Approximate Bayesian Class-Conditional Models under Continuous Representation Shift

Thomas L. Lee
School of Informatics
University of Edinburgh
T.L.Lee-1@sms.ed.ac.uk

Amos Storkey
School of Informatics
University of Edinburgh
a.storkey@ed.ac.uk

Abstract

For models consisting of a classifier in some representation space, learning online from a non-stationary data stream often necessitates changes in the representation. So, the question arises of what is the best way to adapt the classifier to shifts in representation. Current methods only slowly change the classifier to representation shift, introducing noise into learning as the classifier is misaligned to the representation. We propose DeepCCG, an empirical Bayesian approach to solve this problem. DeepCCG works by updating the posterior of a class conditional Gaussian classifier such that the classifier adapts in one step to representation shift. The use of a class conditional Gaussian classifier also enables DeepCCG to use a log conditional marginal likelihood loss to update the representation. To perform the update to the classifier and representation, DeepCCG maintains a fixed number of examples in memory and so a key part of DeepCCG is selecting what examples to store, choosing the subset that minimises the KL divergence between the true posterior and the posterior induced by the subset. We explore the behaviour of DeepCCG in online continual learning (CL), demonstrating that it performs well against a spectrum of online CL methods and that it reduces the change in performance due to representation shift.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 INTRODUCTION

Currently a large open problem in machine learning is how to update complex neural network models online from a non-stationary data stream (Farquhar and Gal, 2018; Antoniou et al., 2020). Methods for solving this problem can be seen as a composition of an encoder and classifier—assuming we are looking at classification. The encoder maps data instances to a representation and the classifier given a data instance in representation space assigns a class to it. One of the difficulties encountered is *representation shift* where when updating on new data the representation of old data shifts. To reduce this problem current methods try to minimise representation shift, often by regularising updates using previous data stored in memory (Delange et al., 2021; van de Ven and Tolias, 2019). However, most methods still use a standard classifier (a fully connected layer, then a softmax) when learning, which when trained normally only slowly adapts to representation shift. This means that in the meantime the classifier and representation are missaglined, introducing noise into learning and so potentially making the representation forget more information about previous data than needed (see analysis of representation shift in Section 5.1).

In this paper we propose a clean way to adapt a classifier in one step to representation shift, preventing the misalignment of the classifier and representation. To do this we propose a new method DeepCCG which leverages a Bayesian class-conditional Gaussian model for classification. The use of such a classifier has three main effects. First, the posterior of the parameters of the class-conditional model can be recomputed in one step to adjust to representation shift. This means that the classifier and the shifted representation will be better aligned, removing the potential for the representation to unnecessarily change to better fit the outdated classifier. Second, the use of a Bayesian class-conditional Gaussian classifier, allows for the use of a conditional marginal likelihood for optimizing the

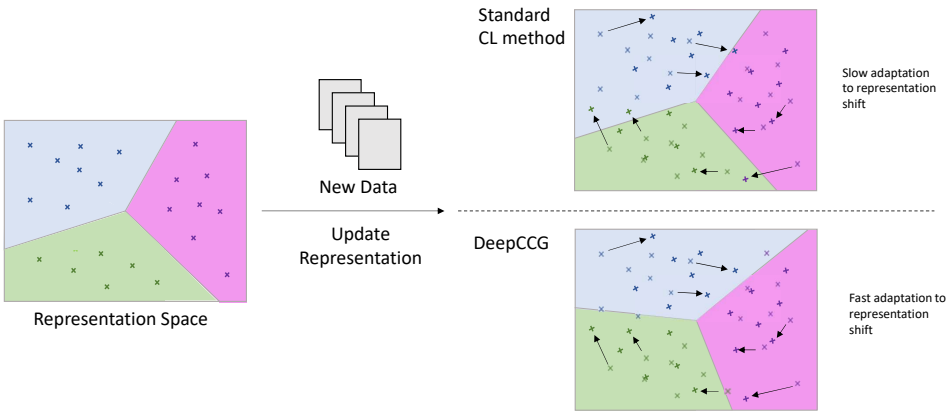

Figure 1: Diagram showing that when updating a model on new data current online continual learning methods only slowly adapt the classifier, i.e. decision boundary in representation space, to representation shift. On the other hand, DeepCCG quickly adapts the decision boundary, improving learning as the classifier is better matched to the current representation. This is illustrated in the diagram as for DeepCCG the decision boundary is adjusted in a single update such that the shifted representations, which are pointed to by arrows, all remain in the correct regions while for standard CL methods this is not the case.

parameters of the embedding function. This ensures that the representation learnt is well matched to the classifier. Third, a fixed subset of data is chosen to track representation shift, by selecting the subset that best recreates the posterior distribution. This method for selecting samples is shown to be robust to general kinds of representation shift (see Section 4) and is required for DeepCCG to perform well (see ablation study in Section 5.1).

To examine the behaviour of DeepCCG we look at the online continual learning (CL) setting (Chaudhry et al., 2019a; Aljundi et al., 2019b). This is a common setting where a learner sees a non-stationary stream of batches of data. The results of our experiments show that DeepCCG performed best out of all methods tested, highlighting the benefit of quickly adapting to representation shift.

The main contributions of this work are:

- A method DeepCCG, which by learning the classifier in a tractable Bayesian manner adapts to representation shift in one step and performs best in our online CL experiments.
- Use of a log conditional marginal likelihood loss term to fit the embedding function, which reduces representation shift and aligns the representation to the class-conditional Gaussian model.
- A new method to select what examples to store in memory, by minimising the KL-divergence between the true posterior and the one induced by the subset of data to be stored in memory.

2 RELATED WORK

There are three main paradigms for solving online CL problems: *regularisation*, *parameter-isolation* and *replay* (Delange et al., 2021). Our work is most closely related to *Replay* methods which aim to solve CL problems by storing a subset of previously seen examples, which are then trained on alongside new incoming data. Replay methods have been shown to have competitive if not the best performance across many settings (van de Ven and Tolias, 2019; Wu et al., 2022; Mirzadeh et al., 2020). The standard replay method is experience replay (ER) (Chaudhry et al., 2020, 2019b; Aljundi et al., 2019a), which in the online setting looked at in this work, selects examples to store using reservoir sampling (Vitter, 1985), we call this variant ER-reservoir (Chaudhry et al., 2020). One of the main questions to be answered by a replay-based approach is how to select what examples to store in memory. While reservoir sampling has been shown to be very effective (Wiewel and Yang, 2021) there have been other methods proposed for sample selection, for example ones which use information-theoretic criteria (Wiewel and Yang, 2021) and others maximising the diversity of the gradients of stored examples (Aljundi et al., 2019b). There has also been a Bayesian method proposed to select samples called InfoGS (Sun et al., 2022), which is somewhat similar to our method but only uses the probability model to select samples, not for prediction or training. Additionally, their exists methods to select what examples to replay at each update step (Aljundi et al., 2019a; Shim et al., 2021) which are complementary/orthogonal to this work.

There has been considerable work on using Bayesian methods in CL (Kessler et al., 2023; Ebrahimi et al., 2020; Kurle et al., 2020; Lyu et al., 2023), perhaps inspired by the fact that true online Bayesian inference cannot suffer from catastrophic forgetting (Nguyen et al., 2018). Bayesian perspectives have mainly been used for regularisation based methods, where a popular approach is to use variational inference (Nguyen et al., 2018; Farquhar and Gal, 2019; Zeno et al., 2018). These variational inference methods focus on the offline CL setting where a learner has access to all of the data of a task at the same time and, in previous work, are often limited to being used in conjunction with small neural networks, mainly due to the need to sample multiple networks when calculating the loss (Henning et al., 2021; Nguyen et al., 2018). Therefore, these methods are not suited to the settings we consider in this paper. Bayesian methods have also been used in generative replay based approaches, where instead of storing and replaying real samples they use generated pseudo-samples (Rao et al., 2019). Generative replay methods focus on the offline scenario, where it is possible to iterate over the whole of a task's data to fit a generator, while we look at the more realistic online scenario in this work. Finally, when it comes to replay with real examples, there has been relatively little work on using Bayesian methods, which we aim to help to fill in by proposing DeepCCG.

A closely related work to DeepCCG is iCaRL (Rebuffi et al., 2017). It is similar because both methods use a class-conditional Gaussian classifier. However, iCaRL only uses a class-conditional Gaussian classifier at test time, using a per-class sigmoid classifier in training. Therefore, iCaRL does not solve the problem we set out to address, that of quickly adapting to representation shift in training, as its classifier in training is similar to other standard CL algorithms and only slowly adapts to representation shift. We also note that while iCaRL's sample selection mechanism is different from DeepCCGs, it can be seen to approximate the same general objective—selecting the subset which best recreates the mean of the whole data, in representation space. Therefore, our principled Bayesian derivation of DeepCCG's sample selection mechanism can be applied to iCaRL's giving it a theoretical grounding, which is not looked at in the iCaRL paper. Another similar approach to DeepCCG is ProtoCL (Kessler et al., 2023), which appeared after DeepCCG's initial release. It is similar in that it uses a class-conditional Gaussian classifier but the learning of the classifier and representation is different and ProtoCL uses random sampling to select what examples to store in memory, which for DeepCCG is shown to have bad performance (see Table 2).

3 ONLINE CL

This work uses the Online CL setting to study how to deal with representation shift; where a learner sees a non-stationary data stream consisting of batches of data. Specifically, we consider here classification problems. Let X denote a shared data space, and C denotes the set of all classes being considered. Additionally, let $t \in T$ denote a particular task. A task t is a data generator which generates data such that each each example consists of a data instance $\mathbf{x} \in X$ and associated label $y \in C_t$. Also, $C_t \subseteq C$ are the subset of classes which all the data generated by task t must belong to.

During training, the learner receives a temporal sequence of data batches with their task identifiers, $((B_j, t_{B_j})|j = 1, 2, \ldots, N)$. Each batch B_j consists of a set of examples drawn from the same task. The task identifier t_{B_j} denotes what task that is, effectively indicating the classes which a data instance $\mathbf{x}$ of that task can belong to. The learner trains on each batch in turn and after training on a batch it must discard all the data it does not store in a fixed-size memory buffer M. Also, the learner never revisits the same batch again.

At test time, we consider two evaluation scenarios, task-incremental and class-incremental learning. For both scenarios the learnt model is evaluated on an unseen test set from all the classes seen during training. For task-incremental learning the learner is provided with task identifiers for each test data instance and without in class-incremental learning. This experimental setup is commonly used in the continual learning literature (Chaudhry et al., 2019a). Additionally, we provide a reference table of terms used in online CL and defined in this section in Appendix C.

4 DEEP CLASS-CONDITIONAL GAUSSIANS

We propose an empirical Bayesian method DeepCCG, which can adapt quickly to representation shift. DeepCCG consists of three main parts. Firstly, a Bayesian class-conditional Gaussian classifier is used on top of a neural network embedding function. The classifier is fit by recomputing its posterior whenever there is a shift in representation; which only requires a single forward pass through the neural network embedding function. Secondly, by using a class-conditional Gaussian classifier we can learn the embedding function using a log conditional marginal likelihood loss. The loss is derived from and uses the learnt classifier in an end-to-end manner and aims to fit a representation that best allows for the classifier to discrimi-

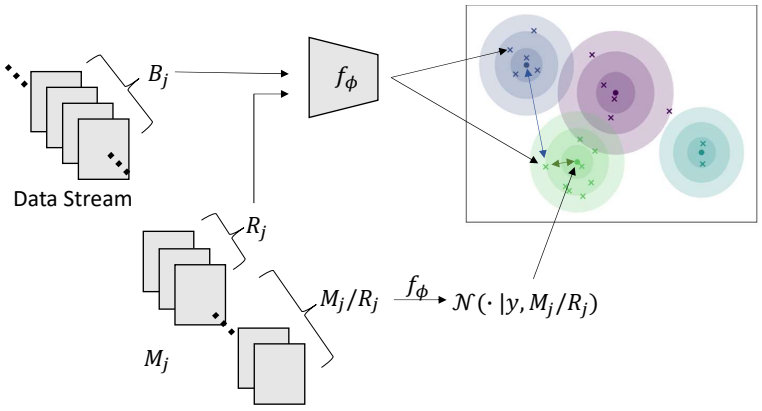

Figure 2: Diagram of DeepCCG's training routine. At time j the learner is given a sample of data B_j and has a memory of stored datapoints M_j. The memory is randomly split into replay data R_j and the rest M_j/R_j. Learning happens by taking a gradient step on the parameters of the embedding function $\boldsymbol{\phi}$ using a log conditional marginal likelihood function over B_j and R_j, where M_j/R_j is used to induce a posterior over the means of the per-class Gaussians and so define the conditional marginal likelihood function used. Therefore, training aims to move the data points into per-class clusters, by drawing the embedded examples of B_j and R_j towards their own class means and away from the other class means, as shown by the coloured arrows in the figure.

nate between classes. Additionally, by conditioning the marginal likelihood on some stored data, the loss aims to minimise the representation shift of previously seen classes. Finally, DeepCCG uses a new method based on minimising information loss to select a subset of examples to store in memory, these examples are needed when updating both the classifier and the embedding function, to model and reduce representation shift. The sample selection mechanism is shown to be robust to certain kinds of representation shift. These three parts are described in detail below and additionally a pseudocode description of the learning of DeepCCG is presented in Algorithm 1.

Probabilistic Classifier We use a Bayesian class-conditional Gaussian model as the classifier for DeepCCG. This model is defined as follows. Let Z denote a representation space and assume that we have a neural network embedding function $f_{\boldsymbol{\phi}} : X \to Z$. We define a class-conditional Gaussian model in the representation space Z,

$$y|t \sim \mathrm{Cat}(C_t, (1/|C_t|)\mathbf{1}) \tag{1}$$
$$\mathbf{z}|y \sim \mathcal{N}(\boldsymbol{\mu}_y, \mathbf{I}) \tag{2}$$
$$\boldsymbol{\mu}_y \sim \mathcal{N}(\mathbf{0}, \mathbf{V}_0), \tag{3}$$

where t is a given task identifier, $\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x})$, $\{\boldsymbol{\mu}_y|y \in C\}$ are the parameters of the classifier being the per-class means in Z and $\mathrm{Cat}(C_t, (1/|C_t|)\mathbf{1})$ is the uniform categorical distribution over the classes of task t. We choose $\mathbf{V}_0 = a\mathbf{I}$, where in practice we take $a \to \infty$. We do not specify a model where the covariance matrix is also a parameter as we learn the embedding function, hence the case of a global shared covariance is implicitly covered through learning a linear remapping to a fixed covariance, meaning we do not lose flexibility by assuming fixed covariances. Also, we assume the labels are conditionally independent given the task, the per-class means $\{\boldsymbol{\mu}_c|c \in C\}$ and the data instances (each an element of X) and that $\mathbf{z}$ is independent of the task given its class y.

The classifier is defined as the posterior predictive distribution of the model, i.e. $p(y|\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}), t_{\mathbf{x}}, D)$ given the data instance $\mathbf{x}$, its task identifier $t_{\mathbf{x}}$ and some

Algorithm 1 DeepCCG update step at time j

input B_j (training batch), t_{B_j} (task identifier for batch), M_j (current memory), $f_{\boldsymbol{\phi}_j}$ (embedding function)

Update class-conditional Gaussian classifier and embedding function:
$R_j = \mathrm{UniformSample}(M_j)$
Calculate posterior of class-conditional Gaussian classifier: $p(\boldsymbol{\mu}_c|M_j/R_j)$ for each class c, using Eq. 7
Calculate $\log(p(y|\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x}), t_{(\mathbf{x},y)}, M_j/R_j))$, for each $(\mathbf{x}, y) \in B_j \cup R_j$, using Eq. 8 and posteriors $p(\boldsymbol{\mu}_c|M_j/R_j)$
Update embedding function: $\boldsymbol{\phi}_{j+1} = \boldsymbol{\phi}_j + \eta\nabla_{\boldsymbol{\phi}} \sum_{(x,y)\in B_j\cup R_j} \log(p(y|\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x}), t_{(\mathbf{x},y)}, M_j/R_j))$

Update memory buffer:
for each class c in M_j **do**
  initialise $\boldsymbol{\beta}$
  for 1 to B **do**
    $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \eta\nabla_{\boldsymbol{\beta}}\mathcal{L}(\boldsymbol{\beta}; B_{j,c}, M_{j,c})$, where $\mathcal{L}(\boldsymbol{\beta}; B_{j,c}, M_{j,c})$ is defined in Eq. 12
  end for
  Set M_{j+1} to be the set of examples, including their task identifiers, with the m largest values in $\boldsymbol{\beta}$
end for

previously seen data D. The posterior predictive distribution can be calculated by using the equivalence,

$$p(y|\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}), t_{\mathbf{x}}, D) = \int p(y|\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}), t_{\mathbf{x}}, \boldsymbol{\theta} = \{\boldsymbol{\mu}_c | c \in C\}) p(\boldsymbol{\theta}|D) d\boldsymbol{\theta}, \quad (4)$$

where

$$p(y|\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}), t_{\mathbf{x}}, \boldsymbol{\theta} = \{\boldsymbol{\mu}_c | c \in C\}) = \frac{p(\mathbf{z}|y, \boldsymbol{\mu}_y) p(y|t_{\mathbf{x}})}{\sum_{c 2 C} p(\mathbf{z}|c, \boldsymbol{\mu}_c) p(c|t_{\mathbf{x}})}, \quad (5)$$

and $p(\boldsymbol{\theta}|D)$ is the product over the posteriors $p(\boldsymbol{\mu}_c|D)$ for each class, where

$$p(\boldsymbol{\mu}_c|D) = p(\boldsymbol{\mu}_c|D_c^Z) \quad (6)$$

$$= \mathcal{N}\left(\boldsymbol{\mu}_y; \overline{D_c^Z}, \frac{1}{|D_c^z|}\mathbf{I}\right). \quad (7)$$

$D_c^Z = \{f_{\boldsymbol{\phi}}(\mathbf{x}) | (\mathbf{x}, y) \in D \wedge y = c\}$ are the representations of the data instances in D of class c and we use the notation $\overline{S}$ to denote the mean of the elements of a set S. The integral in Eq. 4 can be computed, giving a closed form expression for the classifier output (see Appendix A for the full derivation)

$$p(y|\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x}), t_{\mathbf{x}}, D) = \frac{\mathcal{N}\left(\mathbf{z}; \overline{D_y^Z}, (1 + \frac{1}{jD_y^Z j})\mathbf{I}\right)}{\sum_{c 2 C_{t_{\mathbf{x}}}} \mathcal{N}\left(\mathbf{z}; \overline{D_y^Z}, (1 + \frac{1}{jD_y^Z j})\mathbf{I}\right)}. \quad (8)$$

This shows that the classifier gives the probability of $\mathbf{x}$ belonging to a class y by how close its representation $\mathbf{z} = f_{\boldsymbol{\phi}}(\mathbf{x})$ is to the class mean $\overline{D_y^Z}$ and by how many examples DeepCCG has already seen of a class, i.e. $|D_y^Z|$.

To fit the classifier, DeepCCG only needs to compute the posterior of the per-class means. This is because this posterior can be used to create the posterior predictive distribution of any data instance. The posterior is tractable and easy to compute, as shown in Eq. 7. This is due to our choice of model which is the main reason we use this model along with its good performance when used on top of neural network embeddings (Ostapenko et al., 2022; Hayes and Kanan, 2020). If $f_{\boldsymbol{\phi}}$ is fixed, so there is no representation shift, we can update the posterior of the per-class means on seeing batch B_j and its task identifier t_{B_j} by simply using Bayes rule, calculating the posterior $p(\{\boldsymbol{\mu}_c | c \in C\}|B_j, B_{<j})$, where $B_{<j}$ is the union of the batches seen before B_j. However, in our setting, we need to update $f_{\boldsymbol{\phi}}$ continually and so there is representation shift, which changes the value of $p(\{\boldsymbol{\mu}_c | c \in C\}|B_j, B_{<j})$. We cannot recompute $p(\{\boldsymbol{\mu}_c | c \in C\}|B_j, B_{<j})$ as we are unable to store all the previously seen examples. Therefore, DeepCCG stores in memory a buffer of representative examples M and their task identifiers with which we can compute an approximate posterior $p(\{\boldsymbol{\mu}_c | c \in C\}|M)$ after a change in $f_{\boldsymbol{\phi}}$. It is important to note that we can compute this posterior using a single forward pass through $f_{\boldsymbol{\phi}}$ and so we can adapt the classifier in one step to representation shift, improving learning as the classifier and representation are never misaligned.

Learning the Embedding DeepCCG uses a novel log conditional marginal likelihood loss term update the embedding function. The log conditional marginal likelihood loss is given by the Bayesian classifier and uses the classifier so that the loss fits a representation which enables the classifier to best separate the classes. At time j, in response to the arrival of batch B_j, with task identifier t_{B_j}, and having the memory buffer M_j, DeepCCG takes the following steps to update the current embbeding function $f_{\boldsymbol{\phi}_j}$ (shown in Figure 2). First, it selects a random set $R_j \subset M_j$, of size r, from the memory buffer to replay. Then it calculates the output of the classifier for each example $(\mathbf{x}, y) \in B_j \cup R_j$ in the current batch and replay set. The classifier output is the probability of the class being y for $\mathbf{x}$ given by the posterior predictive distribution $p(y|\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x}), t_{(\mathbf{x},y)}, M_j/R_j)$, where we condition on the rest of the stored examples M_j/R_j and $t_{(\mathbf{x},y)}$ is the task identifier for the example. The posterior predictive distributions are calculated using Eq. 8 (where in this case $D = M_j/R_j$). Then the posterior predictive distributions are used as individual log conditional marginal likelihood terms. This leads to the embedding update rule:

$$\boldsymbol{\phi}_{j+1} = \boldsymbol{\phi}_j + \eta \nabla_{\boldsymbol{\phi}} \sum_{(\mathbf{x},y) 2 B_j [R_j} \log(p(y|\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x}), t_{(\mathbf{x},y)}, M_j/R_j)). \quad (9)$$

DeepCCG conditions the marginal likelihood on some stored data instead of using, as standard, only the marginal likelihood. The reason for this is that conditioning the likelihood on some examples M_j/R_j stabilises the output of the classifier and hence the loss term used. This is because by conditioning the marginal likelihood on M_j/R_j, the posterior DeepCCG averages over is more informative than using the unconditioned prior, having some belief of where the position of embedded data instances should be and so provides a better signal to fit the embedding function. Additionally by replaying the examples R_j, treating them like new data, the loss leverages a new type of replay, where performing replay is widely known to

be effective at minimising unnecessary representation shift (Wu et al., 2022).

Sample Selection A key component to DeepCCG is how to select samples to store in the memory buffer. We focus on ensuring the least amount of information is lost about the position of the per-class means of the class-conditional Gaussian classifier, after a shift in representation. Hence, because all the information available to inform the value of the classifiers per-class mean parameters is encapsulated in the full posterior over the seen data, we target storing a set of examples that best recreate that posterior, preventing as much information loss as possible. Therefore, to perform sample selection we minimise the KL divergence between two posterior distributions: the posterior over parameters induced by the new memory being optimized, and the posterior induced by the current batch and the old memory. We keep the number of examples in memory for each class balanced, so minimising the KL divergence is equivalent to minimising each per-class KL divergence. Therefore, we select the new memory for each class using

$$M_{j+1,y} = \arg\min_{M_y^0}(\mathrm{KL}(p(\boldsymbol{\mu}_y|B_{j,y}, M_{j,y})\|p(\boldsymbol{\mu}_y|M_y^0))) = \arg\min_{M_y^0}(\|\overline{B_{j,y}^Z \cup M_{j,y}^Z} - \overline{M_y^{0Z}}\|_2^2), \quad (10)$$

where $M_y^0 \subseteq B_{j,y} \cup M_{j,y}$, $M_{j+1,y}$ is the new memory to be selected for class y, $M_{j,y}$ is the current memory for class y and $B_{j,y}$ is the set of examples of class y in the current batch. Eq. 10 shows that DeepCCG selects the data to store such that they have as close to the same mean as the old memory plus current batch as possible. Importantly, this sample selection mechanism is robust to representation shift, when modeled as an additive i.i.d. change in representation. This is because, given the shifted representations $\mathbf{z} = \mathbf{z} + \boldsymbol{\epsilon}_{\mathbf{z}}$, where $\mathbf{z} \in D^Z \cup M^Z$ and $\boldsymbol{\epsilon}_{\mathbf{z}}$ are i.i.d. sampled from an arbitrary distribution with bounded mean and variance, we have that (proved in Appendix B):

$$\mathsf{E}[\mathrm{KL}(p(\boldsymbol{\mu}_y|D_{j,y}^Z, M_{j,y}^Z)\|p(\boldsymbol{\mu}_y|M_y^{0Z}))] = \|\overline{B_{j,y}^Z \cup M_{j,y}^Z} - \overline{M_y^{0Z}}\|_2^2 + \nu(\frac{1}{|M_y^{0Z}|} - \frac{1}{|B_{j,y}^Z \cup M_{j,y}^Z|}), \quad (11)$$

where the expectation is taken over the additive shift terms $\boldsymbol{\epsilon}$, ν is the sum over the per-dimension variances of the shift distribution, $D_{j,y}^Z = \{\mathbf{z} = \mathbf{z} + \boldsymbol{\epsilon}_{\mathbf{z}} | \mathbf{z} \in D_{j,y}^Z\}$ and $M_{j,y}^Z$ is defined likewise. Therefore, in expectation, the examples selected to be stored in memory after a shift in representation are still the best examples to store in memory, in terms of preserving posterior information. Hence, our sample selection mechanism is robust to this type of representation shift.

Performing the minimization in Eq. 10 is computationally hard, so we utilise a relaxation of the problem using *lasso* (Hastie et al., 2009), whereby our method selects the new memory by assigning to each embedded input z_i a zero-to-one weight β_i and performing gradient decent on the loss

$$\mathcal{L}(\boldsymbol{\beta}; B_{j,y}, M_{j,y}) = \lambda\|\boldsymbol{\beta}\|_1 + \left\|\overline{B_{j,y}^Z \cup M_{j,y}^Z} - \frac{1}{\|\boldsymbol{\beta}\|_1} \sum_{\substack{ij(\mathbf{x}_i, y_i) \\ B_{j,y} \cup M_{j,y}}} \beta_i z_i\right\|_2^2. \quad (12)$$

Then, after termination, our method selects the m examples with the largest weights in $\boldsymbol{\beta}$ to be the examples stored in memory, where they are stored along with their task identifiers.

5 EXPERIMENTS

Benchmarks In our experiments we look at task and class incremental learning in both the disjoint tasks and shifting window settings. Furthermore, we consider three different datasets: CIFAR-100 (Krizhevsky, 2009), MiniImageNet (Vinyals et al., 2016) and CIFAR-10 (Krizhevsky, 2009), where both CIFAR-100 and MiniImageNet contain 100 classes, while CIFAR-10 contains 10 classes. For disjoint tasks, we split data evenly across a certain number of tasks while assigning all examples with a particular class to only one task—this is often called the 'split tasks' setting in previous work (Delange et al., 2021; Chaudhry et al., 2019a). We split CIFAR-10 into 5 tasks where there are 2 classes per task and for CIFAR-100 and MiniImageNet we split the dataset into 20 tasks with 5 classes per task. In the alternative shifting window setting, we split the datasets up into tasks by fixing an ordering of the classes $c_1, \ldots, c_k$ and construct the ith task by selecting a set of examples from classes $c_i, \ldots, c_{i+l}$, where l is the window length. No two tasks contain the same example and each task has the same number of examples per-class. For CIFAR-10 we use a window length of 2 and for CIFAR-100 and MiniImageNet we use a window length of 5. The shifting window setting is somewhat like the blurry task setting considered in other work (Bang et al., 2021), where they have in common that there is class overlap between tasks. However, in the shifting window setting there is also temporal locality between the classes seen. Additionally, for all experiments we train with 500 examples per-class.

We evaluate the methods using a standard metric for CL, average accuracy (Chaudhry et al., 2019a). The average accuracy of a method is the mean accuracy on a reserved set of test data across all tasks after training on all tasks.

Table 1: Results of task-incremental and class-incremental learning experiments on CIFAR-10, CIFAR-100 and MiniImageNet, where SW and DT stand for the shifting window and disjoint tasks settings, respectively. We report mean average accuracy with their standard errors across three independent runs. The results show that DeepCCG performs the best out of the methods tested.

		CIFAR-10		CIFAR-100		MiniImageNet	
Scenario	Method	SW	DT	SW	DT	SW	DT
Task-Inc.	EWC	58.68 1.88	63.33 0.87	36.65 1.07	42.39 0.79	32.42 1.13	29.57 0.69
	PackNet	69.29 2.27	66.97 1.47	40.21 0.96	50.28 0.58	34.15 1.28	37.86 1.71
	ER-reservoir	70.44 0.81	66.71 0.90	54.05 0.63	58.31 1.08	41.04 1.54	40.29 1.08
	A-GEM	57.63 2.16	57.28 2.61	29.01 1.45	39.00 0.75	26.97 1.26	30.08 1.86
	EntropySS	67.93 0.63	64.96 0.91	51.80 0.70	56.75 0.81	40.03 0.61	41.12 0.51
	GSS	71.55 1.45	67.30 1.27	48.20 0.33	49.92 0.50	37.91 0.49	38.77 0.98
	ER-ACE	71.01 0.81	68.94 0.24	52.65 0.09	54.57 0.61	40.06 0.56	39.42 0.20
	DER++	70.86 1.24	67.79 0.84	53.92 1.05	57.08 0.79	41.22 0.35	41.95 1.19
	ESMER	71.00 0.67	65.13 1.92	53.58 0.21	56.90 0.21	41.09 0.93	42.46 0.58
	iCaRL	63.53 0.12	67.98 0.90	31.77 0.33	35.99 0.55	30.96 0.75	30.47 1.29
	DeepCCG (ours)	**74.65 2.00**	**69.29 0.89**	**56.62 0.29**	**60.46 0.24**	**42.18 0.45**	**43.04 0.64**
	SGD	63.21 2.09	63.58 0.31	35.63 1.27	42.50 1.40	33.23 0.67	31.61 0.70
	Multi-Task (UB)	96.20 0.69	73.37 1.82	90.42 1.87	59.76 0.49	89.81 0.25	48.46 1.21
Class-Inc.	EWC	14.36 1.64	13.85 2.17	3.77 0.39	4.51 0.19	2.73 0.21	2.84 0.19
	ER-reservoir	18.96 1.18	16.29 0.75	7.76 0.91	7.19 0.56	5.93 1.05	5.15 0.22
	A-GEM	11.98 0.82	14.88 0.52	1.92 0.04	3.01 0.13	1.21 0.16	1.83 0.04
	EntropySS	8.93 1.11	15.69 0.78	5.85 0.47	7.87 0.71	2.93 0.45	5.63 0.40
	GSS	16.11 1.23	17.61 0.30	6.34 0.20	5.18 0.44	4.52 0.25	4.84 0.16
	ER-ACE	23.76 1.61	20.77 1.55	13.51 0.34	14.72 0.32	8.78 0.48	8.29 0.33
	DER++	18.81 1.17	15.37 1.42	9.53 0.35	7.74 0.17	6.19 0.19	5.78 0.10
	ESMER	17.07 0.89	17.88 0.80	9.04 0.55	7.93 0.14	6.45 0.15	5.77 0.12
	iCaRL	19.48 0.63	19.96 0.80	4.39 0.12	5.02 0.19	3.31 0.11	3.34 0.04
	DeepCCG (ours)	**26.87 0.47**	**24.27 0.44**	**18.00 0.72**	**17.32 0.27**	**11.71 0.39**	**11.88 0.71**
	SGD	13.59 1.76	13.76 1.22	4.09 0.14	3.93 0.29	1.28 0.06	2.14 0.22
	Multi-Task (UB)	40.57 2.33	40.57 2.33	20.22 0.42	20.22 0.42	12.41 0.69	12.41 0.69

Methods We compare DeepCCG to representative methods of the main paradigms of CL. For regularisation methods we compare against a fixed memory variant of EWC (Huszár, 2018; Kirkpatrick et al., 2017); for parameter-isolation methods we compare to PackNet (Mallya and Lazebnik, 2018), but only in the task-incremental setting as it requires task identifiers at test time. For replay methods, which includes DeepCCG, we compare against ER-Reservoir (Chaudhry et al., 2020), A-GEM (Chaudhry et al., 2019a), EntropySS (Wiewel and Yang, 2021), GSS (Aljundi et al., 2019b), ER-ACE (Caccia et al., 2021), DER++ (Buzzega et al., 2020), ESMER (Sarfraz et al., 2023) and iCaRL (Rebuffi et al., 2017). A description of each method compared against is given in Appendix D and we note here that EntropySS and GSS are memory sample selection strategies for ER. When training, all methods are given task identifiers which state what classes are in a task. Task identifiers are also given at test time in the task-incremental scenario while in class-incremental learning the task-identifier at test time is treated as stating a data instance can belong to any class. We also compare against two baselines: SGD which is when learning is performed using SGD with no modification and Multi-Task which is an upper bound and is when we learn the base neural network offline—with task identifiers for the task-incremental experiments and without for the class-incremental experiments. All methods use the same embedding network for all experiments which is a ResNet18 with six times fewer filters and Instance Normalisation (Ulyanov et al., 2016) instead of Batch Normalisation layers (Ioffe and Szegedy, 2015), which is similar to other work (Mirzadeh et al., 2020; Farajtabar et al., 2019). A batch size of 10 is used throughout, with all replay methods having a replay size of 10. A memory size of 10 examples per-class is used for the task-incremental setting, and 30 for the harder class-

incremental setting. Further experimental details are provided in Appendix E.

5.1 Results

In the task-incremental experiments, we see from Table 1 that DeepCCG performs the best out of the methods compared. For example, DeepCCG on CIFAR-100 achieves mean average accuracies of 56.62% and 60.46% for the shifting window and disjoint tasks settings, respectively, which are 2.57% and 2.15% better than any other method. The next best performing methods are ER-reservoir, ER-ACE, ESMER and DER++, which all achieve a second best performance in one of the dataset and setting combination. However there is not one single method that consistently performs second best across all datasets and settings. Overall, DeepCCG had an average performance improvement of 1.62% over the other methods in the task-incremental experiments. Therefore, our experiments show that DeepCCG performs well in task-incremental learning where it is hard to get large performance improvements (Prabhu et al., 2020).

For the experiments on class-incremental learning, we see that DeepCCG performs well, outperforming the other methods. The results of the class-incremental experiments are shown in Table 1 and displays for instance that DeepCCG on CIFAR-100 achieves 4.49% and 2.6% better than any other method, for the shifting window and disjoint tasks settings, respectively. Overall, DeepCCG across all datasets and settings in class-incremental learning has an average performance improvement of 3.37%, which shows that the method performs well and with the results on task-incremental learning shows that DeepCCG is effective in both scenarios. The next best method was ER-ACE which was consistently the second best method in the class-incremental experiments. Also, it is interesting to note that DeepCCG performs better, in terms of performance improvement over the other methods, in the class-incremental experiments than the task-incremental experiments, achieving an average performance improvement of 1.62% in task-incremental learning and 3.37% in class incremental learning. This finding is surprising as in DeepCCG there is no interaction between the mean parameters of classes which do not appear together in a training task. Therefore, these non-interacting mean parameters could be close together, harming performance in class-incremental learning, but this does not seem to be the case in practice and improving upon this is a direction for future work.

The results on the shifting window setting show that current methods do not exploit the increased task overlap in the setting and hence do not utilize the increased ability to transfer knowledge between tasks. This is shown in the results as the performance of the methods are very similar between the shifting window and disjoint tasks settings; while, in the task-incremental experiments, the difference between the multi-task upper bound's performance and the methods is much larger in the shifting window setting than that for the disjoint tasks setting.[1] This suggests that in settings of increased task overlap, where between-task knowledge transfer is more beneficial, current methods do not achieve accuracies near to that which is possible.

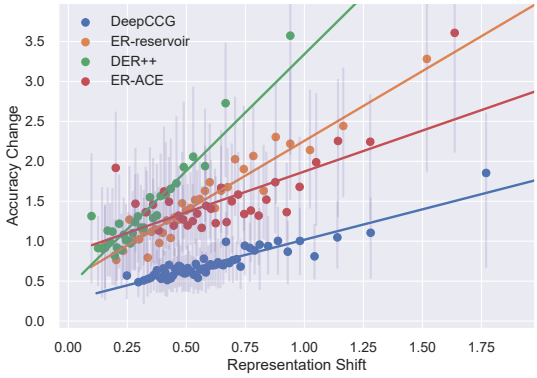

Figure 3: Binned scatter plot showing for the MiniImageNet task-incremental disjoint-tasks setting the change in accuracy against the mean change in representation after learning on a batch for the test data of the first task. The plot shows that for a given shift in representation the accuracy of DeepCCG changes the least.

Analysis of Representation Shift As part of our experiments we assess how well DeepCCG's classifier update adapts to representation shift compared to other methods, showing that DeepCCG's performance is more stable with respect to representation shift. To measure representation shift, after learning on the first task and for each incoming batch of data, we recorded the mean distance over the first task's test data between its representation before and after updating on the incoming batch. Additionally, we measure the change in accuracy on first task's test data before and after updating the method on the incoming batch. We present these results for task-incremental learning on MiniImageNet in Figure 3, displaying some of the best performing methods from our experiments. We also recorded results for other well performing methods but as they do not affect the conclusion of the experiment we do not include them in the figure to aid clarity; instead they are shown in Appendix F.2. Figure 3 shows

[1]For class-incremental experiments the upper bound does not use task identifiers so is identical for the shifting window and disjoint tasks settings.

Table 2: Results of performing an ablation on DeepCCG in the task-incremental learning scenario. We report the mean average accuracy with their standard errors across three independent runs. The results show that all components of DeepCCG are required for it to perform well.

	CIFAR-10		CIFAR-100		MiniImageNet	
Method	SW	DT	SW	DT	SW	DT
ER-reservoir	70.44 $_{0.81}$	66.71 $_{0.90}$	54.05 $_{0.63}$	58.31 $_{1.08}$	41.04 $_{1.54}$	40.29 $_{1.08}$
DeepCCG-reservoir	66.63 $_{1.47}$	63.97 $_{1.06}$	49.95 $_{0.743}$	54.76 $_{0.08}$	38.76 $_{0.65}$	38.14 $_{0.10}$
DeepCCG-standardHead	69.54 $_{1.10}$	63.99 $_{0.41}$	44.05 $_{0.264}$	49.66 $_{0.45}$	33.35 $_{1.78}$	36.76 $_{0.48}$
DeepCCG	**74.65** $_{\mathbf{2.00}}$	**69.29** $_{\mathbf{0.89}}$	**56.62** $_{\mathbf{0.29}}$	**60.46** $_{\mathbf{0.24}}$	**42.18** $_{\mathbf{0.45}}$	**43.04** $_{\mathbf{0.64}}$

that for the same amount of representation shift DeepCCG's accuracy changes less than the other methods, as its line of best fit is nearest to the horizontal axis and is the least steep. This demonstrates that the classifier of DeepCCG is more robust and better adapts to representation shift, validating DeepCCGs use of a Bayesian classifier which in one step adapts to representation shift.

Ablation Study One way to view DeepCCG is as a relative of ER-reservoir, where the methods differ in the sample section mechanism used and the type and learning of the probabilistic classifier and embedding function. Therefore, to analyse DeepCCG, we perform an ablation, creating two adaptations of our method, DeepCCG-reservoir and DeepCCG-standardHead, by changing components of DeepCCG to what they are in ER-reservoir. DeepCCG-reservoir is when we select examples to store in memory using reservoir sampling. DeepCCG-standardHead is when we replace the class-conditional Gaussian model with a standard output head (a fully connected layer, then a softmax) and learn the embedding function using the standard ER loss, while still using DeepCCG's sample selection method. The results for the ablation are presented in Table 2 and show that both DeepCCG-reservoir and DeepCCG-standardHead perform much worse than DeepCCG. Therefore, we have shown that all of DeepCCG's novel components are needed in conjunction for DeepCCG to perform well. We also perform an ablation on the size of memory in Appendix F.1, where we show that DeepCCG performs the best over all the memory sizes tested on.

6 CONCLUSIONS

In this work we have demonstrated that using an empirical Bayesian procedure, DeepCCG, for online continual learning (CL) is a promising approach. The key idea of DeepCCG is to adapt the classifier in one step to representation shift, by using a Bayesian class-conditional Gaussian classifier. The correct use of such a classifier means that there is never a misalignment between the classifier and representation. Therefore, the training signal is less noisy than when there is misalignment, which happens when using previous methods. This was demonstrated in our experiments on online CL, where we show that DeepCCG is more robust to representation shift and outperforms a range of online CL methods.

Acknowledgements

This work was kindly supported by ARM and EPSRC through an iCASE PhD scholarship.

References

Aljundi, R., Caccia, L., Belilovsky, E., Caccia, M., Lin, M., Charlin, L., and Tuytelaars, T. (2019a). Online Continual Learning with Maximal Interfered Retrieval. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, pages 11849–11860.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. (2019b). Gradient Based Sample Selection for Online Continual Learning. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, pages 11816–11825.

Antoniou, A., Patacchiola, M., Ochal, M., and Storkey, A. (2020). Defining Benchmarks for Continual Few-shot Learning. *arXiv preprint arXiv:2004.11967*.

Bang, J., Kim, H., Yoo, Y., Ha, J.-W., and Choi, J. (2021). Rainbow Memory: Continual Learning with a Memory of Diverse Samples. In *Proceedings of the 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8218–8227.

Buzzega, P., Boschini, M., Porrello, A., Abati, D., and Calderara, S. (2020). Dark Experience for General Continual Learning: a Strong, Simple Baseline. *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*, 33:15920–15930.

Caccia, L., Aljundi, R., Asadi, N., Tuytelaars, T., Pineau, J., and Belilovsky, E. (2021). New Insights

on Reducing Abrupt Representation Change in Online Continual Learning. In *Proceedings of the 10th International Conference on Learning Representations*.

Chaudhry, A., Khan, N., Dokania, P., and Torr, P. (2020). Continual Learning in Low-rank Orthogonal Subspaces. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *of the 34th Conference on the Advances in Neural Information Processing Systems*, volume 33, pages 9900–9911.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. (2019a). Efficient Lifelong Learning with A-GEM. In *Proceedings of the 7th International Conference on Learning Representations*.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H., and Ranzato, M. (2019b). On Tiny Episodic Memories in Continual Learning. *arXiv preprint arXiv:1902.10486*.

Delange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2021). A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Ebrahimi, S., Elhoseiny, M., Darrell, T., and Rohrbach, M. (2020). Uncertainty-guided Continual Learning with Bayesian Neural Networks. In *Proceedings of the 8th International Conference on Learning Representations*.

Farajtabar, M., Azizan, N., Mott, A., and Li, A. (2019). Orthogonal Gradient Descent for Continual Learning. *arXiv preprint arXiv:1910.07104*.

Farquhar, S. and Gal, Y. (2018). Towards Robust Evaluations of Continual Learning. *arXiv preprint arXiv:1805.09733*.

Farquhar, S. and Gal, Y. (2019). A Unifying Bayesian View of Continual Learning. *arXiv preprint arXiv:1902.06494*.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition.

Hayes, T. L. and Kanan, C. (2020). Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 220–221.

Henning, C., Cervera, M., D' Angelo, F., von Oswald, J., Traber, R., Ehret, B., Kobayashi, S., Grewe, B. F., and Sacramento, J. a. (2021). Posterior Meta-Replay for Continual Learning. In *Proceedings of the 35th conference on the Advances in Neural Information Processing Systems*, pages 14135–14149.

Huszár, F. (2018). Note on the Quadratic Penalties in Elastic Weight Consolidation. *Proceedings of the National Academy of Sciences*, 115(11):E2496–E2497.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456.

Kessler, S., Cobb, A., Rudner, T. G., Zohren, S., and Roberts, S. J. (2023). On Sequential Bayesian Inference for Continual Learning. *Entropy*, 25(6):884.

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

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. *Preprint*.

Kurle, R., Cseke, B., Klushyn, A., van der Smagt, P., and Günnemann, S. (2020). Continual Learning with Bayesian Neural Networks for Non-Stationary Data. In *Proceedings of the 8th International Conference on Learning Representations*.

Lotfi, S., Izmailov, P., Benton, G., Goldblum, M., and Wilson, A. G. (2022). Bayesian Model Selection, the Marginal Likelihood, and Generalization. *arXiv preprint arXiv:2202.11678*.

Lyu, Y., Wang, L., Zhang, X., Sun, Z., Su, H., Zhu, J., and Jing, L. (2023). Overcoming Recency Bias of Normalization Statistics in Continual Learning: Balance and Adaptation. In *Proceedings of the Thirty-seventh Conference on Neural Information Processing Systems*.

Mallya, A. and Lazebnik, S. (2018). PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773.

Mirzadeh, S. I., Farajtabar, M., Pascanu, R., and Ghasemzadeh, H. (2020). Understanding the Role of Training Regimes in Continual Learning. In *Proceedings of the 33rd conference on the Advances in Neural Information Processing Systems*, pages 7308–7320.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2018). Variational Continual Learning. In *International Conference on Learning Representations*.

Ostapenko, O., Lesort, T., Rodríguez, P., Arefin, M. R., Douillard, A., Rish, I., and Charlin, L.

(2022). Foundational models for continual learning: An empirical study of latent replay. *arXiv preprint arXiv:2205.00329*.

Prabhu, A., Torr, P. H., and Dokania, P. K. (2020). GDumb: A Simple Approach that Questions our Progress in Continual Learning. In *Procceding of the 16th European Conference on Computer Vision*, pages 524–540.

Rao, D., Visin, F., Rusu, A., Pascanu, R., Teh, Y. W., and Hadsell, R. (2019). Continual Unsupervised Representation Learning. In *Proceedings of the 33rd Conference on the Advances in Neural Information Processing Systems*.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. (2017). ICARL: Incremental Classifier and Representation Learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Sarfraz, F., Arani, E., and Zonooz, B. (2023). Error Sensitivity Modulation based Experience Replay: Mitigating Abrupt Representation Drift in Continual Learning. In *Proceedings of the Eleventh International Conference on Learning Representations*.

Shim, D., Mai, Z., Jeong, J., Sanner, S., Kim, H., and Jang, J. (2021). Online Class-Incremental Continual Learning with Adversarial Shapley Value. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

Sun, S., Calandriello, D., Hu, H., Li, A., and Titsias, M. (2022). Information-Theoretic Online Memory Selection for Continual Learning. In *Proceedings of the 10th International Conference on Learning Representations*.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2016). Instance Normalization: the Missing Ingredient for Fast Stylization. *arXiv preprint arXiv:1607.08022*.

van de Ven, G. M. and Tolias, A. S. (2019). Three Scenarios for Continual Learning. *arXiv preprint arXiv:1904.07734*.

Vinyals, O., Blundell, C., Lillicrap, T., and Wierstra, D. (2016). Matching Networks for One Shot Learning. In *Proceedings of the 30th Conference on the Advances in neural information processing systems*.

Vitter, J. S. (1985). Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57.

Wiewel, F. and Yang, B. (2021). Entropy-Based Sample Selection for Online Continual Learning. In *Proceedings of the 28th European Signal Processing Conference*, pages 1477–1481.

Wu, T., Caccia, M., Li, Z., Li, Y.-F., Qi, G., and Haffari, G. (2022). Pretrained Language Models in Continual Learning: A Comparative Study. In *Proceedings of the 10th International Conference on Learning Representations*.

Zeno, C., Golan, I., Hoffer, E., and Soudry, D. (2018). Task Agnostic Continual Learning Using Online Variational Bayes. *arXiv preprint arXiv:1803.10123*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. Yes
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes
   (b) Complete proofs of all theoretical results. Yes
   (c) Clear explanations of any assumptions. Yes

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes, all experiments were run using a single NVIDIA GeForce GTX 1050 GPU

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes
   (b) The license information of the assets, if applicable. Yes

(c) New assets either in the supplemental material or as a URL, if applicable. Yes

(d) Information about consent from data providers/curators. Not Applicable

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. Not Applicable

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

Appendices

A DETAILS OF LEARNING THE EMBEDDING FUNCTION

For DeepCCG, at time j, to compute the update to the embedding function $f_{\boldsymbol{\phi}_j}$, when given a batch B_j and a corresponding memory M_j, we proceed using the following steps. Firstly, we randomly sample $R_j \subset M_j$ of size r from memory. Then using only the other examples in memory, M_j/R_j, we compute the posterior density for each class mean— $\boldsymbol{\mu}_c$, with $c \in C$:

$$p(\boldsymbol{\mu}_c|M_j/R_j) = p(\boldsymbol{\mu}_c|M^Z_{j,c}/R^Z_j) \tag{13}$$

$$= \mathcal{N}\left(\boldsymbol{\mu}_y; \overline{M^Z_{j,c}/R^Z_j}, \frac{1}{m-r}\mathbf{I}\right), \tag{14}$$

where $M^Z_{j,c} = \{f_{\boldsymbol{\phi}_j}(\mathbf{x})|(\mathbf{x}, y) \in M_j \wedge y = c\}$ are the embeddings of points in the memory buffer with class y. R^Z_j is defined likewise and we use the notation $\overline{S}$ to denote the mean of the elements of a set S. Then we compute the posterior distribution of the embedded inputs $z \in B^Z_j \cup R^Z_j$ for each class $c \in C$ utilizing

$$p(\mathbf{z}|c, M_j/R_j) = p(\mathbf{z}|c, M^Z_{j,c}/R^Z_j) \tag{15}$$

$$= \int p(\mathbf{z}|c, \boldsymbol{\mu}_c)p(\boldsymbol{\mu}_c|M^Z_{j,c}/R^Z_j)d\boldsymbol{\mu}_c \tag{16}$$

$$= \mathcal{N}\left(\mathbf{z}; \overline{M^Z_{j,c}/R^Z_j}, (1 + \frac{1}{m-r})\mathbf{I}\right). \tag{17}$$

Next, we compute the posterior predictive for each example $(\mathbf{x}, y) \in B_j \cup R_j$ with a task identifier $t_{(\mathbf{x},y)}$, which is known for examples in the current batch and is stored by our method for examples stored in memory, and where $\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x})$ using

$$p(y|\mathbf{z}, t_{(\mathbf{x},y)}, M_j/R_j) = \frac{p(\mathbf{z}|y, M^Z_{j,y}/R^Z_j)p(y|t_{(\mathbf{x},y)})}{\sum_{c2C} p(\mathbf{z}|Y = c, M^Z_{j,c}/R^Z_j)p(Y = c|t_{(\mathbf{x},y)})} \tag{18}$$

$$= \frac{p(\mathbf{z}|y, M^Z_{j,y}/R^Z_j)}{\sum_{c2C_{t_{(\mathbf{x},y)}}} p(\mathbf{z}|Y = c, M^Z_{j,c}/R^Z_j)} \tag{19}$$

Finally, we update the embedding function by performing a gradient step using the formula

$$\boldsymbol{\phi}_{j+1} = \boldsymbol{\phi}_j + \eta\nabla_{\boldsymbol{\phi}} \sum_{(\mathbf{x},y)2B_j [R_j} \log(p(y|\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x}), t_{(\mathbf{x},y)}, M_j/R_j)), \tag{20}$$

which can be seen as a per-example log conditional marginal likelihood (Lotfi et al., 2022).

By using Eq. 17 and 19, the closed form of the posterior predictive distribution for a example $(\mathbf{x}, y)$ with task identifier $t_{(\mathbf{x},y)}$ and where $\mathbf{z} = f_{\boldsymbol{\phi}_j}(\mathbf{x})$ is

$$p(y|\mathbf{z}, t_{(\mathbf{x},y)}, M_j/R_j) = \frac{p(\mathbf{z}|y, M^Z_{j,y}/R^Z_j)}{\sum_{c2C_{t_{(\mathbf{x},y)}}} p(\mathbf{z}|Y = c, M^Z_{j,c}/R^Z_j)} \tag{21}$$

$$= \frac{\mathcal{N}\left(\mathbf{z}; \overline{M^Z_{j,y}/R^Z_j}, (1 + \frac{1}{m\ r})\mathbf{I}\right)}{\sum_{c2C_{t_{(\mathbf{x},y)}}} \mathcal{N}\left(\mathbf{z}; \overline{M^Z_{j,c}/R^Z_j}, (1 + \frac{1}{m\ r})\mathbf{I}\right)} \tag{22}$$

B PROOF OF ROBUSTNESS OF SAMPLE SELECTION MECHANISM TO i.i.d. REPRESENTATION SHIFT

We show that given a shift in representation that DeepCCG's sample selection mechanism maintains the property in expectation that is minimises the KL-Divergence between the posterior over the currently accessible data and the posterior induced by the examples selected to be store in memory. The proof is as follows: Let Z be the representation space and assume we are given a batch $B^Z_{j,y}$ and memory $M^Z_{j,y}$ of data points in Z for a given class y. We define the representations shift as $\mathbf{z} \ = \mathbf{z} + \boldsymbol{\epsilon}_\mathbf{z}$ where $\mathbf{z} \in B^Z_{j,y} \cup M^Z_{j,y}$ and $\boldsymbol{\epsilon}_\mathbf{z}$ is i.i.d. sampled from an arbitrary distribution with bounded mean and variance (i.e., $\mathsf{E}[\boldsymbol{\epsilon}_\mathbf{z}] < \infty$ and $\mathrm{Var}\,(\boldsymbol{\epsilon}_\mathbf{z}) < \infty$)). Furthermore, define $\boldsymbol{\eta}_\mathbf{z} = \boldsymbol{\epsilon}_\mathbf{z} - \mathsf{E}[\boldsymbol{\epsilon}_z]$, hence it is $\boldsymbol{\epsilon}$ shifted to have zero mean. Additionally, let $B^Z_{j,y} = \{\mathbf{z} \ = \mathbf{z} + \boldsymbol{\epsilon}_\mathbf{z} | \mathbf{z} \in B^Z_{j,y}\}$ and define $M^Z_{j,y}$ and M^{0Z}_y likewise. Therefore we have that,

$$\mathsf{E}[\mathrm{KL}(p(\boldsymbol{\mu}_y|D^Z_{j,y}, M^Z_{j,y})\|p(\boldsymbol{\mu}_y|M^{0Z}_y \ ))] = \mathsf{E}[\|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\ \|^2_2] \tag{23}$$

$$= \mathsf{E}[\|\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} \ 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \mathbf{z} \ - \frac{1}{|M^{0Z} \ |} \sum_{\mathbf{z} \ 2 M^{0Z}_y} \mathbf{z} \ \|^2_2] \tag{24}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} \ 2 B^Z_{j,y} \ [\ M^Z_{j,y}} z_k - \frac{1}{|M^{0Z} \ |} \sum_{\mathbf{z} \ 2 M^{0Z}_y} z_k)^2] \tag{25}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} (z_k + \mathsf{E}[\epsilon_{z,k}] + \eta_{z,k}) - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} (z_k + \mathsf{E}[\epsilon_{z,k}] + \eta_{z,k}))^2] \tag{26}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} (z_k + \eta_{z,k}) - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} (z_k + \eta_{z,k}) \tag{27}$$

$$+ (\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \mathsf{E}[\boldsymbol{\epsilon}_{z,k}] - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \mathsf{E}[\boldsymbol{\epsilon}_{z,k}]))^2] \tag{28}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} (z_k + \eta_{z,k}) - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} (z_k + \eta_{z,k}))^2] \ \ (\text{as } \epsilon_{z,k} \text{ are i.i.d.}) \tag{29}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} z_k - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} z_k \tag{30}$$

$$+ \frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \eta_{z,k} - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})^2] \tag{31}$$

$$= \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} z_k - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} z_k)^2] \tag{32}$$

$$+ 2\mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} z_k - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} z_k)(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \eta_{z,k} - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})] \tag{33}$$

$$+ \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \eta_{z,k} - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})^2] \tag{34}$$

$$= \sum_{k=0}^{d} (\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} z_k - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} z_k)^2 \tag{35}$$

$$+ \sum_{k=0}^{d} \mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} \ [\ M^Z_{j,y}} \eta_{z,k} - \frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})^2] \ \ (\text{as } \mathsf{E}[\eta_{z,k}] = 0) \tag{36}$$

$$= \|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\|_2^2 + \sum_{k=0}^{d} \Bigg(\mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} [M^Z_{j,y}} \eta_{z,k})^2] + \mathsf{E}[(\frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})^2] \tag{37}$$

$$- 2\mathsf{E}[(\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{\mathbf{z} 2 B^Z_{j,y} [M^Z_{j,y}} \eta_{z,k})(\frac{1}{|M^{0Z}_y|} \sum_{\mathbf{z} 2 M^{0Z}_y} \eta_{z,k})] \Bigg) \tag{38}$$

$$= \|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\|_2^2 + \sum_{k=0}^{d} \Bigg(\frac{\text{Var}\left(\epsilon_k\right)}{|B^Z_{j,y} \cup M^Z_{j,y}|} + \frac{\text{Var}\left(\epsilon_k\right)}{|M^{0Z}_y|} \tag{39}$$

$$- \frac{2}{|B^Z_{j,y} \cup M^Z_{j,y}||M^{0Z}_y|} \sum_{\mathbf{z} 2 B^Z_{j,y} [M^Z_{j,y}} \sum_{\mathbf{z}^0 2 M^{0Z}_y} \mathsf{E}[\eta_{z,k}\eta_{z^0,k}] \Bigg) \quad \text{(using formula for variance of means)} \tag{40}$$

$$= \|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\|_2^2 + (\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} + \frac{1}{|M^{0Z}_y|}) \sum_{k=0}^{d} \text{Var}\left(\epsilon_k\right) \tag{41}$$

$$- \frac{2}{|B^Z_{j,y} \cup M^Z_{j,y}||M^{0Z}_y|} \sum_{k=0}^{d} \sum_{\mathbf{z} 2 M^{0Z}_y} \mathsf{E}[\eta^2_{z,k}] \tag{42}$$

$$= \|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\|_2^2 + (\frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|} + \frac{1}{|M^{0Z}_y|}) \sum_{k=0}^{d} \text{Var}\left(\epsilon_k\right) - \frac{2}{|B^Z_{j,y} \cup M^Z_{j,y}|} \sum_{k=0}^{d} \text{Var}\left(\epsilon_k\right) \tag{43}$$

$$= \|\overline{B^Z_{j,y} \cup M^Z_{j,y}} - \overline{M^{0Z}_y}\|_2^2 + (\frac{1}{|M^{0Z}_y|} - \frac{1}{|B^Z_{j,y} \cup M^Z_{j,y}|}) \sum_{k=0}^{d} \text{Var}\left(\epsilon_k\right) \tag{44}$$

which completes the proof and where in the main paper we define $\nu = \sum_{k=0}^{d} \text{Var}\left(\epsilon_k\right)$.

C TABLE OF CONTINUAL LEARNING TERMINOLOGY

Table 3: Definition of continual learning terms used in the text.

Term	Definition
Online continual learning	A continual learning setup where the learner sees the data batch by batch and cannot revisit previous batches. Each batch is generated by a task $t \in T$, and when training the learner is told what task that is.
Task	A task $t \in T$ is a data generator, which generates data (i.e. data instances $\mathbf{x} \in X$ and labels $y \in C$) from a given subset of the classes $C_t \subseteq C$.
Task identifier	States for some example $(\mathbf{x}, y)$ (or batch of examples) what task it was generated from.
Disjoint tasks setting	A setting where no two tasks generate data from the same class. In other words for any two task t and t^0 we have that $C_t \cap C_{t^0} = \emptyset$.
Shifting window setting	A setting where we fix an ordering of the classes $c_1, \ldots, c_k$ and define the ith task to generate data from the classes $c_i, \ldots, c_{i+l}$, where l is a free parameter of setting called the window length.
Task-incremental learning	A scenario where at test time the learner has access to the task identifiers of the data instances in the test set.
Class-incremental learning	A scenario where at test time the learner does not have access to task identifiers and so classifies across all tasks/classes seen.

D DESCRIPTION OF METHODS USED FOR COMPARISON

In our experiements we compared DeepCCG to the following continual learning methods: EWC (Huszár, 2018; Kirkpatrick et al., 2017), PackNet (Mallya and Lazebnik, 2018), ER-Reservoir (Chaudhry et al., 2020), A-GEM (Chaudhry et al., 2019a), EntropySS (Wiewel and Yang, 2021), GSS (Aljundi et al., 2019b), ER-ACE (Caccia et al., 2021), DER++ (Buzzega et al., 2020), ESMER (Sarfraz et al., 2023) and iCaRL (Rebuffi et al., 2017). Below we give a short description of each method.

EWC is a parameter regularisation approach which uses an l_2 regulariser to ensure the parameters do not move to far away from the weighted average of the parameters fitted for the previous tasks. Additionally, this l_2 regulariser is weighted using the Fisher information of the fitted parameters on previous tasks.

PackNet is a parameter isolation method which freezes a certain proportion of the filters/nodes after learning on a task. This means that for future tasks learning takes place only on the remaining part of the neural network which is not frozen, preventing the forgetting of the subnetworks used for the previous tasks. To select which filters are frozen for a task PackNet uses a filter pruning method.

ER-Reservoir is the standard replay method. It stores previously seen examples in a replay buffer and in training "replays" a random batch of the stored examples by training on them along side the examples of the current batch in the data stream. Also, it selects what examples to store in memory using reservoir sampling (Vitter, 1985).

A-GEM is a replay method which prevents the average loss on examples stored in the memory buffer from increasing. This is achieved by projecting the proposed weight update vector (i.e. the gradient of the loss w.r.t. the weights), such that the above constraint is satisfied while minimally changing the update vector.

EntropySS replaces the sample selection method of ER-Reservoir with one which chooses which examples to store in memory by trying to ensure the entropy of the stored examples is maximised.

GSS replaces the sample selection method of ER-Reservoir with one which chooses which examples to store in memory by maximising the variance of gradient values of the loss for the stored examples.

ER-ACE is a replay method which uses pseudo task-identifiers such that previous tasks are seen to be merged into a single task.

DER++ is a replay method which in addition to using standard replay adds another regularisation term which prevents the logits of previous examples straying from the values they took when they were given to the network for the first time.

ESMER is a replay method which uses knowledge distillation to an exponentially moving average of the network to regularise updates.

iCaRL is a replay method which uses a distillation regulariser and per-class sigmoids when learning. Additionally, at test time it uses a nearest means classifier, using the examples stored in memory to form the per-class means. Importantly, we use the adaptation of iCaRL given in Buzzega et al. (2020), such that it can be used in task incremental learning.

E ADDITIONAL EXPERIMENTAL DETAILS

[2]In addition to the experimental details mentioned in the main body of the paper, there are a few more details to mention. First, for methods with hyperparameters we performed a grid search using the same experimental set up as the real experiments and 10% of the training data as validation data. This led to the selection of the hyperparameters of EWC and PackNet shown in Tables 4 and 5. For DER++ and ESMER we tune the hyperparams on CIFAR-100 and then use the same hyperparams across all datasets. The hyperparameters selected for DER++ were $\alpha = 0.5$ and $\beta = 1.0$; and for ESMER were $\alpha_{EMA} = 0.999$, $\alpha_l = 0.99$, $\beta = 2.5$, $\gamma = 0.2$ and $r = 0.7$. Second, we use the same learning rate of 0.1 and the standard gradient decent optimiser

[2]Code for DeepCCG is available at https://github.com/Tlee43/DeepCCG.

for all methods. These were chosen by looking at the commonly selected values in previous work (Mirzadeh et al., 2020) and were shown to be performative for all methods tested. Third, for the multi-task upper bound we train using two epochs in the task-incremental scenario and three epochs in the class-incremental scenario, to more fairly upper bound the performance of the methods. Last, as ESMER stores both an exponential moving average of the model and examples in memory, we ensure it uses roughly the same amount of memory as other methods by reducing the number of examples it stores. This means that for experiments on CIFAR-100 and MiniImageNet we reduce ESMER's memory size by 400 examples (the rough number of examples that take up the same space as a copy of the model) and we do not reduce the memory size for CIFAR-10 as it is already less than 400 examples.

Table 4: Values selected for the hyperparameters of EWC and PackNet in the task-incremental learning experiments, which are the regularisation coefficient and the percentage of available filters to be used per task, respectively. SW stands for shifting window and DT stands for disjoint tasks.

	CIFAR-10		CIFAR-100		MiniImageNet	
Method	SW	DT	SW	DT	SW	DT
EWC	1	6	6	6	2	9
PackNet	0.3	0.3	0.1	0.1	0.05	0.2

Table 5: Values selected for the hyperparameters of EWC in the class-incremental learning experiments, which is the regularisation coefficient. SW stands for shifting window and DT stands for disjoint tasks.

	CIFAR-10		CIFAR-100		MiniImageNet	
Method	SW	DT	SW	DT	SW	DT
EWC	1	1	6	1	6	1

F ADDITIONAL RESULTS

F.1 Results on Using Different Memory Sizes

Table 6: Results of experiments looking at the effect of memory buffer size m for the replay methods tested, using the shifting window setting on CIFAR-100 in task-incremental learning scenario. We report mean average accuracy with their standard errors across three independent runs.

Method	m=750	m=1000	m=1250
ER-reservoir	53.17 $_{0.656}$	54.05 $_{0.626}$	55.22 $_{0.592}$
A-GEM	27.18 $_{1.091}$	29.01 $_{1.449}$	27.87 $_{0.172}$
EntropySS	50.52 $_{0.725}$	51.80 $_{0.700}$	53.34 $_{0.372}$
GSS	47.15 $_{0.766}$	48.20 $_{0.332}$	46.18 $_{0.341}$
DeepCCG (ours)	**53.42** $_{\mathbf{0.460}}$	**56.62** $_{\mathbf{0.288}}$	**57.98** $_{\mathbf{0.514}}$

One additional useful experiment is looking at the relationship between performance and the size of memory used for DeepCCG. Therefore, in Table 6 we show the performance of DeepCCG and other replay methods compared against with varying memory size. Table 6 shows that when the memory size is increased to $m = 1250$, DeepCCG has an improvement in average accuracy relative to other methods, as it achieves 2.76% better than any other method for $m = 1250$, while for $m = 1000$ it achieves 2.57% better than any other method, a 0.19% improvement. When the memory size is decreased to $m = 750$, we see that DeepCCG's performance drops more than other methods as it is only 0.25% better than other methods in this case. Therefore, our experiments show that compared to other replay methods DeepCCG's performance increases the most when m increases and that for small buffer sizes DeepCCG performs less well, potentially due to the fact that the examples in the memory buffer are used to infer the posterior over the means of the per-class Gaussians and so the method needs a given

amount of examples to specify the means well. We also note that in our experiments the performance ranking of the methods does not change with m.

F.2 Additional Representation Shift Results

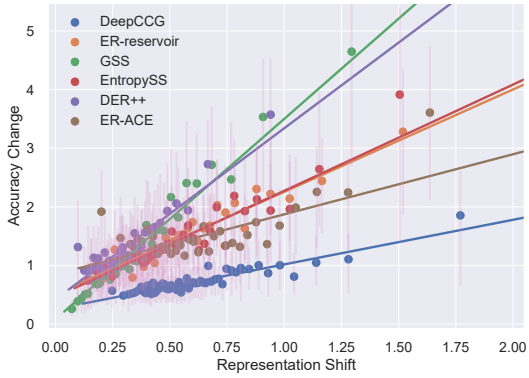

Figure 4: Binned scatter plot showing for the MiniImageNet disjoint tasks setting the change in accuracy against the mean change in representation after learning on a batch for the test data of the first task.