by Ebrahimi et al. (2019), from the only experiment where task

Table 4: Varying capacity with cyclic Rotated MNIST (12 Tasks), per task and overall accuracies (in %)

Capacity (i.e., N)	0°	30°	60°	90°	Overall
2	80.14±0.71	96.92±0.09	84.91±0.98	96.57±0.14	89.63±0.14
4	96.38±0.08	96.90±0.10	97.00±0.12	96.49±0.24	96.69±0.07
8	96.46±0.10	97.03±0.21	97.39±0.23	96.75±0.07	96.90± 0.04
12	96.36±0.07	97.50±0.01	97.51±0.01	96.82±0.07	97.07±0.05

identity was not used at inference time. As UCB requires to train each task until reaching a plateau, in our method we also assume that each task is present until training reaches a plateau to ensure fair comparison. We do not report the running time because the original paper (Ebrahimi et al., 2019) does not have such results. We see that our approach outperforms the two baselines by both the accuracy and number of model parameters.

In Table 4, we consider the case where the four different angles of rotated MNIST appear cyclically so that there are 12 tasks in total, i.e., the sequence of tasks are $0^\circ, 30^\circ, 60^\circ, 90^\circ, 0^\circ, 30^\circ, 60^\circ, ...$ The results are obtained using our method with knowledge transfer (i.e., Ours-T). We consider the impact of storage capacity N. We see that choosing $N = 4$ gives close to the highest accuracy, since there are only four unique rotation degrees out of the 12 tasks. When we set $N = 2$, the overall accuracy is still good. Furthermore, we observed that our algorithm saved models for degrees 30° and 90° when $N = 2$, which shows that our model management approach is effective because the 30° and 90° models may be still good for 0° and 60°, respectively.

6 Conclusion

In this work, we have studied an efficient continual learning technique with three characteristics: 1) low complexity and fast running time, 2) does not require task identity knowledge in either training or inference (i.e., task-free), 3) does not require storing training data samples. Our results have shown that our method can outperform various state-of-the-art baselines in terms of accuracy, storage, and speed, for certain continual learning settings. The interesting observation is that using small standalone BNN models as experts as in our method may outperform existing approaches that use a much larger model where small portions of the model represent different experts. Because our BNN models are very small, we still maintain low complexity and fast speed even though we save multiple models.

Our work suggests that, in the state-of-the-art baseline methods where multiple experts are branched out from a single complex model, the benefit of having such a complex model in those methods appears questionable at least on some datasets and task definitions. Our experiments have used datasets that are commonly used in state-of-the-art continual learning works. For more complex datasets, it is possible that some entity of sufficient complexity that is shared across multiple experts can be potentially beneficial. However, how such common entity should be constructed remains an interesting direction for future work, since existing approaches with such spirit may underperform our method that is based on small BNN models.

A limitation of this work is that the accuracy drops if the mini-batch size during inference is small, suggesting that observing multiple samples from the same task is useful for model selection. Furthermore, the improvement of CN-DPM's accuracy when applying the same homogeneous mini-batch during inference shows that identifying the correct expert may be a critical challenge in continual learning techniques that use multiple experts/models, especially for class incremental settings such as Split CIFAR-10.

References

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11254–11263, 2019a.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *Advances in Neural Information Processing Systems*, pp. 11816–11825, 2019b.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *arXiv preprint arXiv:1612.00889*, 2016.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*, 2018.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*, 2019.

Aristotelis Chrysakis and Marie-Francine Moens. Online continual learning from imbalanced data. *Proceedings of Machine Learning Research*, 2020.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *arXiv preprint arXiv:1909.08383*, 2019.

Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *arXiv preprint arXiv:1906.02425*, 2019.

European Union. General data protection regulation, 2015. URL `http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf`.

Mehrdad Farajtabar, Navid Azizan, Alex Mott, and Ang Li. Orthogonal gradient descent for continual learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773, 2020.

Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pp. 569–578. ACM, 2011.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Xu He and Herbert Jaeger. Overcoming catastrophic interference using conceptor-aided backpropagation. 2018.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In *Advances in Neural Information Processing Systems*, pp. 4652–4662, 2017.

Soochan Lee, Junsoo Ha, Dongsu Zhang, and Gunhee Kim. A neural dirichlet process mixture model for task-free continual learning. *arXiv preprint arXiv:2001.00689*, 2020.

Yu Li, Zhongxiao Li, Lizhong Ding, Peng Yang, Yuhui Hu, Wei Chen, and Xin Gao. Supportnet: solving catastrophic forgetting in class incremental learning with support data. *arXiv preprint arXiv:1806.02942*, 2018.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

Dushyant Rao, Francesco Visin, Andrei Rusu, Razvan Pascanu, Yee Whye Teh, and Raia Hadsell. Continual unsupervised representation learning. In *Advances in Neural Information Processing Systems*, pp. 7647–7657, 2019.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pp. 2990–2999, 2017.

Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*, 2019.

Junfeng Wen, Yanshuai Cao, and Ruitong Huang. Few-shot self reminder to overcome catastrophic forgetting. *arXiv preprint arXiv:1812.00543*, 2018.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

A APPENDIX

A.1 ADDITIONAL DETAILS FOR EXPERIMENTS

For our method, we initialize the prior as a normal distribution $\mathcal{N}(0, 1)$ for Split-MNIST, Rotated-MNIST, and Permuted-MNIST, and $\mathcal{N}(0, 0.01)$ for Split CIFAR and MNIST-SVHN. The variational distribution q is also a normal distribution $\mathcal{N}(\mu, \sigma^2)$, where the initial mean μ is randomly sampled from $\mathcal{N}(0, 0.01)$ and the initial standard deviation is chosen such that $\sigma = \log(1 + e^{\rho})$ where ρ is sampled $\mathcal{N}(-5, 0.1)$, as suggested in the code by Shridhar et al. (2019). The time window length T for task detection is set to the number of mini-batches in one epoch.

The results of Permuted MNIST with CN-DPM was obtained by setting the learning rates of both the generator and discriminator to $1/4$ of the original learning rates used for the MNIST dataset in the original paper by Lee et al. (2020). The reason is that, without this change, the training process with CN-DPM gets into ill-conditions that cause large fluctuations in the loss, ultimately generating a very large number of experts making the training excessively slow.

A.2 TASK-WISE ACCURACIES

Tables 5–8 report the *per task* and *overall* accuracies for 5-Split MNIST, Split CIFAR, MNIST-SVHN, Rotated MNIST, and Permuted MNIST, respectively. The accuracy is in percentages (%).

Table 5: Accuracies of 5-Split MNIST and Split CIFAR-10

5-Split-MNIST						
Method	0/1	2/3	4/5	6/7	8/9	**Overall**
Fine-Tuning	0.00±0.00	0.00±0.00	0.00±0.00	0.00± 0.00	98.48±0.05	19.73±0.01
Reservoir	91.07±0.98	78.21±1.19	74.63±2.31	85.67±1.45	99.1±0.18	85.83±0.53
CN-DPM	98.27±0.18	93.59±0.54	92.24±0.33	94.95±0.24	92.57±0.11	94.38±0.11
Ours-NT	99.92±0.02	94.63±1.20	99.54±0.18	99.73±0.08	97.81±1.01	**98.32**±0.29
Ours-T	99.91±0.01	97.85±0.93	99.30±0.18	99.51±0.17	78.87±12.40	94.94±2.41

Split-CIFAR-10						
Method	0/1	2/3	4/5	6/7	8/9	**Overall**
Fine-Tuning	0.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00	96.76±0.23	19.35±0.05
Reservoir	20.85±2.49	15.58±3.20	29.07±3.59	60.85±2.32	96.28±0.44	44.53±0.71
CN-DPM	54.19±3.38	32.67±2.34	35.63±2.46	50.63±1.54	59.16±2.40	46.46±0.58
Ours-NT	80.79±3.62	56.39±9.21	28.42±10.71	47.92±10.27	24.75±7.72	47.65±4.00
Ours-T	84.66±0.20	61.38±8.40	37.03±11.03	85.02±1.70	41.27±4.90	**61.87**±2.81

Table 6: Accuracies of MNIST-SVHN

Method	MNIST	SVHN	**Overall**
Fine Tuning	77.82±5.31	94.29±0.53	85.78±2.83
Reservoir	96.46±0.14	93.15± 0.38	94.90±0.14
CN-DPM	99.24±0.10	92.52±0.13	**95.38±0.12**
Ours-NT	98.83±0.09	85.3± 1.96	92.06±1.02
Ours-T	98.62± 0.08	88.26±0.16	93.34±0.09

Table 7: Accuracies of Rotated MNIST

Method	0°	30°	60°	90°	**Overall**
Fine-tuning	21.96±1.04	42.86±1.76	84.42±0.77	98.46±0.06	61.93±0.73
Reservoir	76.38±2.09	85.39±1.63	93.69±0.40	98.49±0.07	88.49±0.79
CN-DPM	16.42±1.65	29.03±1.40	72.78±1.11	94.40±0.10	53.15±0.92
Ours-NT	96.34±0.11	96.83±0.07	96.67±0.05	96.36±0.14	96.57±0.04
Ours-T	95.07±0.28	97.64±0.16	97.65±0.06	96.96±0.08	**96.93**±0.11

Table 8: Accuracies of Permutated MNIST

Method	T0	T1	T2	T3	T4
Fine-Tuning	12.12±2.42	15.7±1.05	19.12±4.09	16.78±3.98	24.00±4.52
Reservoir	75.46±1.62	81.15±2.65	78.84±3.14	81.93±3.17	86.22±1.55
CN-DPM	13.55±4.31	14.14±4.14	13.99±3.76	14±3.83	13.67±3.26
Ours-NT	96.64±0.11	96.84±0.11	96.38±0.11	96.13±0.5	95.67±0.14
Ours-T	96.62±0.12	97.43±0.13	96.89±0.15	88.77±0.49	89.60±0.85

Method	T5	T6	T7	T8	T9	**Overall**
Fine-Tuning	32.04±7.51	48.27±3.86	70.44±4.07	90.02±1.25	97.77±0.19	42.62±1.54
Reservoir	89.05±0.74	91.95±0.58	95.09±0.62	96.8±0.16	98.17±0.14	87.47±0.60
CN-DPM	13.46±3.45	13.32±3.45	12.81±3.48	12.03±3.20	12.03±2.87	14.07±3.51
Ours-NT	95.21±0.17	95.04±0.16	94.84±0.11	94.79±0.26	94.48±0.14	**95.60**±0.04
Ours-T	89.04±0.05	88.56±0.29	88.40±0.56	87.65±0.71	87.62±0.69	91.05±0.35

A.3 VARYING SIZE OF EVALUATION BATCH FOR PROPOSED APPROACH

The results for different mini-batch sizes at inference are shown[4] in Table 9, where $R = 10$. We see that when the mini-batch size is 1, the accuracies of 5-Split MNIST and Split CIFAR are substantially worse than those with mini-batch size of 64. However, we obtain similar accuracies for MNIST-SVHN, Rotated MNIST, and Permuted MNIST across different mini-batch sizes, which suggests that task-homogeneity within a mini-batch is more need for class incremental learning than for non-class incremental learning. Comparing with Table 2, we see that out method still gives higher accuracies than baseline methods for rotated and permuted MNIST even if the mini-batch size is 1 for inference. However, the accuracies for other datasets and task partitions are worse. This suggests an interesting phenomenon that our method works well for continual learning scenarios with certain characteristics.

[4]For permuted MNIST, only the results for one instance of the experiment are given, because we were not able to obtain the results of 5 runs before the author response deadline, thus there is no standard deviation shown for this case in the table.

Table 9: Accuracies of Different Mini-batch Sizes at Inference

		Size: 1	Size: 2	Size: 4	Size: 8
5-Split MNIST	Ours-NT-10	59.15±1.22	70.43±1.44	79.91±1.38	86.68±1.24
	Ours-T-10	56.11±1.54	68.79±1.14	78.71±0.66	85.18±093
Split CIFAR-10	Ours-NT-10	28.05±0.33	28.91±0.56	30.98±0.51	35.12±0.98
	Ours-T-10	31.17±0.67	34.22±0.81	38.67±0.95	45.08±0.31
MNIST-SVHN	Ours-NT-10	90.76±0.39	92.17±0.22	93.18±0.15	93.60±0.19
	Ours-T-10	89.82±1.03	91.61±0.56	92.91±0.31	93.49±0.16
Rotated MNIST	Ours-NT-10	89.76±0.43	93.15±0.23	95.53±0.09	96.54±0.08
	Ours-T-10	89.38±0.64	92.81±0.30	95.01±0.16	96.22±0.08
Permuted MNIST	Ours-NT-10	89.04	94.13	95.52	95.60
	Ours-T-10	83.78	88.68	90.96	91.45

		Size: 16	Size: 32	Size: 64
5-Split MNIST	Ours-NT-10	93.10±1.09	96.37±1.03	98.59±0.52
	Ours-T-10	90.06±1.48	94.07±1.31	96.57±1.51
Split CIFAR-10	Ours-NT-10	40.73±1.86	45.69±4.01	47.17±7.70
	Ours-T-10	53.01±1.69	59.13±3.11	64.24±4.64
MNIST-SVHN	Ours-NT-10	93.64±0.17	93.66±0.18	93.65±0.19
	Ours-T-10	93.60±0.17	93.61±0.16	93.59±0.17
Rotated MNIST	Ours-NT-10	96.80 ± 0.06	96.87±0.06	96.89±0.027
	Ours-T-10	96.69±0.08	96.87±0.05	97.01±0.14
Permuted MNIST	Ours-NT-10	95.64	95.65	95.61±0.04
	Ours-T-10	91.53	91.53	91.06±0.35

A.4 Comparison with Task-Based Methods

In this experiment, we compare our approach with some existing *task-based* approaches that require knowing the task identities during training, such as **Orthogonal Gradient Descent** (Farajtabar et al., 2020), an approach that restricts the direction of the gradient updates to avoid catastrophic forgetting, **A-GEM** that works by ensuring that the episodic memory loss over the previous tasks does not increase (Chaudhry et al., 2018), **EWC** (Kirkpatrick et al., 2017) which is a popular regularization technique using Fisher information to find the importance of weights, and **Fine-Tuning SGD** which is a similar baseline as Fine-Tuning used in the main paper, which shows what happens if nothing is done to avoid catastrophic forgetting. For these baseline approaches, we report the original results from the paper by Farajtabar et al. (2020) and we run our approach for the same number of epochs (i.e., 5) as for the baselines in order to compare fairly. Table 10 shows that our *task-free* approach is also able to outperform *task-based* baseline approaches. We emphasize here that even in this experiment, our approach is run without knowledge of task identities, for both training and inference, whereas all baselines have such knowledge during training.

Table 10: Comparison with task-based methods

Rotated MNIST					
Method	0°	10°	20°	30°	40°
OGD	75.6±2.1	86.6 ±1.3	91.7 ±1.1	94.3± 0.8	93.4±1.1
A-GEM	72.6±1.8	84.4 ±1.6	91.70 ±1.1	93.9± 0.6	94.6±1.1
EWC	61.9±2.0	78.1 ±1.8	89.0 ±1.6	94.4± 0.7	93.9±0.6
Fine-Tuning (SGD)	62.9±1.0	78.5 ±1.5	88.6 ±1.4	95.1± 0.5	94.1±1.1
Ours-NT	94.47±0.09	95.91±0.42	96.61±0.07	96.30±0.26	**97.11**±0.04
Ours-T	**95.26**±0.29	**96.97**±0.17	**97.18**±0.35	**97.02**±0.28	97.59±0.18

Permutated MNIST					
Method	T1	T2	T3	T4	T5
OGD	79.5±2.3	88.9 ±0.7	89.6±0.3	91.8± 0.9	92.4±1.1
A-GEM	85.5±1.7	87.0 ±1.5	89.6 ±1.1	91.2± 0.8	93.9±1.0
EWC	64.5±2.9	77.1 ±2.3	80.4 ±2.1	87.9±1.3	93.0±0.5
Fine-Tuning (SGD)	60.6±4.3	77.6 ±1.4	79.9 ±2.1	87.7±2.9	92.4±1.1
Ours-NT	94.16±0.18	95.58±0.02	96.24±0.03	**96.20**±0.13	**95.68**±0.05
Ours-T	**94.28**±0.20	**96.54**±0.11	**96.42**±0.07	95.05±0.37	89.13±2.90

A.5 ADDITIONAL EXPERIMENTS FOR MODEL MANAGEMENT

In this experiment, five tasks of Split-MNIST appear cyclically so that there are 15 tasks in total, i.e., the sequence of tasks are $0/1, 2/3, 4/5, 6/7, 8/9, 0/1, 2/3, ...$ Results reported in Table 11 show that our model management approach is able to remove redundant models and can reduce the number of stored models to 5 without affecting the accuracy. The results are obtained using our method with knowledge transfer (i.e., Ours-T).

Table 11: Varying Capacity, Cyclic Split-MNIST (15 tasks)

Capacity (i.e.,N)	0/1	2/3	4/5	6/7	8/9	Overall
5	99.80±0.23	95.66 ±1.10	99.55±0.07	99.60±0.55	97.074±0.25	98.33±0.30
15	99.92±0.02	98.17 ±0.48	99.73±0.13	99.7±0.09	96.63 ±9.39	98.89±0.17