Accepted Manuscript
To appear at the 17th IEEE International Conference on Automatic Face and Gesture Recognition (FG), 2023.

Latent Generative Replay for Resource-Efficient Continual Learning of Facial Expressions

Samuil Stoychev, Nikhil Churamani and Hatice Gunes
Department of Computer Science and Technology, University of Cambridge, United Kingdom
ss2719@cantab.ac.uk, {nikhil.churamani,hatice.gunes}@cl.cam.ac.uk

Abstract— **Real-world Facial Expression Recognition (FER) systems require models to constantly learn and adapt with novel data. Traditional Machine Learning (ML) approaches struggle to adapt to such dynamics as models need to be re-trained from scratch with a combination of both old and new data. Replay-based Continual Learning (CL) provides a solution to this problem, either by storing previously seen data samples in memory, sampling and interleaving them with novel data (*rehearsal*) or by using a generative model to simulate *pseudo-samples* to replay past knowledge (*pseudo-rehearsal*). Yet, the high memory footprint of *rehearsal* and the high computational cost of *pseudo-rehearsal* limit the real-world application of such methods, especially on resource-constrained devices. To address this, we propose Latent Generative Replay (LGR) for pseudo-rehearsal of low-dimensional latent features to mitigate forgetting in a resource-efficient manner. We adapt popular CL strategies to use LGR instead of generating pseudo-samples, resulting in performance upgrades when evaluated on the CK+, RAF-DB and AffectNet FER benchmarks where LGR significantly reduces the memory and resource consumption of replay-based CL without compromising model performance.**

I. Introduction

Recent years have seen an increased application of ML algorithms on resource-constrained devices such as mobile phones, robots and Internet of Things (IoT) devices [1], [2]. This is particularly true for camera-based applications for analysing human faces [3], [4], [5]. However, deploying these models on such devices remains a challenge due to the high resource consumption of deep learning frameworks, which are often designed to run on server-class GPUs [6]. Moreover, deep learning's high computational cost has a detrimental impact on the environment with initiatives such as "Green AI" [7] calling for a reduction in the energy consumption and carbon footprint associated with deep learning.

Real-world application of ML-based Facial Expression Recognition (FER) requires models to continuously adapt to novel information, integrating this new information *on-the-fly*, by retraining (partially or from scratch) with this additional data. This may mean learning new expression categories that are not part of the model's training (**Problem 1**). For example, a model may be trained to classify the 'universal expressions' [8] but now needs to learn secondary or compound expressions [9], outside the training distribution. However, as past data may not always be available, adapting to novel information can result in the model *overwriting* past knowledge, leading to *catastrophic forgetting* [10]. Additionally, gradient-based learning requires data samples to be *independently* and *identically drawn* (*i.i.d*). However, this assumption is violated in real-world conditions [11], [12], [13] where data is available only incrementally.

Lifelong or Continual Learning (CL) [14], [15] addresses this problem by enabling models to adapt with sequentially made available streams of data, incrementally learning novel information while preserving past knowledge. This may be achieved by regularising model updates to discourage rapid and catastrophic changes in model parameters [16], [17] or using Bayesian principles to infer task-conditioned parameter distributions directly from the data [18] and rehearsing task-specific weights using hypernetworks [19] or by periodically replaying past knowledge using rehearsal [20] or pseudo-rehearsal [21], interleaving it with novel data to simulate *i.i.d* settings. Both regularisation and replay-based CL has proven to be effective for incremental and sequential learning of information [15], however, they incur additional computational and memory costs (**Problem 2**), making their application on resource-constrained devices problematic [22]. Yet, with CL algorithms being increasingly deployed on such devices [23], [24], several studies point out the importance of considering the resource consumption of CL strategies [22], [25].

In this work, we propose *Latent Generative Replay (LGR)* to reduce the memory and resource consumption of replay-based CL. LGR uses a relatively simpler generative model to learn low-dimensional *feature representations* enabling replay 'from the middle' of the network, instead of learning to simulate high-dimensional data, incrementally learning FER categories (**Solution 1**). Since intermediate features are significantly smaller compared to input samples, LGR reduces the overall memory and resource consumption of the model (**Solution 2**). To demonstrate the resource-efficiency of LGR, we adapt Deep Generative Replay (DGR) [26]-based approaches by replacing their generative models used for pseudo-replay of input data with a simpler generative model for LGR (**Contribution 1**). We evaluate LGR across 3 popular FER benchmarks, namely, the CK+ [27], RAF-DB [28] and AffectNet [29] datasets. Our experiments show that LGR significantly improves upon the resource-efficiency of DGR-based methods (**Contribution 2**). Furthermore, it provides improvements over rehearsal strategies such as Naïve Rehearsal (NR) [30] and Latent Replay (LR) [31] on several metrics (**Contribution 3**), encouraging the use LGR-based pseudo-rehearsal for resource-efficient CL.

N. Churamani is funded by the EPSRC grant EP/R513180/1 (ref. 2107412). H. Gunes is supported by the EPSRC project ARoEQ under grant ref. EP/R030782/1.

II. Replay-based Continual Learning

Under CL settings, models encounter novel data one task at a time [32]. Thus, at any given time, models have access to data only from the current task (or class). As gradient-based learning methods adapt model parameters to 'fit' the training data, the model 'fine-tunes' itself towards only the current task and the past knowledge is *overwritten* [33]. A straightforward way to mitigate such *forgetting* is periodically replay and *rehearse* past information, interleaving it with novel data. This allows the models to balance novel *vs.* past learning as the model is trained on mixed batches of past and present data samples, alleviating *forgetting* of information. Replay-based CL strategies include physically storing data samples from previous tasks in memory buffers and regularly sampling from them (known as *rehearsal* [20]) mixing it with new data. Alternatively, a generative or probabilistic model is maintained that learns the inherent data statistics, enabling models to draw *pseudo-samples* to be replayed (known as *pseudo-rehearsal*) [21] along with novel data.

A. Rehearsal

Rehearsal-based CL methods [20], [15] maintain memory buffers to store data samples from previously seen tasks, enabling the model to rehearse past knowledge along with novel information. However, as physical copies of data need to be stored, this increases the memory footprint of models particularly when learning with high-dimensional images or if the number of tasks to be learnt increases significantly.

A simple strategy for rehearsal can be to fix the size of the memory buffer to be '*large enough*' and divide it equally amongst data from all the tasks, popularly known as Naïve Rehearsal (NR) [30]. However, as the number of tasks increases, the fraction of memory allocated to each task shrinks, resulting in fewer samples per task for rehearsal. Other more sophisticated strategies focus on prioritising replay [34], storing and replaying exemplars from each task to best approximate task means [35], [36] or applying reservoir sampling to fix a budget for each seen task [37].

Pellegrini *et al.* [31] propose an improved and more resource-efficient strategy by enabling Latent Replay (LR) of information. This works on the assumption that bottom layers of (deep) neural models essentially act as high-level feature extractors that change very little once the model is trained. Meanwhile, task-specific knowledge is encoded in the top layers of the network where the weights are less stable. They split the model into two parts; feature extractor layers (the *root*) and task-descriptive layers (the *top*) with a *latent replay layer* separating them. Instead of storing high-dimensional samples from previously seen tasks, they store *latent layer* features in the memory buffer for different tasks. Although this reduces the memory footprint of traditional rehearsal-based CL, budgeting the memory buffer appropriately to store and rehearse samples for each task remains a challenge.

B. Generative Replay

Even though rehearsal-based methods are relatively simple to implement and offer an effective solution towards mitigating *catastrophic forgetting* [38], they require allocating large chunks of memory for rehearsal. Storing data samples can also be problematic and impractical owing to security or compliance reasons, for instance, when storing facial images of several individuals. Thus, to avoid maintaining memory buffers altogether, whether storing actual data samples or latent representations, *pseudo-rehearsal* or *generative replay* may be used to learn the inherent data statistics and draw *pseudo-samples* [21] as and when needed, to be replayed to the model. Since no actual data needs to be stored in memory buffers, this reduces the memory cost of replay-based CL.

Recent advances in generative models [39], [40], particularly in their ability to generate high-quality data samples, have greatly enhanced the potential of generative replay-based methods [26], [41], [42], [43].

C. Generative Replay of Features

Despite generative models allowing for effective *pseudo-rehearsal* of information while significantly bringing down the memory cost of replay-based CL, they become hard to train as the number of tasks increases. Apart from learning the tasks, the generator needs to learn to reconstruct high-quality and discriminative samples for previously seen tasks. This adds significant computational expense to the model and stabilising learning for both the solver and the generator may become intractable for a large number of tasks. To address this, recent methods focus on investigating replaying low-dimensional feature representations, instead of high-dimensional input patterns [44], [45].

Xiang *et al.* [46] propose a GAN-based solution which consists of three networks: a pre-trained sub-net for extracting spatial feature maps, a generator that reconstructs these feature maps and a discriminator for classification and discrimination of real *vs.* synthetic features. Despite resolving the problem of reconstructing high-dimensional data by learning to generate spatial feature maps that 'use the statistics of feature embeddings', fine-tuning network layers for both the generator and the discriminator is complex and resource-intensive. Liu *et al.* [47], on the other hand, propose a hybrid model combining generative replay and feature distillation. For each task, a feature extractor and a classifier learn to solve the task while a generator learns the distribution of extracted features. For the next task, a distillation loss is used to distil discriminative features from the previously seen tasks to the new feature extractor while the previously trained generator provides pseudo-samples to rehearse past knowledge at the feature level. Van de Ven *et al.* [48] propose a brain-inspired solution merging a generator into the main learning model by equipping it with generative feedback connections. Their Replay through Feedback (RtF) approach is implemented as a symmetric auto-encoder model equipped with a *softmax* layer resulting in only a single model to be trained. They also propose a modification as 'internal replay' that focuses on replaying learnt 'hidden' representations generated by the "network's own, context-modulated feedback connections", instead of pseudo-samples of data, to alleviate forgetting.

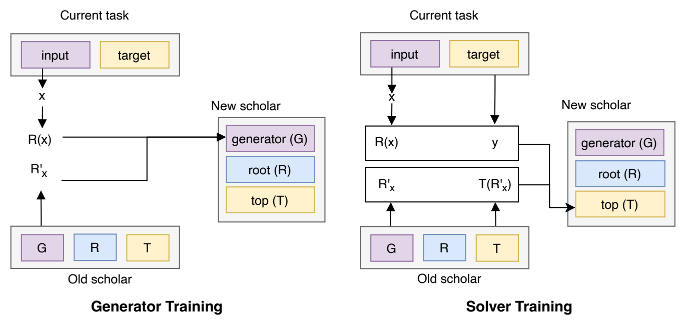

Fig. 1. Latent Generative Replay (LGR). $\boldsymbol{G}$, $\boldsymbol{R}$ and $\boldsymbol{T}$ denote the *generator*, *root* and *top* of the scholar, respectively. $\boldsymbol{R(\cdot)}$ and $\boldsymbol{T(\cdot)}$ indicate the output of the *root* and *top* and $\boldsymbol{R'_x}$ denotes the reconstructed latent representations. Adapted from [26].

III. Latent Generative Replay

Replaying network-extracted latent representations can be functionally equivalent to using high-dimensional input patterns under the assumption that the bottom (feature extractor) layers of the model are pre-trained and change very little [31]. This reduces the memory-consumption of the replay-based models by storing 'lighter' latent representations in the memory buffer instead of high-dimensional data samples. Van de Ven *et al.* [48], on the other hand, not only merge a generative model within the main learning model but also use 'internal replay' of 'hidden' representations generated by the network's own feedback connections, eliminating the need for maintaining a memory buffer completely.

Inspired by the conclusions from these works, we propose the *Latent Generative Replay (LGR)* approach as a resource-efficient pseudo-rehearsal strategy that combines the benefits of generative replay with using low-dimensional latent representations, reducing both memory and computational expenses. We base LGR under the same assumption [31] that once feature extraction layers of the model are trained sufficiently and relatively *frozen*, the extracted feature representations can be used effectively to rehearse past knowledge. This is different from other works towards generative replay of features as these methods focus on training both the feature extractor layers (or the discriminator to generate normalised embeddings) and the generator simultaneously. Furthermore, these methods are not focused towards optimising the resource-efficiency of CL necessarily, and, instead, focus on model performance as the only criteria, limiting their real-world application on resource-constrained devices.

For our implementation, we adapt the *scholar*-based architecture of the Deep Generative Replay (DGR) [26] approach to use LGR instead. DGR maintains a *scholar* consisting of a *solver* (the classifier model) and a *generator* (to draw pseudo-samples of past data). The *scholar* is progressively updated after each task, using the knowledge from previous tasks and integrating novel learning. We split the *scholar* into three components: (a) the *generator* (G) to reconstruct and replay *latent representations*, (b) the pre-trained and frozen *root* (R) for the *solver* to extract task-independent latent representations, and (c) the *top* (T) for the *solver* that learns task-discriminative information (see Figure 1). This is different from the DGR approach where the scholar consists of a single, combined solver that learns task-discriminative information while its generator is used to reconstruct high-dimensional pseudo-samples representing the input data.

The training process for LGR (see Algorithm 1) is split into two steps. Firstly, for training the generator (G), input data from the current task x is processed and latent representations $R(x)$ are extracted using the pre-trained *root*. $R(x)$ is then interleaved with generated latent patterns R'_x for all previously seen tasks and used to update the generator. Training G after each task on both $R(x)$ and R'_x (only $R(x)$ is used for Task 1) ensures that the updated generator encodes both new and old tasks and can be used to simulate pseudo-samples for both in the future. Pseudo-samples R'_x generated for the previous tasks are also passed through the *top* of the solver to obtain labels $T(R'_x)$ for them. Once the generator is updated, the *top* of the *solver* is updated to learn the new task. For this, the current task data $\langle R(x), y\rangle$ is combined with the labelled *latent* pseudo-samples $\langle R'_x, T(R'_x)\rangle$ generated for previously seen tasks. We modify the original DGR algorithm [26] to use root-extracted latent representations (*line* 5) and train the new scholar (the top T and generator G) using latent representations instead (*lines* $13-16$). G_{old} and T_{old} correspond to the *generator* and *top* for the old model and G_{new} and T_{new} represent the updated models. The coefficient r defines the fraction of real data to be mixed with the pseudo-samples for training.

LGR is designed to reduce the memory and computational cost of replay-based CL on two accounts: firstly, instead of allocating memory resources to store data samples, we use a 'lightweight' generative model to reconstruct low-dimensional latent features for pseudo-rehearsal. Secondly, as the lower layers (the *root*) of the model are frozen, and only the *top* is updated to learn task-relevant information, the computational expense of learning is reduced.

Algorithm 1 Latent Generative Replay

```
1:  $G_{old}, T_{old} \leftarrow \varnothing$
2:  $R \leftarrow$ pre-trained root or feature extractor
3:  for each task $i$ do
4:      $\langle x_i, y_i\rangle \leftarrow$ inputs and targets from current task distribution $D_i$
5:      $R_x \leftarrow R(x_i)$ // Root-extracted latent features.
6:      Initialise $G_{new}$ and $T_{new}$
7:      if $G_{old} \neq \varnothing$ then
8:          $R'_x \leftarrow$ generate synthetic data with $G_{old}$
9:          $y' \leftarrow T_{old}(R'_x)$
10:     else
11:         $R'_x, y' \leftarrow \{\}$ // No replay data for Task 1.
12:     end if
13:     Generator Training:
14:         Train $\boldsymbol{G_{new}}$ on $\boldsymbol{x' \in (R_x \cup R'_x)}$
15:             $L(\phi,\psi) = -\mathbb{E}_{z\sim q_\phi(z|x')}[log(p_\psi(x'|z))] + \mathbb{KL}(q_\phi(z|x') \parallel p(z))$
16:     Solver (Top) Training:
17:         Train $\boldsymbol{T_{new}}$ on $\boldsymbol{\langle R_x, y_i\rangle \cup \langle R'_x, y'\rangle}$
18:             $L(\theta_i) = r\mathbb{E}_{(R_x,y_i)\sim D_i}[L(T(R_x;\theta_i),y_i)] + (1-r)\mathbb{E}_{R'_x\sim G_{old}}[L(T(R'_x;\theta_i),y')]$
19:     $G_{old} \leftarrow G_{new}$
20:     $T_{old} \leftarrow T_{new}$
21: end for
```

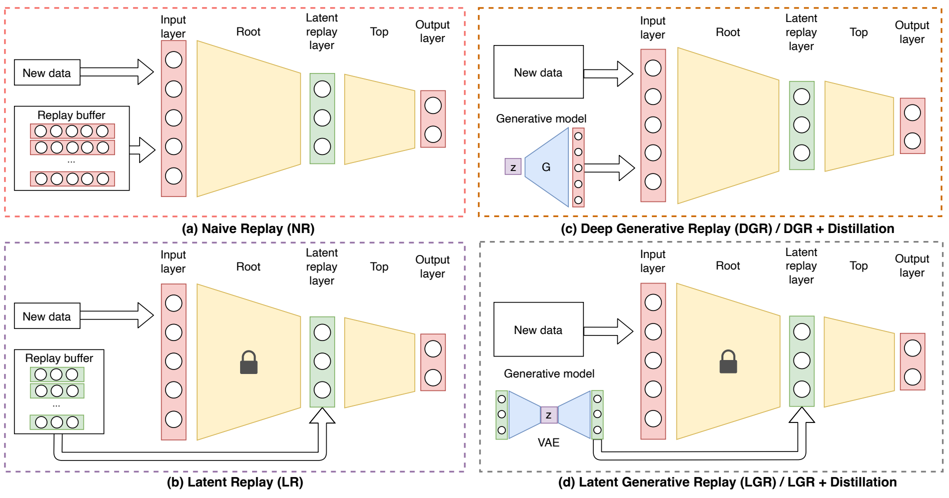

Fig. 2. **Compared Approaches:** (a) Naïve Rehearsal (NR) [30], (b) Latent Replay (LR) [31], (c) Deep Generative Replay (DGR) [26] and DGR+d [49], and (d) Latent Generative Replay (LGR) applied to DGR and DGR+d for replaying low-dimensional latent representations.

IV. Evaluation

We evaluate LGR, both for model performance and resource-efficiency, across several metrics comparing it with other popular replay-based CL methods.

A. Learning Scenario

We conduct our experiments under Task Incremental Learning (Task-IL) settings [32] to evaluate the resource-efficiency of replay-based CL where models are expected to learn one task at a time with each task consisting of two classes. We split the datasets by randomly shuffling the *classes* and grouping them into *tasks*.

B. Datasets

We set-up two criteria for selecting the different benchmarks; complexity and the scale of the dataset and select 3 different FER benchmark datasets: CK+ [27], RAF-DB [28], and AffectNet [29], enabling a comprehensive analysis on a variety of data settings. CK+ evaluates the model under relatively simpler (*lab-controlled*) data settings on a smaller scale as we select the last 3 frames from each video to represent the corresponding expression category and the first frame to represent *neutral* [50]. RAF-DB, on the other hand, evaluates the model under complex *in-the-wild* data settings however with a relatively lower number ($\approx 15K$) of samples while AffectNet represents large scale learning under complex *in-the-wild* data settings. We use the *downsampled* split for AffectNet [29] for our comparisons consisting of $\approx 90K$ training samples and $4K$ test samples. Both CK+ and AffectNet consist of 8 classes, namely, *anger, happiness, sadness, surprise, disgust, fear, contempt* and *neutral* while Real-world Affective Faces Database (RAF-DB) consists of only 7 classes as *contempt* samples are not available. The default train-test (or train-validation) split is used for RAF-DB and AffectNet while for CK+ we perform a cross-subject split, assigning 86 subjects to the train set and 37 to the test dataset, resulting in a $70:30$ data-split while maintaining the data distributions.

C. Compared Approaches

We compare the proposed LGR approach with the DGR [26] approach which uses generative replay of high-dimensional input images. A simple extension to DGR is proposed by [49] to improve its ability to mitigate *forgetting* where, instead of using 'hard targets' (using the *argmax* label), the model is trained with 'soft targets', directly using the *softmax* output for a more fine-grained learning process for the scholar, improving its predictive performance. We also adapt this DGR *with Distillation* (DGR+d) [49] approach to use LGR (LGR+d) for a fair comparison. For both DGR and DGR+d, the *solver* is split into two parts; the *root* and the *top*. While the *root* is pre-trained and frozen, the *top* of the model is trained to learn novel tasks. We also include the Naïve Rehearsal (NR) [30] and Latent Replay (LR) [31] approaches in our evaluations as they rely on explicitly storing previously seen training data and periodically replaying them to mitigate forgetting. Figure 2 presents a schematic understanding of the compared approaches.

D. Experiment Settings

1) Evaluation Metrics: We compare the different CL methods on their resource-efficient performance across several evaluation metrics, adapting the implementations presented in [51]. Real-world applications of FER on resource-constrained devices such as robots, require them to be able to perform computations under limited storage and processing capabilities. As a result, achieving higher model accuracy while incurring high memory and compute cost may not be desirable. We evaluate the different approaches across several dimensions, measuring model performance, in terms of classification accuracy as well as memory and resource-efficiency of CL methods. All model results are run for 3 repetitions and average results are reported for each dataset. ***Average Accuracy (%):*** Average accuracy is calculated as the mean test-set accuracy across all tasks at the end of learning. $Acc_{avg} = \sum_{i=1}^{n} \frac{acc(T_i)}{n}$, where $acc(T_i)$ is the test accuracy on i^{th} task and n is the number of tasks.

TABLE I

SELECTED HYPER-PARAMETERS FOR THE THREE DATASETS.

Method	Model Hyper-parameters	
NR [30]	$B_{size} = 1500$	
LR [31]	$B_{size} = 1000$	
DGR [26]	$G_{FC} = 1600$	$G_{OUT} = 30,000$
DGR+d [49]	$G_{FC} = 1600$	$G_{OUT} = 30,000$
LGR (Ours)	$G_{FC} = 200$	$G_{OUT} = 4096$
LGR+d (Ours)	$G_{FC} = 200$	$G_{OUT} = 4096$

Training Time (s): Training time is computed as the *absolute time elapsed* (in seconds) during training.

RAM Consumption (MB): We place multiple checkpoints throughout the code to evaluate the amount of RAM (MB) allocated to the running process. RAM consumption is computed as the peak RAM usage (max usage during training).

GPU Consumption (MB): We use Nvidia's `nvidia-smi` tool, similar to [52], [53], to query GPU memory usage, every second, and calculate the *Peak GPU Memory Consumption* (in MB) by the model during training.

CPU Usage (s): CPU usage is measured as the CPU time (in seconds) allocated to the training process. This is calculated as the difference between the CPU time at the end *vs.* at the beginning of the training process.

GPU Usage (%): GPU usage is also calculated using Nvidia's `nvidia-smi` tool recording the GPU usage every second. GPU usage is reported as the % of the GPU allocated for training process. GPU usage values are also logged at different intervals during the training process but *average* GPU usage is reported (different from peak consumption in MB), calculated over the training process.

2) Implementation Details: For the different CL approaches compared, we use a *VGG-16*-based backbone [54], pre-trained on ImageNet, as the *root* to extract 4096-d latent featrues from ($100 \times 100 \times 3$) RGB input images. The VGG-16-based network is chosen after an ablation experiment (see section V-D) conducted with three other network backbones: MobileNet-V2 [55], ResNet-18 [56] and AlexNet [57] where the VGG-16 model is able to balance model performance across all performance metrics used for all the three datasets. For fairness of comparisons between the different approaches, roots are pre-trained for all the approaches ensuring that none of them incur an additional computational and memory cost of having to train the *root* from scratch. For implementing the generator, we use the Variational Autoencoder (VAE) [40] architecture.

All models are trained using the *Adam* optimiser ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The learning-rate is set to $1.0e^{-4}$ over 2000 iterations and the batch-size is set to 128 for all the datasets except for CK+ where it is set to 32 as some classes had fewer than 128 samples. Model hyper-parameters (see Table I) are set based on separate grid-searches for each model and selecting the best-performing values. B_{size} denotes the corresponding memory buffer-size for NR and LR, G_{FC} for DGR and LGR-based methods denotes the size of the fully-connected (dense) layers of the generator while G_{OUT} denotes the size of the reconstructed output of the corresponding generators. All models are implemented using the PyTorch Python Library adapting the code repository made available by [32]. To standardise the results, all experiments are conducted on a system with an Nvidia Quadro RTX 8000 GPU, an 8-core Intel Xeon Gold CPU @2.30GHz and 64 GB of RAM. The experiments are run individually to avoid potential interference in system metrics from other processes.

TABLE II

TASK-IL RESULTS ON CK+. BEST RESULTS FOR EACH ROW ARE HIGHLIGHTED IN **BOLD** WHILE SECOND BEST ARE IN [BRACES].

	NR [30]	**LR** [31]	**DGR** [26]	**LGR** (Ours)		**DGR+d** [49]	**LGR+d** (Ours)	
Accuracy (%) ▲	**97.29**	[*96.96*]	78.04	92.03	▲	80.30	92.93	▲
Training time (s) ▼	834.40	**741.17**	1164.33	[*827.80*]	▼	1,189.10	839.00	▼
RAM Usage (MB) ▼	2774.88	2805.65	**2758.76**	2807.11	▲	[*2759.77*]	2806.54	▲
CPU Usage (s) ▼	927.78	**834.23**	1259.08	[*924.10*]	▼	1287.63	937.37	▼
GPU Usage (%) ▼	29.42	19.72	44.70	[*19.07*]	▼	43.73	**19.02**	▼
GPU Memory (MB) ▼	4816.67	[*2556.00*]	6820.00	**2354.00**	▼	6820.00	**2354.00**	▼

V. RESULTS

For our evaluations, we split the compared approaches into 3 different sections; rehearsal based NR and LR, DGR using *hard targets* and its LGR adaptation and finally, DGR and LGR using *soft targets* (that is, DGR+d and LGR+d).

A. CK+ Results

Table II presents the results on the Extended Cohn-Kanade (CK+) dataset, where the highest accuracy is reported by the NR approach while LR achieves second best results. Despite following the Task-IL protocol, these results improve upon the state-of-the-art for CK+ which is at 96.8% [58]. Since CK+ is a relatively smaller dataset ($\approx$ 1300 samples), almost the entire dataset can be held in the memory buffers for NR ($B_{size} = 1500$) and LR ($B_{size} = 1000$). As a result, these methods are able to mitigate forgetting successfully. LGR-based methods outperform their DGR-based counterparts across all metrics other than RAM usage. The lower accuracy of the DGR and DGR+d may be because of the lower number of samples for CK+ making it harder to reconstruct high-dimensional discriminative facial images for each expression class. LGR handles this increased data complexity better as it does not require reconstructing high-dimensional pseudo-samples, making it more robust. LR reports the lowest Training Time and CPU usage as it does not need to train a generative model and also stores only latent representations in the memory buffer. LGR-based methods, however, seem to close the gap to LR, achieving the best GPU Usage and memory consumption and second-best results on training time and CPU usage. Directly compared to DGR-based methods, LGR offers improvements in model accuracy with reductions in all other metrics other than RAM usage.

B. RAF-DB Results

Table III presents the results for RAF-DB, where the highest accuracy is reported by the proposed LGR+d approach with LGR achieving the second-best results. These results are comparable to the state-of-the-art results of 88.98% [59] despite following the incremental learning protocol. Since

TABLE III

TASK-IL RESULTS ON RAF-DB. BEST RESULTS FOR EACH ROW ARE HIGHLIGHTED IN **BOLD** WHILE SECOND BEST ARE IN [BRACES].

	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	79.52	79.69	81.79	[*83.33*] ▲	82.53	**86.91** ▲
Training time (s)▼	1444.60	**1102.67**	2010.47	[*1159.43*] ▼	2020.03	1182.37 ▼
RAM Usage (MB) ▼	2836.50	2831.52	[*2808.43*]	2846.20 ▲	**2789.65**	2845.07 ▲
CPU Usage (s) ▼	1465.10	**1112.74**	2008.67	[*1167.70*] ▼	2018.84	1197.15 ▼
GPU Usage (%) ▼	48.31	37.00	64.07	**35.92** ▼	63.97	[*36.01*] ▼
GPU Memory (MB) ▼	7390.67	**4310.00**	8204.00	[4314.00] ▼	8204.00	[*4314.00*] ▼

TABLE IV

TASK-IL RESULTS ON AFFECTNET. BEST RESULTS FOR EACH ROW ARE HIGHLIGHTED IN **BOLD** WHILE SECOND BEST ARE IN [BRACES].

	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	67.46	67.50	67.46	66.63 ▼	**69.41**	[*69.34*] ▼
Training time (s)▼	5507.53	**5108.70**	6154.00	5435.43 ▼	6124.80	[*5410.70*] ▼
RAM Usage (MB) ▼	2918.02	2898.21	**2874.37**	2917.04 ▲	[*2894.28*]	2908.52 ▲
CPU Usage (s) ▼	5651.47	**5251.80**	6272.37	5583.11 ▼	6274.24	[*5557.61*] ▼
GPU Usage (%) ▼	12.43	8.87	21.18	**8.38** ▼	20.37	[*8.59*] ▼
GPU Memory (MB) ▼	7581.33	[*4310.00*]	8204.00	**4154.00** ▼	8204.00	**4154.00** ▼

RAF-DB is a much larger dataset ($\approx 30K$ samples) compared to CK+ ($\approx 1.3K$ samples), NR and LR approaches face the challenge of allocating an appropriate fraction of the memory buffer for representative samples for each expression category. LGR-based methods also outperform their DGR-based counterparts across all metrics other than RAM usage where they are fractionally more expensive. Different from CK+, here we see that the DGR-based approaches are able to achieve high accuracy scores as with the increased number of training samples available, the models are able to efficiently generate pseudo-samples for the different expression classes to mitigate forgetting. LGR improves on this further by eliminating the need to generate high-dimensional data samples and instead applies generative replay of learnt feature representations to mitigate forgetting in the model. Similar to CK+ results, LR reports the lowest Training Time and CPU usage as well as the lowest GPU consumption, fractionally better than LGR-based methods. LGR-based methods, while achieving the best model accuracy, also close the gap to LR, achieving the best GPU Usage and similar GPU memory consumption. They also achieve second-best results on training time and CPU usage, compared to LR. Directly compared to DGR-based methods, despite fractionally higher RAM consumption LGR offers improvements in model accuracy with reductions on all other metrics.

C. AffectNet Results

Table IV presents AffectNet results, where the highest accuracy is reported DGR+d approach with the proposed LGR+d adaptation achieving the second-best results. These results significantly improve upon the baseline results (50%) on the *downsampled* data split for AffectNet [29], Similar to RAF-DB the higher number of data samples per task allows the DGR-based methods to effectively construct representative pseudo-samples for each expression category, mitigating forgetting. LR and NR are also achieve comparable results, effectively balancing novel *vs.* past learning. LR, similar to CK+ and RAF-DB evaluations, reports the lowest training time and CPU usage while LGR+d is able to close the gap, achieving second-best results on these metrics but improves upon LR by achieving the lowest GPU usage and peak GPU memory consumption. Directly compared to DGR-based methods, LGR+d achieves second-best model accuracy while significantly reducing the training time, CPU usage, GPU usage and GPU memory consumption.

D. Ablation: Selecting the Pre-trained Backbone as Root

LGR works under the assumption that with a sufficiently pre-trained *root*, the extracted feature representations can be used effectively to rehearse past knowledge, eliminating the need to operate with high-dimensional input patterns. Thus, the selection of the *root* network becomes critical to the performance of the model. In the results presented above, we use a VGG-16-based [54] *root* for all the models. This is done after comparing the results across the three datasets with four different backbones trained on ImageNet, namely, AlexNet [57], MobileNet-V2 [55], ResNet-18 [56] along with VGG-16. We use the feature extraction layers, that is, up to the flattened output of the final *conv* layer, as the model root where this flattened output represents the *latent replay layer* used to train the LGR generator model for pseudo-rehearsal. For the solver, we append a *top* network with a dense (with 128 units) layer and an output layer.

Table V presents the results comparing the four backbone networks on AffectNet, the largest amongst the compared datasets. As can be seen, across all backbones, LGR-based approaches offer improvements over their DGR-based counterparts with the most significant difference witnessed for VGG-16. Furthermore, this is also significant as, with VGG-16 being the 'heaviest' of the backbones with $\approx 138M$ parameters, LGR is able to significantly reduce the memory and computational expense of all the compared models. VGG-16 also helps achieve the highest model accuracy across all approaches between the four backbones, with AlexNet achieving the next best results while also being a relatively 'heavy' backbone with $\approx 61M$ parameters. ResNet-18 is relatively 'lighter' with $\approx 11M$ parameters but achieves a reasonable compromise between model accuracy and memory and resource-efficiency of the different approaches. The performance gains, however, are not at par with AlexNet or VGG-16. MobileNet-V2 is the 'lightest' with only $\approx 3.5M$ parameters which also affects the model accuracy, providing the worst results amongst the four backbones.

VI. DISCUSSION AND CONCLUSIONS

A. Resource-efficiency of LGR

With ML-based FER becoming pervasive in its application in our daily lives, it is important to consider the computational, and in turn, environmental and monetary cost of running these algorithms, particularly on resource-crunched devices. Our central objective, in this work, is to present a resource-efficient pseudo-rehearsal method that

TABLE V

TASK-IL RESULTS ON AFFECTNET ACROSS PRE-TRAINED ALEXNET [57], MOBILENET-V2 [55], RESNET-18 [56] AND VGG-16 [54] NETWORKS AS MODEL ROOT. BEST RESULTS FOR EACH METRIC FOR INDIVIDUAL ROOTS ARE HIGHLIGHTED IN **BOLD** WHILE SECOND BEST ARE IN [BRACES].

AlexNet-based Root Model	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	67.38	65.25	65.22	63.32 ▼	**69.07**	67.24 ▼
Training time (s) ▼	[*4832.23*]	4860.53	5309.63	**4809.50** ▼	5105.90	4947.90 ▼
RAM Usage (MB) ▼	3006.94	2891.12	[*2866.98*]	2868.26 ▲	**2865.94**	2882.76 ▲
CPU Usage (s) ▼	[*4975.21*]	5005.71	5455.58	**4951.63** ▼	5249.81	5093.02 ▼
GPU Usage (%) ▼	1.54	**1.07**	7.31	[*1.29*] ▼	7.34	1.31 ▼
GPU Memory (MB) ▼	5217.33	[*1818.00*]	5836.00	**1542.00** ▼	5836.00	**1542.00** ▼

ResNet-18-based Root Model	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	[*61.36*]	**61.84**	52.69	53.71 ▲	56.26	60.83 ▲
Training time (s) ▼	5068.93	**4884.23**	5272.73	4995.77 ▼	5271.57	[*4955.30*] ▼
RAM Usage (MB) ▼	3254.08	2926.91	**2866.26**	2923.30 ▲	[*2868.86*]	2920.68 ▲
CPU Usage (s) ▼	5218.62	**5029.07**	5417.99	5136.38 ▼	5415.59	[*5095.95*] ▼
GPU Usage (%) ▼	3.00	**2.22**	9.76	[*2.35*] ▼	10.27	**2.22** ▼
GPU Memory (MB) ▼	4990.67	[*1578.67*]	5248.00	**1554.00** ▼	5248.00	1554.00 ▼

MobileNet-V2-based Root Model	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	**59.76**	59.23	53.06	53.37 ▲	53.38	[*59.41*] ▲
Training time (s) ▼	[*5013.67*]	5062.03	5480.90	**4974.10** ▼	5403.23	5016.10 ▼
RAM Usage (MB) ▼	2987.18	3117.64	[*2869.16*]	2870.93 ▲	2869.65	**2866.70** ▲
CPU Usage (s) ▼	[*5158.61*]	5209.47	5631.19	**5117.24** ▼	5553.72	5160.64 ▼
GPU Usage (%) ▼	4.04	**2.54**	10.42	2.72 ▼	10.95	[*2.60*] ▼
GPU Memory (MB) ▼	4919.33	[*1676.67*]	5414.00	**1604.00** ▼	5414.00	**1604.00** ▼

VGG-16-based Root Model	NR [30]	LR [31]	DGR [26]	LGR (Ours)	DGR+d [49]	LGR+d (Ours)
Accuracy (%) ▲	67.46	67.50	67.46	66.63 ▼	**69.41**	[*69.34*] ▼
Training time (s) ▼	5507.53	**5108.70**	6154.00	5435.43 ▼	6124.80	[*5410.70*] ▼
RAM Usage (MB) ▼	2918.02	2898.21	**2874.37**	2917.04 ▲	[*2894.28*]	2908.52 ▲
CPU Usage (s) ▼	5651.47	**5251.80**	6272.37	5583.11 ▼	6274.24	[*5557.61*] ▼
GPU Usage (%) ▼	12.43	8.87	21.18	**8.38** ▼	20.37	[*8.59*] ▼
GPU Memory (MB) ▼	7581.33	[*4310.00*]	8204.00	**4154.00** ▼	8204.00	**4154.00** ▼

can reduce memory and computational costs of CL without compromising on model performance. Our experiments show that the proposed LGR approach significantly improves upon DGR in terms of memory and resource consumption. LGR (and LGR+d) consistently perform better than DGR, reducing the training time and CPU usage while also offering a significant reduction in GPU usage and memory consumption. At the same time, they offer competitive, if not better, performance with respect to replay-based NR and LR. Although LR consistently performs the best in terms of reduced training times and CPU usage, LGR comes close second while also significantly outperforming all methods on GPU consumption. With LGR, we propose eliminating the need of maintaining memory buffers to store data samples (as in the case of NR) or even latent representations (as in the case of LR) significantly reducing the memory footprint of replay-based CL. Furthermore, as LGR uses a 'lighter' generator that learns to reconstruct only low-dimensional representations, there are significant reductions in terms of computational expenses compared to other pseudo-rehearsal methods such as DGR. Additionally, LGR uses a pre-trained *root* with the reconstructed latent representations replayed to the *solver* 'from the middle' of the network. This further reduces the computational cost of LGR.

B. Limitations and Future Work

We use a *root* network for LGR following the VGG-16 architecture pre-trained on ImageNet. Despite optimising some network parameters like G_{FC} and G_{OUT}, this still limits the application of LGR to certain domains that are well-represented in the *root* pre-training. To realise the true potential of LGR, especially for real-world adaptation, it is important to allow the *root* to also evolve during the training, at least *slowly* at domain-level, allowing the model to extend its learning to novel data, more freely. Van de Ven *et al.* [48] in their *brain-inspired* approach embed the generator model within the main learning model for *internal replay* of information. Principally, this is similar to our motivations for *latent generative replay* but may also offer the additional benefit of extending model learning to novel applications without needing to pre-train the *root*. However, they do not optimise for resource-efficiency, which forms the central focus of our approach. Future work for us would focus on enabling real-time adaptation for the *root* while still adhering to the assumptions of feature extraction layers remaining sufficiently consistent across learning [31]. Additionally, comparing generated latent representations with *root*-extracted features, at different stages of learning, will help evaluate LGR's ability to efficiently model input data distributions. Furthermore, we would also like to extend LGR to other resource-intensive replay strategies such as Gradient Episodic Memory (GEM) [36] and iCaRL [35] and evaluate the real-world application of LGR on robotic platforms.

ACKNOWLEDGEMENT

For the purpose of *open access*, the authors have applied a Creative Commons Attribution (CC BY) license to any Accepted Manuscript version arising.

Data Access Statement: This study involves secondary analyses of existing datasets, that are described and cited in the text. Licensing restrictions prevent sharing of the datasets.

Code Access: `https://bit.ly/3S7Awj5`.

REFERENCES

[1] S. Voghoei *et al.*, "Deep learning at the edge," in *International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2018, pp. 895–901.

[2] D. Preuveneers *et al.*, "Resource usage and performance trade-offs for machine learning models in smart environments," *Sensors*, vol. 20, no. 4, p. 1176, 2020.

[3] I. Abdić *et al.*, "Driver frustration detection from audio and video in the wild," in *25th International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, p. 1354–1360.

[4] C. Hewitt *et al.*, "CNN-based Facial Affect Analysis on Mobile Devices," *arXiv preprint arXiv:1807.08775*, 2018.

[5] Y. Guo *et al.*, "Real-time facial affective computing on mobile devices," *Sensors*, vol. 20, no. 3, p. 870, Feb. 2020.

[6] T. Chen *et al.*, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation*. USENIX, Oct. 2018, pp. 578–594.

[7] R. Schwartz *et al.*, "Green AI," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[8] P. Ekman *et al.*, "Constants across cultures in the face and emotion." *J Pers Soc Psychol*, vol. 17, no. 2, p. 124, 1971.

[9] R. W. Picard, *Affective Computing*. Cambridge, MA, USA: MIT Press, 1997, vol. 252.

[10] M. McCloskey *et al.*, "Catastrophic interference in connectionist networks: The sequential learning problem," ser. Psychology of Learning and Motivation. Academic Press, 1989, vol. 24, pp. 109 – 165.

[11] M. Dundar *et al.*, "Learning Classifiers When the Training Data is Not IID," in *International Joint Conference on Artifical Intelligence*, 2007, p. 756–761.

[12] T. Lesort *et al.*, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Information fusion*, vol. 58, pp. 52–68, 2020.

[13] N. Churamani *et al.*, "Continual Learning for Affective Robotics: Why, What and How?" in *IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, 2020, pp. 425–431.

[14] S. Thrun, *Lifelong Learning Algorithms*. Boston, MA: Springer US, 1998, pp. 181–209.

[15] G. I. Parisi *et al.*, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[16] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[17] F. Zenke *et al.*, "Continual learning through synaptic intelligence," in *Proceedings of the 34th International Conference on Machine Learning*. JMLR.org, 2017, pp. 3987–3995.

[18] C. Henning *et al.*, "Posterior meta-replay for continual learning," in *Conference on Neural Information Processing Systems*, 2021.

[19] J. von Oswald *et al.*, "Continual learning with hypernetworks," in *International Conference on Learning Representations*, 2020.

[20] A. Robins, "Catastrophic forgetting in neural networks: the role of rehearsal mechanisms," in *Proceedings 1993 The First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems*, Nov 1993, pp. 65–68.

[21] A. Robins, "Catastrophic forgetting, rehearsal and pseudorehearsal," *Connection Science*, vol. 7, no. 2, pp. 123–146, 1995.

[22] Y. D. Kwon *et al.*, "Exploring system performance of continual learning for mobile and embedded sensing applications," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 319–332.

[23] E. Fini *et al.*, "Online continual learning under extreme memory constraints," in *European Conference on Computer Vision*. Springer, 2020, pp. 720–735.

[24] J. Smith *et al.*, "Memory-efficient semi-supervised continual learning: The world is its own replay buffer," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[25] N. D. Rodríguez *et al.*, "Don't forget, there is more than forgetting: new metrics for Continual Learning," in *Continual Learning Workshop at NeurIPS.*, 2018.

[26] H. Shin *et al.*, "Continual Learning with Deep Generative Replay," in *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 2990–2999.

[27] P. Lucey *et al.*, "The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression," in *Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010)*, San Francisco, USA, 2010, pp. 94–101.

[28] S. Li *et al.*, "Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2852–2861.

[29] A. Mollahosseini *et al.*, "Affectnet: A database for facial expression, valence, and arousal computing in the wild," *IEEE Transactions on Affective Computing*, 2018.

[30] Y. Hsu *et al.*, "Re-evaluating Continual Learning Scenarios: A Categorization and Case for Strong Baselines," in *Continual Learning Workshop at NeurIPS.*, 2018.

[31] L. Pellegrini *et al.*, "Latent replay for real-time continual learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 203–10 209.

[32] G. M. van de Ven *et al.*, "Three scenarios for continual learning," *CoRR*, vol. abs/1904.07734, 2019.

[33] R. Kemker *et al.*, "Measuring catastrophic forgetting in neural networks," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, ser. AAAI'18. AAAI Press, 2018.

[34] G. Hu *et al.*, "Prioritized experience replay for continual learning," in *2021 6th International Conference on Computational Intelligence and Applications (ICCIA)*, 2021, pp. 16–20.

[35] S.-A. Rebuffi *et al.*, "iCaRL: Incremental Classifier and Representation Learning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[36] D. Lopez-Paz *et al.*, "Gradient episodic memory for continual learning," in *International Conference on Neural Information Processing Systems*, ser. NIPS'17, 2017, pp. 6470–6479.

[37] D. Rolnick *et al.*, "Experience Replay for Continual Learning," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[38] B. Bagus *et al.*, "An Investigation of Replay-based Approaches for Continual Learning," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1–9.

[39] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani *et al.*, Eds., vol. 27. Curran Associates, Inc., 2014.

[40] D. P. Kingma *et al.*, "Auto-encoding variational bayes," 2014.

[41] A. Seff *et al.*, "Continual learning in generative adversarial nets," *CoRR*, vol. abs/1705.08395, 2017.

[42] C. V. Nguyen *et al.*, "Variational continual learning," in *International Conference on Learning Representations*, 2018.

[43] N. Churamani *et al.*, "CLIFER: Continual Learning with Imagination for Facial Expression Recognition," in *IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, 2020, pp. 322–328.

[44] R. Kemker *et al.*, "FearNet: Brain-Inspired Model for Incremental Learning," in *International Conference on Learning Representations*, 2018.

[45] K. Thandiackal *et al.*, "Match what matters: Generative implicit feature replay for continual learning," *CoRR*, vol. abs/2106.05350, 2021.

[46] Y. Xiang *et al.*, "Incremental Learning Using Conditional Adversarial Networks," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6618–6627.

[47] X. Liu *et al.*, "Generative feature replay for class-incremental learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, Jun. 2020.

[48] G. M. van de Ven *et al.*, "Brain-inspired replay for continual learning with artificial neural networks," *Nature Communications*, vol. 11, no. 1, Aug. 2020.

[49] G. M. van de Ven *et al.*, "Generative replay with feedback connections as a general strategy for continual learning," *CoRR*, vol. abs/1809.10635, 2018.

[50] S. Li *et al.*, "Deep facial expression recognition: A survey," *IEEE Transactions on Affective Computing*, March 2020.

[51] V. Lomonaco *et al.*, "Avalanche: an end-to-end library for continual learning," in *IEEE/CVF Computer Vision and Pattern Recognition Conference*, ser. Continual Learning in Computer Vision Workshop, 2021.

[52] T. Chen *et al.*, "Training deep nets with sublinear memory cost," *CoRR*, vol. abs/1604.06174, 2016.

[53] N. K. Jha *et al.*, "The ramifications of making deep neural networks compact," in *2019 32nd International Conference on VLSI Design (VLSID)*, 2019, pp. 215–220.

[54] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[55] M. Sandler *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[56] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[57] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, 2012, p. 1097–1105.

[58] H. Ding *et al.*, "FaceNet2ExpNet: Regularizing a deep face recognition net for expression recognition," in *12th IEEE International Conference on Automatic Face Gesture Recognition (FG)*, 2017, pp. 118–126.

[59] T.-H. Vo *et al.*, "Pyramid with super resolution for in-the-wild facial expression recognition," *IEEE Access*, vol. 8, pp. 131 988–132 001, 2020.

Latent Generative Replay for Resource-Efficient Continual Learning of Facial Expressions

Supplementary Data and Results

Samuil Stoychev, Nikhil Churamani and Hatice Gunes
Department of Computer Science and Technology, University of Cambridge, United Kingdom
`ss2719@cantab.ac.uk, {nikhil.churamani,hatice.gunes}@cl.cam.ac.uk`

I. Datasets and Experiment Settings

A. Datasets

1) CK+: The CK+ dataset is a popular Facial Expression Recognition (FER) dataset consisting of video recordings of 123 subjects, annotated for 7 expression classes namely, *Anger, Surprise, Fearful, Disgust, Happy, Sad* and *Contempt*. In total, there are 327 sequences showing a shift from *neutral* to peak expression intensity. We take the last 3 frames for each sequence to represent the corresponding expression class while the first frame is used to constitute the *neutral* class samples. That gives us a total of 1310 images, each labelled with one of 8 expressions. CK+ is an imbalanced dataset with respect to the class distribution with 'happy' samples dominating the distribution while other classes such as 'contempt' are underrepresented. To partition the images into a train and a test set, we perform a *cross-subject* split, assigning 86 subjects to the train set and 37 to the test dataset (or a $70 : 30$ split) while maintaining the data distribution. Cross-subject validation is a common approach in FER ensuring that the learned model can generalise well to new, unseen subjects.

2) RAF-DB: The Real-world Affective Faces Database (RAF-DB) dataset consists of $\approx 15K$ facial images downloaded from the internet and manually labelled by multiple annotators for six expression classes namely, *Surprise, Fearful, Disgust, Happy, Sad* and *Anger* along with *Neutral* faces. RAF-DB samples represent *in-the-wild* settings with great diversity with respect to gender, race, head pose, facial attributes like facial hair and skin colour, and variations resulting from lighting conditions. For our experiments, we use the original train-test split as provided by the authors of the dataset.

3) AffectNet: The AffectNet dataset consists of more than a million facial images downloaded from the internet. Half of the images ($\approx$ 500K) are manually labelled by multiple annotators for arousal and valence labels as well as eight facial expression categories namely *Anger, Surprise, Fearful, Disgust, Happy, Sad, Contempt* and *Neutral*. Much like RAF-DB, AffectNet samples also represent *in-the-wild* conditions. As these images are captured randomly from the web, for several samples the face is either occluded or not present or annotated for the data samples. Thus, from the manually labelled samples, $\approx 287k$ provide expression labels. The data distribution for the AffectNet dataset is also highly imbalanced for the 8 expression classes with 'Happy' samples constituting $\approx 32\%$ of the usable samples. We use the *downsampled* data split for AffectNet with $\approx 90K$ training samples and $4K$ test samples.

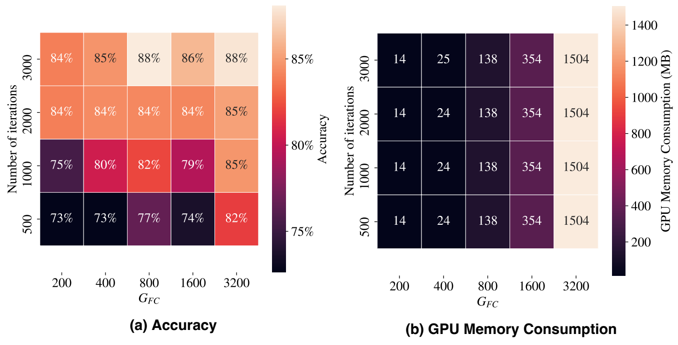

Fig. 1. (a) Accuracy and (b) GPU Memory Consumption for LGR over different hyper-parameters for CK+.

B. Preprocessing

We pre-process the images by extracting the face region in the images using a pre-trained Face Detector from the the dlib Python library[1]. We additionally extend the width and the height of the detected area by 20% to ensure the cropped area captures facial features such as the ears, the forehead and the chin. The extracted face-centred images are then resized to $(100 \times 100 \times 3)$ before using them to train the models.

II. Setting Model Hyper-parameters

Latent Generative Replay (LGR) is parameterised by the size of the generator, more specifically, the number of hidden units G_{FC} and the output size of the generator G_{OUT}. For the three datasets, G_{FC} and G_{OUT} are of different sizes with G_{OUT} fixed at $30,000$ (image size) for Deep Generative Replay (DGR) and 4096 (latent representations from VGG-16) for LGR-based approaches. Thus, the performance of the pseudo-rehearsal methods are hinged on optimising G_{FC} while the buffer size B_{size} determines the performance for rehearsal-based methods. Thus, we conduct a grid-search hyper-parameter optimisation for LGR on by searching over

[1]http://dlib.net

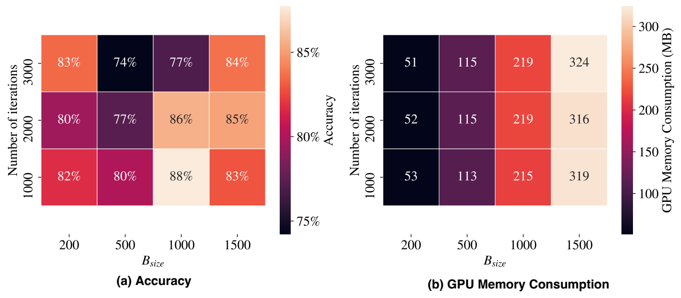

Fig. 2. (a) Accuracy and (b) GPU Memory Consumption for LR over different hyper-parameters for CK+.

$G_{FC} \in [200, 400, 800, 1600, 3200]$ and varying the number of iterations across $i \in [500, 1000, 2000, 3000]$. For each distinct combination of hyper-parameters, we train the model 3 times and average the system metrics.

We see that increasing the value of the G_{FC} tends to improve model accuracy (see Figure 1 (a)) yet, a higher G_{FC} also translates to a higher memory cost with the GPU memory consumption increasing non-linearly and growing over 100 times as G_{FC} increases from 200 to 3200 (see Figure 1 (b)). This rapid increase in GPU memory consumption results from the increased size of the generator. Although here we only illustrate the trade-off between accuracy and GPU memory consumption, an increase in G_{FC} size also increases other system metrics including GPU usage, CPU usage and training time. The number of iterations does not impact memory consumption and improves model accuracy with convergence at around 2000 iterations.

A similar process is repeated for optimising the hyper-parameters for Naïve Rehearsal (NR), LR and DGR. For NR and LR, the buffer size (B_{size}) is optimised by varying it between 200 and 1500 (see Figure 2). B_{size} introduces a trade-off, similar to G_{FC} for LGR, where a larger replay buffer size tends to increase model accuracy while also linearly increasing the memory consumption of the model.

III. Ablation Results

A. Results for CK+

1) AlexNet: Table I presents CK+ results using a pre-trained AlexNet-based root network.

TABLE I
CK+ results using AlexNet. Best results for each row are in **bold** while second best are in [braces].

	NR	LR	DGR	LGR	DGR+d	LGR+d
Accuracy (%) ▲	**98.19**	[*95.55*]	93.23	92.31 ▼	91.97	95.01 ▲
Training time (s) ▼	[*661.17*]	**632.20**	871.13	681.60 ▼	879.27	670.67 ▼
RAM Usage (MB) ▼	2802.47	2781.29	**2754.90**	2782.38 ▲	[*2759.07*]	2781.86 ▲
CPU Usage (s) ▼	[*758.71*]	**729.03**	962.26	778.49 ▼	977.42	766.90 ▼
GPU Usage (%) ▼	[*3.91*]	**2.64**	27.06	4.55 ▼	26.93	4.47 ▼
GPU Memory (MB) ▼	3916.00	[*1744.00*]	5682.00	**1546.00** ▼	5682.00	**1546.00** ▼

2) MobileNet-V2: Table II presents CK+ results using a pre-trained MobileNet-V2-based root network.

TABLE II
CK+ results using MobileNet-V2. Best results for each row are in **bold** while second best are in [braces].

	NR	LR	DGR	LGR	DGR+d	LGR+d
Accuracy (%) ▲	78.75	[*82.37*]	73.34	78.59 ▼	76.02	**87.96** ▲
Training time (s) ▼	[*720.23*]	**678.17**	922.10	743.73 ▼	994.47	723.03 ▼
RAM Usage (MB) ▼	**2755.10**	2760.72	2765.46	2763.42 ▲	2765.23	[*2759.17*] ▲
CPU Usage (s) ▼	819.46	**775.46**	1,015.95	846.30 ▼	1,091.35	[*816.91*] ▼
GPU Usage (%) ▼	8.71	**5.73**	30.55	[*6.61*] ▼	29.24	6.87 ▼
GPU Memory (MB) ▼	3580.00	[*1329.33*]	5074.00	**1284.00** ▼	5074.00	**1284.00** ▼

3) ResNet-18: Table III presents CK+ results using a pre-trained ResNet-18-based root network.

TABLE III
CK+ results using ResNet-18. Best results for each row are in **bold** while second best are in [braces].

	NR	LR	DGR	LGR	DGR+d	LGR+d
Accuracy (%) ▲	[*82.74*]	81.66	74.34	79.48 ▲	74.63	**88.47** ▲
Training time (s) ▼	[*701.57*]	**668.03**	920.30	702.10 ▼	923.60	722.80 ▼
RAM Usage (MB) ▼	**2752.49**	2821.64	[*2759.36*]	2821.92 ▲	2759.57	2821.43 ▲
CPU Usage (s) ▼	801.59	**768.71**	1018.13	[*798.76*] ▼	1019.85	820.10 ▼
GPU Usage (%) ▼	9.24	**6.07**	31.53	7.24 ▼	31.22	[*6.90*] ▼
GPU Memory (MB) ▼	3618.00	[*1321.33*]	5166.00	**1308.00** ▼	5166.00	**1308.00** ▼

B. Results for RAF-DB

1) AlexNet: Table IV presents RAF-DB results using a pre-trained AlexNet-based root network.

TABLE IV
RAF-DB results using AlexNet. Best results for each row are in **bold** while second best are in [braces].

	NR	LR	DGR	LGR	DGR+d	LGR+d
Accuracy (%) ▲	75.72	78.62	81.58	[*84.28*] ▲	83.91	**86.85** ▲
Training time (s) ▼	[*797.83*]	**740.27**	1063.10	809.23 ▼	1129.30	803.93 ▼
RAM Usage (MB) ▼	3057.71	2797.17	[*2795.09*]	2803.99 ▲	**2792.01**	2,801.98 ▲
CPU Usage (s) ▼	805.54	**740.60**	1064.31	[*798.84*] ▼	1146.20	805.16 ▼
GPU Usage (%) ▼	9.08	**6.48**	34.42	[*7.73*] ▼	33.34	7.89 ▼
GPU Memory (MB) ▼	5232.00	[*1818.00*]	5836.00	**1542.00** ▼	5836.00	**1542.00** ▼

2) MobileNet-V2: Table V presents RAF-DB results using a pre-trained MobileNet-V2-based root network.

TABLE V
RAF-DB results using MobileNet-V2. Best results for each row are in **bold** while second best are in [braces].

	NR	LR	DGR	LGR	DGR+d	LGR+d
Accuracy (%) ▲	**81.06**	80.94	80.10	80.10 ▲▼	80.10	[*81.02*] ▲
Training time (s) ▼	979.93	**834.13**	1250.57	902.70 ▼	1241.73	[*875.70*] ▼
RAM Usage (MB) ▼	2940.20	**2790.07**	2793.48	2800.13 ▲	2798.96	[*2792.01*] ▼
CPU Usage (s) ▼	966.44	**837.57**	1257.85	908.76 ▼	1246.60	[*877.00*] ▼
GPU Usage (%) ▼	20.69	**14.45**	42.70	[*14.57*] ▼	43.26	15.00 ▼
GPU Memory (MB) ▼	4880.00	[*1679.33*]	5414.00	**1604.00**	5414.00	**1604.00** ▼

3) ResNet-18: Table VI presents RAF-DB results using a pre-trained ResNet-18-based root network.

TABLE VI
RAF-DB RESULTS USING RESNET-18. BEST RESULTS FOR EACH ROW ARE HIGHLIGHTED IN **BOLD** WHILE SECOND BEST ARE IN [BRACES].

	NR	LR	DGR	LGR		DGR+d	LGR+d	
Accuracy (%) ▲	82.17	[*82.64*]	80.10	80.10	▲▼	80.10	**82.84**	▲
Training time (s) ▼	882.30	**792.53**	1225.70	884.13	▼	1220.10	[*868.23*]	▼
RAM Usage (MB) ▼	3240.22	2846.39	[*2798.86*]	2850.88	▲	**2793.68**	2850.32	▲
CPU Usage (s) ▼	883.95	**785.29**	1228.59	891.16	▼	1223.57	[*875.40*]	▼
GPU Usage (%) ▼	19.33	[*13.08*]	41.95	**12.94**	▼	41.63	13.24	▼
GPU Memory (MB) ▼	5017.33	[*1577.33*]	5248.00	**1554.00**	▼	5248.00	**1554.00**	▼

IV. MODEL ARCHITECTURES

A. *VGG-16 Based Solver*

We use a VGG-16-based architecture pre-trained on the ImageNet benchmark to implement the solver (see Figure 3). We use $(100 \times 100 \times 3)$ RGB images as input to the model. The model is split into the pre-trained *root* layers resulting in a 4096-d latent representation layer while the final 3 fully-connected layers are used for task-specific learning in the *top* of the model.

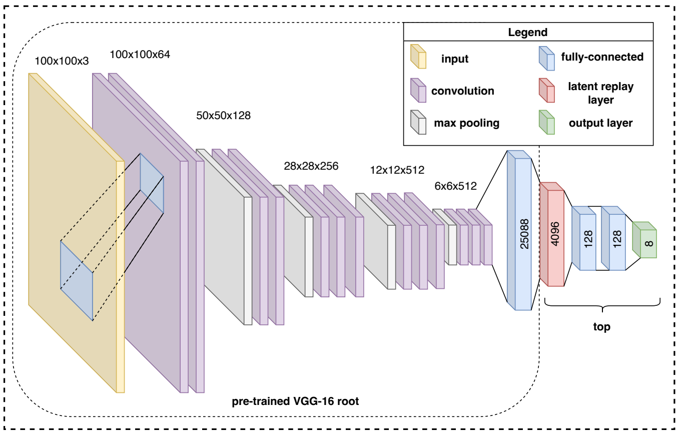

Fig. 3. VGG-16 Implementation.

Table VII presents the implementation summary of the model along with the number for parameters and estimated model size (in MB). Since the *root* of the model is pre-trained and frozen during the training, only the *top* of the model contributes to the number of trainable parameters.

B. *Variational Autoencoder (VAE)*

For pseudo-rehearsal, both in DGR and LGR-based methods, we use the Variational Autoencoder (VAE) architecture as the *generator* model for the *scholar*. Based on the image size $(100 \times 100 \times 3)$ or the size of the latent representation layer (4096 for VGG-16-based solver), the generator uses several fully-connected layer to reconstruct the latent representations. The generator is split into 4 sub-modules representing the encoder (`fcE`), the decpder (`fcD`, network layers extracting the latent variable z (`toZ`) and network layers sampling from the latent variable z (`fromZ`).

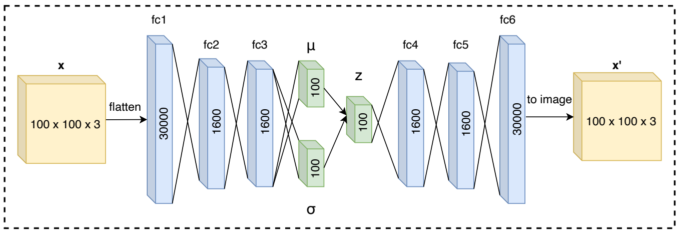

Fig. 4. VAE implementation for the DGR *generator*.

Table VIII presents the implementation summary of the VAE-based generator (as shown in Figure 5) used for reconstructing 4096d latent representations from the VGG-16 root for the LGR-based methods along with the number for parameters and estimated model size (in MB).

TABLE VII
SUMMARY OF THE VGG-16-BASED SOLVER ARCHITECTURE.

Layer	Output Shape	Number of Parameters
`Conv2d-1`	[-1, 64, 100, 100]	1,792
`ReLU-2`	[-1, 64, 100, 100]	0
`Conv2d-3`	[-1, 64, 100, 100]	36,928
`ReLU-4`	[-1, 64, 100, 100]	0
`MaxPool2d-5`	[-1, 64, 50, 50]	0
`Conv2d-6`	[-1, 128, 50, 50]	73,856
`ReLU-7`	[-1, 128, 50, 50]	0
`Conv2d-8`	[-1, 128, 50, 50]	147,584
`ReLU-9`	[-1, 128, 50, 50]	0
`MaxPool2d-10`	[-1, 128, 25, 25]	0
`Conv2d-11`	[-1, 256, 25, 25]	295,168
`ReLU-12`	[-1, 256, 25, 25]	0
`Conv2d-13`	[-1, 256, 25, 25]	590,080
`ReLU-14`	[-1, 256, 25, 25]	0
`Conv2d-15`	[-1, 256, 25, 25]	590,080
`ReLU-16`	[-1, 256, 25, 25]	0
`MaxPool2d-17`	[-1, 256, 12, 12]	0
`Conv2d-18`	[-1, 512, 12, 12]	1,180,160
`ReLU-19`	[-1, 512, 12, 12]	0
`Conv2d-20`	[-1, 512, 12, 12]	2,359,808
`ReLU-21`	[-1, 512, 12, 12]	0
`Conv2d-22`	[-1, 512, 12, 12]	2,359,808
`ReLU-23`	[-1, 512, 12, 12]	0
`MaxPool2d-24`	[-1, 512, 6, 6]	0
`Conv2d-25`	[-1, 512, 6, 6]	2,359,808
`ReLU-26`	[-1, 512, 6, 6]	0
`Conv2d-27`	[-1, 512, 6, 6]	2,359,808
`ReLU-28`	[-1, 512, 6, 6]	0
`Conv2d-29`	[-1, 512, 6, 6]	2,359,808
`ReLU-30`	[-1, 512, 6, 6]	0
`MaxPool2d-31`	[-1, 512, 3, 3]	0
`AdaptiveAvgPool2d-32`	[-1, 512, 7, 7]	0
`Linear-33`	[-1, 4096]	102,764,544
`Linear-34`	[-1, 128]	524,416
`Dropout-35`	[-1, 128]	0
`Linear-36`	[-1, 8]	1,032
Total parameters:	118,004,680	
Estimated Total Size (MB):	493.62	

The VAE-based generator used for reconstructing $100 \times 100 \times 3$ samples for the DGR-based methods is illustrated in Figure 8 while Table IX presents the implementation summary of the model along with the number for parameters and estimated model size (in MB).

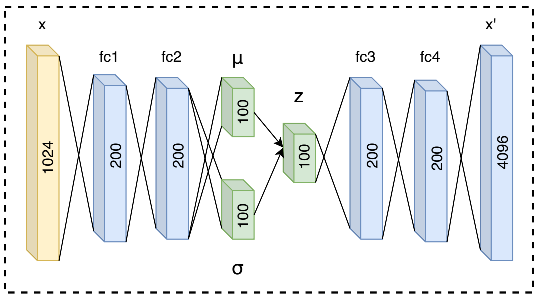

Fig. 5. VAE implementation for the LGR *generator*.

TABLE VIII
SUMMARY OF THE VAE-BASED GENERATOR USED FOR LGR.

Layer	Output Shape	Number of Parameters
`fcE`		
`Flatten-1`	[-1, 4096]	0
`LinearExcitability-2`	[-1, 200]	819,400
`ReLU-3`	[-1, 200]	0
`fc_layer-4`	[-1, 200]	0
`LinearExcitability-5`	[-1, 200]	40,200
`ReLU-6`	[-1, 200]	0
`fc_layer-7`	[-1, 200]	0
`toZ`		
`LinearExcitability-8`	[-1, 100]	20,100
`fc_layer-9`	[-1, 100]	0
`LinearExcitability-10`	[-1, 100]	20,000
`fc_layer-11`	[-1, 100]	0
`fromZ`		
`LinearExcitability-12`	[-1, 200]	20,200
`ReLU-13`	[-1, 200]	0
`fcD`		
`LinearExcitability-14`	[-1, 200]	40,200
`ReLU-15`	[-1, 200]	0
`fc_layer-16`	[-1, 200]	0
`LinearExcitability-17`	[-1, 4096]	819,400
`Sigmoid-18`	[-1, 4096]	0
`fc_layer-19`	[-1, 4096]	0
Total parameters:	1,785,004	
Estimated Total Size (MB):	6.90	

TABLE IX
SUMMARY OF THE VAE-BASED GENERATOR USED FOR DGR.

Layer	Output Shape	Number of Parameters
`fcE`		
`Flatten-1`	[-1, 30000]	0
`LinearExcitability-2`	[-1, 1600]	48,001,600
`ReLU-3`	[-1, 1600]	0
`fc_layer-4`	[-1, 1600]	0
`LinearExcitability-5`	[-1, 1600]	2,561,600
`ReLU-6`	[-1, 1600]	0
`fc_layer-7`	[-1, 1600]	0
`toZ`		
`LinearExcitability-8`	[-1, 100]	160,100
`fc_layer-9`	[-1, 100]	0
`LinearExcitability-10`	[-1, 100]	160,000
`fc_layer-11`	[-1, 100]	0
`fromZ`		
`LinearExcitability-12`	[-1, 1600]	161,600
`ReLU-13`	[-1, 1600]	0
`fcD`		
`LinearExcitability-14`	[-1, 1600]	2,561,600
`ReLU-15`	[-1, 1600]	0
`fc_layer-16`	[-1, 1600]	0
`LinearExcitability-17`	[-1, 30000]	48,001,600
`Sigmoid-18`	[-1, 30000]	0
`fc_layer-19`	[-1, 30000]	0
Total parameters:	101,649,308	
Estimated Total Size (MB):	393.08	

V. ADDITIONAL RESULTS: MNIST AND CIFAR-10 EXPERIMENTS

We also evaluate the proposed LGR approach on popular vision benchmarks of MNIST and CIFAR-10. While MNIST evalutes the model on simpler data setting with a lower number of samples, CIFAR-10 consists of complex data settings (real-world images) albeit with the same size as that of MNIST.

MNIST: The MNIST dataset consists of 70000 (32×32) grayscale images of handwritten digits between $0-9$, split into 60000 training images and 10000 test images. Each of the 10 classes consists of 6000 training images and 1000 test images, making it a perfectly balanced dataset.

CIFAR-10: The CIFAR-10 datasetCIFAR-10 constitutes real-world images of objects such as airplanes, automobiles, ships and animals such as dogs, cats and birds amongst others classified in to 10 different classes. Each class consists of 6000 $(32 \times 32 \times 3)$ RGB images with 5000 images used for training and 1000 images constituting the test-set. For our evaluations we convert these images to grayscale before training the models.

Due to the small size of the images for both MNIST and CIFAR-10, instead of using using 'heavy' pre-trained backbones such as VGG-16, we use a simpler *LeNet*-based model as the root and pre-train it on an *unseen* split of the train data by setting aside 10000 images from the training set and use the other 50000 for training the Continual Learning (CL) models. For fairness, this is done for both DGR and LGR ensuring that none of the approaches incur an additional computational and memory cost of having to train the *root* from scratch.

A. Hyper-parameter Optimisation

Latent Generative Replay (LGR) is parameterised by the size of the generator, more specifically, the number of hidden units G_{FC} and the output size of the generator G_{OUT}. To optimise for model accuracy and resource-consumption, we run the Task-IL experiments with the MNIST dataset setting $G_{FC} = G_{OUT} \in [10, 20, 50, 100, 200, 300, 400]$. Varying the size of the generator introduces a trade-off between accuracy and GPU memory consumption (see Figure 6). As the generator gets smaller, GPU memory consumption reduces but so does the average accuracy of the model.

A lighter generative model may become incapable of generating representative samples and fail to mitigate catastrophic forgetting, effectively. Even though we illustrate the change in GPU memory consumption in Figure 6, a *lighter* generator also tends to decrease other performance metrics such as training time and GPU usage as the generative model becomes less computationally expensive. The trade-

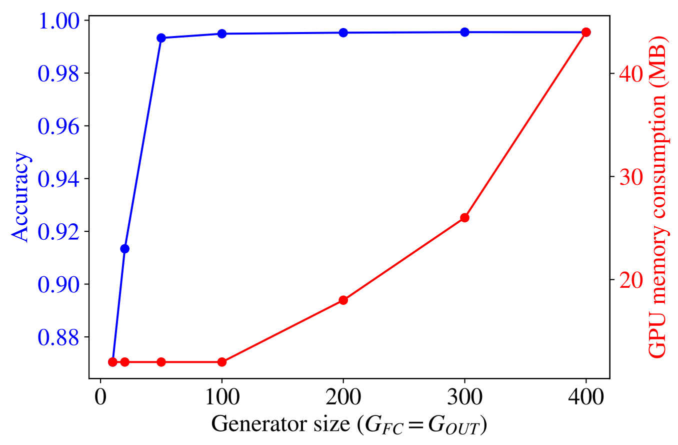

Fig. 6. Accuracy vs. Generator Size trade-off.

off the generator size introduces is thus not only between accuracy and GPU memory consumption, but more generally between accuracy and resource consumption. Thus, selecting the "optimal" generator size comes down to balancing the two objectives.

TABLE X
SELECTED HYPER-PARAMETERS FOR MNIST AND CIFAR-10 DATASETS.

Method	Hyper-parameters	
NR	$B_{size} = 1000$	
LR	$B_{size} = 1000$	
DGR	$G_{FC} = 400$	$G_{OUT} = 1024$
DGR+d	$G_{FC} = 400$	$G_{OUT} = 1024$
LGR	$G_{FC} = 128$	$G_{OUT} = 128$
LGR+d	$G_{FC} = 128$	$G_{OUT} = 128$

We set $G_{FC} = G_{OUT} = 128$ for LGR for MNIST experiments. Other hyper-parameters for all the compared approaches can be seen in Table X. For DGR and DGR+d, we follow the recommendations from van de Ven *et al.* 2019. For the rehearsal methods (NR and LR), we use a buffer size (B_{size}) of 1000, which has been shown to perform well for these methods. We use the same model parameters for the CIFAR-10 experiments as well owing to the similar scale of the datasets.

B. Results

1) MNIST: MNIST results for Task-IL can be seen in Table XI where we see that LGR and LGR+d improve on the accuracy scores of DGR and DGR+d, respectively, albeit marginally. More importantly, LGR-based methods outperform their DGR variants on memory and resource-efficiency metrics. LGR and LGR+d reduce the training time required, CPU and GPU usage as well significantly reduce the GPU memory consumption of DGR and DGR+d, respectively. However, LGR+d witnesses a slight increase in RAM usage. LGR and LGR-d report the lowest GPU memory consumption of all the approaches reducing it by $\approx 0.8\%$ compared to the *lightest* LR approach. Meanwhile, LR and NR emerge as more efficient options in terms of computation, overall, reporting lower training times as well as lower CPU and GPU usage (in $\%$) than pseudo-rehearsal methods.

TABLE XI
MNIST RESULTS FOR TASK-IL. BEST RESULTS FOR EACH ROW ARE IN **BOLD** WHILE SECOND-BEST ARE IN [BRACES].

	NR	**LR**	**DGR**	**LGR**	**DGR+d**	**LGR+d**
Accuracy (%) ▲	98.82	98.86	99.16	[*99.13*] ▲	99.42	**99.49** ▲
Training time (s) ▼	[*48.10*]	**44.33**	62.66	61.36 ▼	64.20	61.23 ▼
RAM Usage (MB) ▼	**2312.49**	[*2315.63*]	2316.18	2315.89 ▼	2316.95	2319.30 ▲
CPU Usage (s) ▼	[*76.27*]	**72.63**	92.16	90.55 ▼	93.68	90.34 ▼
GPU Usage (%) ▼	[*8.83*]	**6.06**	16.00	10.52 ▼	15.23	11.01 ▼
GPU Memory (MB) ▼	1039.66	[*963.00*]	991.00	**955.00** ▼	991.00	**955.00** ▼

2) CIFAR-10 Results: Task-IL experiments with CIFAR-10 (see Table XII) show that the LGR+d approach outperforms all others in terms of model accuracy. Furthermore, LGR-based methods outperform their DGR variants on all the metrics not only improving on model performance but also reducing the memory and resource consumption of these methods. Overall, NR and LR methods remain consistent with LR requiring the least training time and the least CPU and GPU usage (in $\%$), while LGR-based methods offer significant savings in RAM and GPU Memory consumption.

TABLE XII
CIFAR-10 RESULTS FOR TASK-IL. BEST RESULTS FOR EACH ROW ARE HIGHLIGHTED IN **BOLD**.

	NR	**LR**	**DGR**	**LGR**	**DGR+d**	**LGR+d**
Accuracy (%) ▲	[*71.15*]	66.39	66.25	67.47 ▲	68.48	**72.71** ▲
Training time (s) ▼	[*78.87*]	**74.37**	114.53	103.97 ▼	115.67	104.30 ▼
RAM Usage (MB) ▼	3468.45	[*3467.73*]	3499.95	**3467.40** ▼	3500.97	3481.04 ▼
CPU Usage (s) ▼	[*135.87*]	**130.91**	169.35	158.56 ▼	169.94	159.38 ▼
GPU Usage (%) ▼	[*7.88*]	**6.46**	26.00	12.09 ▼	25.44	11.88 ▼
GPU Memory (MB) ▼	1116.33	[*1002.33*]	1379.00	**993.00** ▼	1379.00	**993.00** ▼

C. Model Architectures

1) LeNet-Based Solver: For the MNIST and CIFAR-10 evaluations, the solver is implemented using a LeNet-based architecture (see Figure 7). Input to the image consists of (32×32) grayscale images that are passed to 2 convolutional (*conv*) layers followed by a max pooling layer. The extracted features are then flattened out and passed through a fully-connected layer to extract the latent feature representations. These latent representations are then passed through the classifier consisting of 2 fully-connected layers.

The model is divided into two parts with the *conv* layers along with the latent representation layer forming the *root* of the solver with the rest of the network forming the *top* of the network adapting to task-specific learning. Table XIII presents the implementation summary of the model along with the number of parameters as well as the estimated model size (in MB).

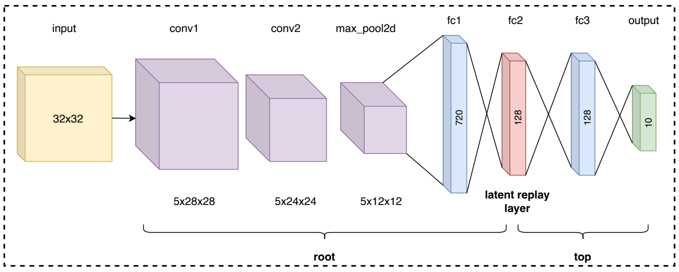

Fig. 7. LeNet-based CNN Implementation.

TABLE XIII
SUMMARY OF THE LENET-BASED CONVOLUTIONAL NEURAL NETWORK (CNN) ARCHITECTURE.

Layer	Output Shape	Number of Parameters
`Conv2d-1`	[-1, 5, 28, 28]	130
`Conv2d-2`	[-1, 5, 24, 24]	630
`Dropout-3`	[-1, 5, 12, 12]	0
`Linear-4`	[-1, 128]	92,288
`Linear-5`	[-1, 128]	16,512
`Dropout-6`	[-1, 128]	0
`Linear-7`	[-1, 10]	1,290
Total parameters:	110,850	
Estimated Total Size (MB):	0.49	

2) *Variational Autoencoder (VAE):* For pseudo-rehearsal, both in DGR and LGR-based methods, we use the Variational Autoencoder (VAE) architecture as the *generator* model for the *scholar*. Based on the image size (32×32 for MNIST/CIFAR-10) or the size of the latent representation layer (128 for LeNet-based solver), the generator uses several fully-connected layer to reconstruct the latent representations. The generator is split into 4 sub-modules representing the encoder (`fcE`), the decpder (`fcD`, network layers extracting the latent variable z (`toZ`) and network layers sampling from the latent variable z (`fromZ`).

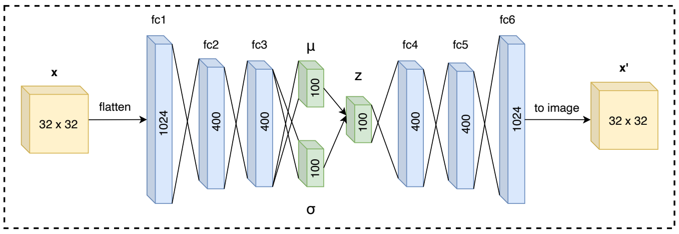

Fig. 8. VAE implementation for the *generator* for MNIST/CIFAR-10.

The VAE-based generator used for reconstructing 32×32 MNIST/CIFAR-10 samples for the DGR-based methods is illustrated in Figure 8 while Table XIV presents the implementation summary of the model along with the number for parameters and estimated model size (in MB).

Table XV presents the implementation summary of the VAE-based generator used for reconstructing 128d latent representations for the MNIST/CIFAR-10 samples for the LGR-based methods along with the number for parameters and estimated model size (in MB).

TABLE XIV
SUMMARY OF THE VAE-BASED GENERATOR FOR DGR FOR MNIST/CIFAR-10.

Layer	Output Shape	Number of Parameters
`fcE`		
`Flatten-1`	[-1, 1024]	0
`LinearExcitability-2`	[-1, 400]	410,000
`ReLU-3`	[-1, 400]	0
`fc_layer-4`	[-1, 400]	0
`LinearExcitability-5`	[-1, 400]	160,400
`ReLU-6`	[-1, 400]	0
`fc_layer-7`	[-1, 400]	0
`toZ`		
`LinearExcitability-8`	[-1, 100]	40,100
`fc_layer-9`	[-1, 100]	0
`LinearExcitability-10`	[-1, 100]	40,000
`fc_layer-11`	[-1, 100]	0
`fromZ`		
`LinearExcitability-12`	[-1, 400]	40,400
`ReLU-13`	[-1, 400]	0
`fcD`		
`LinearExcitability-14`	[-1, 400]	160,400
`ReLU-15`	[-1, 400]	0
`fc_layer-16`	[-1, 400]	0
`LinearExcitability-17`	[-1, 1024]	410,624
`Sigmoid-18`	[-1, 1024]	0
`fc_layer-19`	[-1, 1024]	0
Total parameters:	1,261,924	
Estimated Total Size (MB):	4.88	

TABLE XV
SUMMARY OF THE VAE-BASED GENERATOR FOR LGR FOR MNIST/CIFAR-10.

Layer	Output Shape	Number of Parameters
`fcE`		
`Flatten-1`	[-1, 128]	0
`LinearExcitability-2`	[-1, 128]	16,512
`ReLU-3`	[-1, 128]	0
`fc_layer-4`	[-1, 128]	0
`LinearExcitability-5`	[-1, 128]	16,512
`ReLU-6`	[-1, 128]	0
`fc_layer-7`	[-1, 128]	0
`toZ`		
`LinearExcitability-8`	[-1, 100]	12,900
`fc_layer-9`	[-1, 100]	0
`LinearExcitability-10`	[-1, 100]	12,800
`fc_layer-11`	[-1, 100]	0
`fromZ`		
`LinearExcitability-12`	[-1, 128]	12,928
`ReLU-13`	[-1, 128]	0
`fcD`		
`LinearExcitability-14`	[-1, 128]	16,512
`ReLU-15`	[-1, 128]	0
`fc_layer-16`	[-1, 128]	0
`LinearExcitability-17`	[-1, 128]	16,512
`Sigmoid-18`	[-1, 128]	0
`fc_layer-19`	[-1, 128]	0
Total parameters:	105,966	
Estimated Total Size (MB):	0.40	