Continual Learning of 3D Point Cloud Generators

Michał Sadowski[1] ✉, Karol J. Piczak[1] ✉, Przemysław Spurek[1], and Tomasz Trzciński[2,1,3]

[1] Jagiellonian University
[2] Warsaw University of Technology
[3] Tooploox
m.sadowski@doctoral.uj.edu.pl, {karol.piczak, przemyslaw.spurek}@uj.edu.pl, tomasz.trzcinski@pw.edu.pl

Abstract. Most continual learning evaluations to date have focused on fully supervised image classification problems. This work for the first time extends such an analysis to the domain of 3D point cloud generation, showing that 3D object generators are prone to catastrophic forgetting along the same vein as image classifiers. Classic mitigation techniques, such as regularization and replay, are only partially effective in alleviating this issue. We show that due to the specifics of generative tasks, it is possible to maintain most of the generative diversity with a simple technique of uniformly sampling from different columns of a progressive neural network. While such an approach performs well on a typical synthetic class-incremental setup, more realistic scenarios might hinder strong concept separation by shifting task boundaries and introducing class overlap between tasks. Therefore, we propose an autonomous branch construction (ABC) method. This learning adaptation relevant to parameter-isolation methods employs the reconstruction loss to map new training examples to proper branches of the model. Internal routing of training data allows for a more effective and robust continual learning and generation of separate concepts in overlapping task setups.

Keywords: continual learning · 3D point clouds · generative models · reconstruction loss

1 Introduction

Catastrophic forgetting is a well-known phenomenon of incremental training. Numerous studies have been performed in supervised image classification problems [18,5,10] showing that models trained on a sequence of disjoint tasks lose their discriminative capability very rapidly. This decline is especially profound when the task identity is not accessible during evaluation (*class-incremental learning*).

As continual learning of generative models is still a scarcely researched subject, most works have concentrated on images. In this paper, we shift our focus to generative models of 3D point clouds trained incrementally on an object reconstruction task. This problem still requires a certain level of complexity from

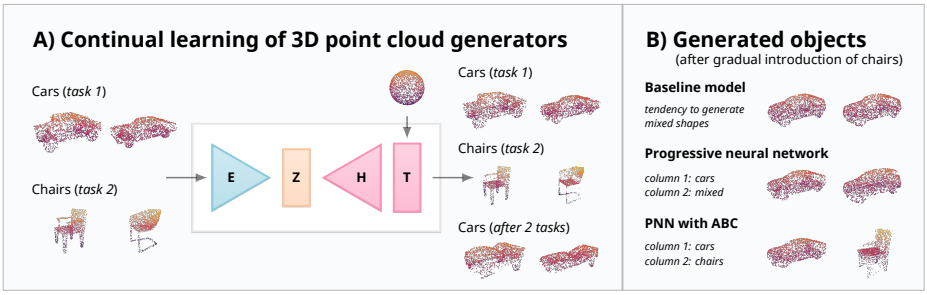

Fig. 1: *Panel A* summarizes continual learning of *HyperCloud*-based point cloud generators. A convolutional encoder (E) converts training examples into a latent representation (Z) that is used by a hypernetwork decoder (H) to create dedicated networks (T) transforming uniform spheres into the original shapes. After introducing a new class, the ability to generate instances of the old class is quickly forgotten. *Panel B* shows examples of objects generated in the *gradual introduction* task setup. Autonomous branch construction helps maintain concept separation between columns of a progressive neural network.

the model but is much easier to evaluate than its image counterparts relying on the FID [8] metric.

Our goal is to create a generator that will provide us with a diverse pool of objects based on all the classes in the training dataset. As the number of encountered classes grows, we expect that introducing new concepts will not require a complete re-training of the model. Unfortunately, in contrast to reconstruction tasks selected by Thai et al. [26], our experiments show that point cloud generators trained sequentially with new classes rapidly lose their ability to generate examples of previously seen objects, similarly to their image counterparts.

A typical approach to combat the loss of previous knowledge is to use various regularization techniques to maintain model parameters close to their values suitable for previous tasks. Another even more effective solution is to support the model with some form of memory. For this purpose, we can either use a memory buffer replaying exact samples from the past or an auxiliary generative model. Unfortunately, classic regularization and memory-based techniques prove to be only partially effective in alleviating catastrophic forgetting in point cloud generators.

Therefore, our first proposition is to apply a simple and effective mechanism that maintains most of the model's generative diversity by sampling generated objects uniformly from all the columns of a *progressive neural network* [21]. This approach can also be employed with other similar parameter isolation techniques, such as PackNet [17] or SupSup [27].

This sampling procedure performs well on a typical class-incremental setup where task boundaries are clearly defined, and there is no overlap between classes in different training exposures. Such an assumption might not hold in more realistic scenarios, where new concepts can appear gradually, and examples of

previously seen classes can intermix with the current task. While more realistic task definitions are a still-developing area of research in continual learning, a couple of papers have proposed methods more suitable for such setups. However, they mainly concentrate on image tasks and classification problems. In our case, we can propose a different solution, tailored explicitly for generative tasks involving a reconstruction loss.

As our main contribution, we propose an adaptation to the learning procedure, dubbed ABC (*autonomous branch construction*), applicable to various parameter isolation techniques which create separate branches of the model. This method allows for effective learning of new tasks with unknown overlap with previously seen classes by dynamically matching training examples with the most corresponding model branch.

2 Related work

2.1 Point cloud generation

Although a 3D point cloud has a seemingly simple structure, its generation with deep learning is challenging. Gadelha et al. [7] and Stypułkowski et al. [25] introduced methods based on variational and adversarial auto-encoders, which are trained by directly optimizing the chamfer (CD) or earth mover's distance (EMD) between two point sets, producing a fixed number of points for each shape. PointFlow, a probabilistic framework proposed by Yang et al. [29], uses a continuous normalizing flow for the distribution of shapes and points given a shape for point cloud generation. Spurek et al. [24] introduced the HyperCloud model based on a hypernetwork, which returns weight definitions for a second network. This object-specific target network then transforms a uniform 3D ball into a given shape. The advantage of this approach lies in the possibility of generating a fully adjustable number of points for each shape, effectively parametrizing its surface. For this reason, we choose HyperCloud as our baseline model.

2.2 Continual learning for generative models

Works in this area focus mainly on GAN and VAE models trained on MNIST, Fashion MNIST, SVHN, and CIFAR10. Several authors [22,16,28] apply EWC [13] and generative replay [23] showing good results from the latter, provided that the dataset is simple enough.

A different approach in VAE models is latent regularization. Achille et al. [2] introduce a VASE model for learning nonoverlapping disentangled representations with an auxiliary classifier inferring the most likely environment and a replay feedback loop. BooVAE [14] learns the approximation of the aggregated posterior as a prior for each task, while Keng et al. [11] present a similar approach with maximization of the mutual information between classes and latent variables during training.

2.3 Unsupervised continual learning

Several papers that have tackled continual learning with more diverse task setups and unsupervised learning show various resemblances to our approach. Lee et al. [15] describe a model expansion approach for task-free setups. However, their problem is multi-class and requires label knowledge. Khare et al. [12] introduce a similar method for routing training data, though focusing on classification tasks. Abati et al. [1] use task-specific gating modules inferring the task identity with an auxiliary task classifier. Continual unsupervised representation learning [20] also shares many similarities with our work, but it requires deep generative replay to mitigate forgetting in the shared representation. De Lange & Tuytelaars [4] use dynamic memory partitioning for replaying prototypes. The importance of task data distribution has been described by Hsu et al. [9].

3 Our method

In this section, we first characterize continual learning scenarios considered in our work, motivating why a typical sequence of disjoint tasks might be overly simplifying. We follow with a description of progressive neural networks, showing how we can use them in continual learning of point cloud generators, and noting their potential weaknesses in more elaborate task setups. Finally, we introduce our method that can mitigate these issues.

3.1 Continual learning setting

There are many scenarios of continual learning experiments. Most of them are designed specifically for supervised classification problems. Only class-incremental learning (defined in [10]) is compatible with generative models without the need to impose additional constraints. It can be easily applied by training a model sequentially on separate classes chosen from the dataset. Although it perfectly demonstrates the phenomenon of catastrophic forgetting, such a setup may oversimplify the problem when applying parameter isolation techniques, which might easily use different model branches for generative purposes.

Moreover, it is unlikely that we will encounter a perfect separation between exposures to different classes in real-world problems. More realistic scenarios should assume the potential for classes to overlap and repeat. Examples of such more complex data splits for continual learning were introduced in [6,9]. We follow this approach, and besides standard disjoint class-incremental learning, we experiment on tasks with shifted boundaries, task repetition, and gradual class introduction according to the Dirichlet distribution.

3.2 Parameter isolation techniques

One of the most common continual learning parameter isolation methods is the use of progressive neural networks [21]. This approach works at the architectural

level by incorporating agents that learn a series of tasks and transfer knowledge to improve convergence speed. For each new task, we instantiate a new column (a neural network) and freeze the weights of the rest of the model. In order to prevent the model from overgrowing, we can decrease layer sizes in consecutive columns. Knowledge transfer is enabled via lateral connections to the corresponding layers of previously learned columns. This aspect differentiates progressive neural networks from simply learning an ensemble of separate models, allowing for faster knowledge acquisition through forward transfer. A progressive neural network starts with only one column. When training of the first task completes, this column is frozen. Then, we add a second column. During training, both columns process the input. In consecutive tasks, we add new columns analogically. Progressive neural networks are the most effective when the task identity is known during the testing phase because it is possible to choose an appropriate column to process the data. We assume that such information is not available in our experiments. However, as our goal is not to classify point clouds but to generate a diverse pool of samples from the model in the testing phase, we propose to sample an equal number of 3D shapes from each column.

As long as each column learns a single class, this approach performs remarkably well. However, it deteriorates when we move to tasks in which classes can overlap. Similar to joint training, it shifts the distribution of the generated objects in the direction of a shape averaged between classes. To disentangle class information, we propose autonomous branch construction.

3.3 Autonomous branch construction

We introduce autonomous branch construction (ABC) as an extension for parameter isolation methods in generative models. The general idea of ABC can be described as routing of training examples based on a thresholded reconstruction loss. This allows for effective learning of new tasks with unknown overlap with previously seen classes.

More specifically, we split each training iteration into two steps: parameter selection and restricted model weights update.

During the parameter selection step, we pass the whole input batch through all the branches (columns in PNN nomenclature) of the model and calculate reconstruction loss for each one of them. Then, we assign each sample to a branch for which it has the lowest loss. This means that training examples are preferably assigned to branches that have already encountered a similar concept. If the loss value exceeds a predefined threshold, we assign them to a new model branch (column). Throughout each task, only one new branch can be created. If the new branch has not been selected with a required frequency (e.g. in task repetition where all concepts can be mapped to previous branches), it is discarded.

In the second phase, we perform backpropagation for standard model loss and update model weights. However, each reconstructed sample influences only the branch assigned to it in the parameter selection step. It means that each branch of the model specializes in specific class reconstruction and generation.

4 Evaluation protocol

4.1 Point cloud generation

We evaluate our continual learning setups on a HyperCloud [24] neural network - a state-of-the-art model for generating 3D point clouds. Its architecture consists of three neural networks working in tandem, as depicted in Figure 1a. The encoder part maps an input point cloud to a lower-dimensional latent space. The hypernetwork decoder maps values from the latent space to a vector of weights, constructing a separate target network for each object. This network models a function $T : \mathbb{R}^3 \longrightarrow \mathbb{R}^3$, which transforms points from the prior distribution (unit uniform ball) to the elements of the given object. Therefore, a target network fully defines a 3D object and can produce any number of points representing its surface.

The whole system can be trained with a reconstruction loss using a selected distance metric (e.g., chamfer or earth mover's distances). In addition to the reconstruction loss, an additional term ensures that the latent values are distributed according to the standard normal density.

In order to validate that the results are not specific to the application of a hypernetwork approach, we repeat the baseline experiment on an analogous model with the decoder mapping latent values directly to 3D shapes (with a predefined number of points).

4.2 Model evaluation

As our main evaluation metric we choose Jensen-Shannon Divergence (JSD) which measures the distance between P_r and P_g, the marginal distributions of points in the S_r (set of reference point clouds) and S_g (set of generated point clouds), computed by discretizing the space into 28^3 voxels:

$$\mathrm{JSD}\left(P_r, P_g\right) = \frac{1}{2} D_{KL}\left(P_r \| M\right) + \frac{1}{2} D_{KL}\left(P_g \| M\right) \tag{1}$$

where $M = \frac{1}{2}(P_r + P_g)$ and $D_{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence between two distributions. We expect the JSD for an ideal model to approach 0, although a model always generating the average shape can also achieve a perfect score, which is a drawback of this metric. Therefore, we also support our findings with additional metrics used for assessing the quality of point cloud generators [3,29]: coverage (COV), minimum matching distance (MMD) and 1-nearest neighbor accuracy (1-NNA). These metrics come in two variants employing chamfer (CD) or earth mover's distance (EMD) as a measure of similarity.

Besides parameter isolation methods, for the first time, we evaluate point cloud generators on representative continual learning methods from the other two general families: replay methods and regularization-based methods. As an example of regularization techniques, we employ Elastic Weight Consolidation (EWC) [13], which tries to safeguard values of parameters essential to good performance on prior tasks by assigning a penalty to deviations from their previous value.

For comparison, we also employ an identity importance matrix resulting in L_2 regularization on the distance to old parameters.

In terms of memory-based approaches, we evaluate generative replay [23], which, apart from the main task solving network, uses a generator model to recreate samples of previously seen classes. HyperCloud is a generative model itself, so there is no need to make this distinction. During training, a copy of the main model can generate additional data samples, which are concatenated with each batch of the training data. The ratio r between sampled and real data depends on the desired importance of a new task compared to older tasks. In our experiments, we test three different values 20%, 40%, 60%. For comparison, we also perform experiments with exact replay. Instead of using an old copy of a generative model to generate data, we sample real data of previous tasks from different memory buffer sizes (128 or 512) with a 1:1 or 1:2 ratio of old to new examples.

5 Experiments

5.1 Implementation details

We use HyperCloud architecture as described in [24] where the encoder is a PointNet-like [19] network consisting of 1D convolutional layers with 64, 128, 256, 512, 512 channels, and fully connected layers with 512 and 2048 neurons. The latent dimension is set to 2048. The decoder is a fully connected neural network with layer widths equal to 64, 128, 512, 1024, 2048 and a final layer mapping directly to the target network. The target network is a small multilayer perceptron with 32, 64, 128, and 64 neurons. All networks use ReLU activations.

We train the HyperCloud model with Adam optimizer, setting the learning rate to 0.0001, with standard betas of 0.9 and 0.999, and no weight decay. We utilize chamfer loss for reconstruction, setting the reconstruction loss coefficient to 0.05 and KLD loss coefficient to 0.5. We also define the time span of the progressive unit sphere radius normalization [24] as 100 epochs. The training batch size is set to 64 and 32 for computing validation metrics. We perform validation every 10 epochs for the first 50 epochs of a given task, then decay to a frequency of 25 (past 50 epochs) and 50 (past 100 epochs). Each epoch processes 5000 examples from the current task. The ABC threshold is set at 8.0.

Our main experiment evaluates models trained on point clouds of *cars*, *chairs*, *airplanes* and *tables* from the ShapeNet dataset. Each point cloud has 2048 points. 85% of the objects are selected for training purposes, 5% of instances in the whole dataset are designated for metric evaluation.

We train the first class for 1000 epochs, and subsequent tasks are limited to 500 epochs. For regularization methods, we also perform a hyperparameter sweep over independent encoder and hypernetwork regularization strengths.

5.2 Task setups

In our baseline class-incremental learning setup, each task consists of samples from one class (1: *cars*, 2: *chairs*, 3: *airplanes* and 4: *tables*). In the task repetition

sequence, we train models sequentially on three tasks (1: *cars*, 2: *chairs*, 3: *cars*). The overlapping sequence presents a shift in the task boundary where 30% examples of the last class are still seen in the new task (1: *100% cars*, 2: *30% cars, 70% chairs*). This is akin to baseline training with exact replay. The gradual introduction of new concepts presents new classes with a decreasing frequency derived from the expected value of Dirichlet distribution with $\alpha = 2$ (1: *100% cars*, 2: *69% cars, 31% chairs*). This would have a similar effect as an introduction of a very aggressive replay technique.

5.3 Results

We present detailed results for our evaluation protocols in Table 1. An evolution of the JSD metric in the main experiment can be seen for selected methods in Figure 2.

Baseline training without any mitigation technique shows strong signs of catastrophic forgetting across all metrics. While regularization techniques can influence this process only marginally, memory-based approaches show a much more favorable behavior. A progressive neural network (PNN) effectively maps distinct concepts in a single-class per task scenario. This can be seen in the PNN generating examples only from the last column, which rapidly diverge from the mixed distribution. A uniform sampling technique has an excellent profile that almost matches the best available baseline of joint training on all classes for 2500 epochs.

A significant drawback of progressive neural networks is the cost of model expansion. In a standard setting, each new column has the same size as the original model. In order to reduce this impact, we also evaluate a *decaying PNN* where the first column is of standard size, but each consecutive column has a 50% reduced size compared to the previous column. For a sequence of four tasks, we can safely use such a reduction (from 4 model sizes to 1.875) without impacting the performance.

We also verify additional variants of the baseline model. Replacing the hypernetwork with a standard VAE decoder has no favorable influence on the final metrics, similarly with unsupervised pre-training on 30 separate classes. When compared with a control experiment (additional 1000 epochs on the first task), it seems that slight improvements are only due to the elongated training procedure.

When we introduce task repetition, vanilla PNN still performs much better than the baseline model. However, it loses some of its edge due to a shifting balance in the generated classes. Introducing ABC helps the performance by preventing the model from creating a new column for a repeated concept.

Similar behavior can be seen with last class overlap. The baseline model mostly shifts to the new class with some decrease in forgetting due to the replay characteristic of this setup. Vanilla ABC will learn the new column in the same way, but it maintains a previous column that helps compensate for this setback. In contrast, PNN with ABC is capable of better concept isolation, creating two separate columns, one for each class.

Table 1: Evaluation of models trained sequentially with different continual learning techniques. Generative capability is assessed across four metrics (with either chamfer or earth mover's distances). Cells represent final metric values after training on all tasks and validating on an equal mix of all seen classes. Values for regularization and replay techniques show average results across hyperparameter sweeps.

Results for incremental learning with single-class tasks

Method	**JSD**	**MMD**		**COV%**		**1-NNA%**	
		CD	*EMD*	*CD*	*EMD*	*CD*	*EMD*
Baseline (HyperCloud)	35.3	14.8	16.7	23.8	22.7	80.3	80.5
Baseline (VAE)	41.4	12.5	17.4	23.4	17.2	88.5	91.6
Baseline with pre-training	33.7	15.6	16.6	28.1	26.6	82.2	82.4
Pre-training control	31.7	14.2	16.2	29.3	29.3	79.1	79.9
L_2 regularization	28.6	9.1	14.4	25.1	26.3	91.7	93.9
EWC regularization	28.8	8.9	13.9	28.0	27.0	87.9	88.6
Exact replay	18.1	6.0	**11.6**	41.8	39.8	75.5	77.1
Generative replay	19.2	10.5	15.1	46.5	**48.4**	**69.1**	**72.3**
PNN (*first column*)	27.2	17.4	15.0	21.5	19.9	76.8	81.8
PNN (*last column*)	38.4	15.9	19.7	28.9	27.0	89.3	96.5
PNN (*uniform sampling*)	9.5	**5.4**	12.2	51.2	**48.4**	**69.1**	83.8
Decaying PNN (*uniform*)	**8.0**	5.5	12.2	**52.7**	**48.4**	74.8	82.6
Joint training (best case)	8.0	5.0	10.6	54.7	53.5	63.5	65.0
Ideal metric behavior:	$\rightarrow 0$	$\rightarrow 0$		$\rightarrow 100\%$		$\rightarrow 50\%$	

Results for different task setups

Task repetition							
Baseline (HyperCloud)	15.3	10.0	11.9	39.5	39.5	71.3	69.7
PNN (*uniform sampling*)	7.7	3.5	9.3	**52.3**	**50.0**	64.6	70.3
PNN with ABC (*uniform*)	**3.8**	**3.3**	**8.8**	51.6	**50.0**	**63.1**	**67.4**
30% overlap of the last class							
Baseline (HyperCloud)	9.1	3.9	9.9	42.6	44.9	67.8	70.7
PNN (*uniform sampling*)	8.0	3.5	9.7	50.0	**46.9**	66.4	77.1
PNN with ABC (uniform)	**4.9**	**3.0**	**8.9**	**50.8**	46.1	**58.8**	**70.3**
Gradual introduction of new concepts							
Baseline (HyperCloud)	**6.2**	3.5	**9.0**	48.8	**52.7**	68.2	**69.7**
PNN (*uniform sampling*)	9.2	4.9	10.3	44.1	49.6	66.8	74.6
PNN with ABC (uniform)	6.6	**2.6**	9.1	**49.2**	39.8	**59.6**	77.5

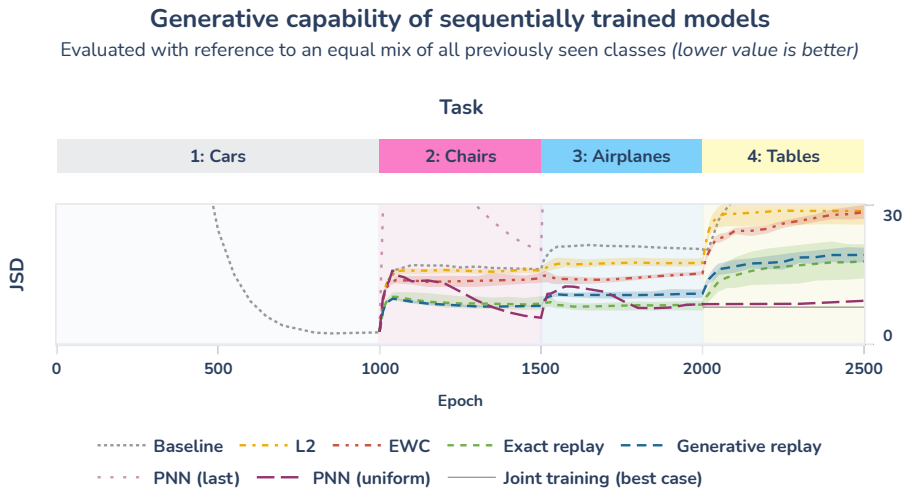

Fig. 2: JSD values (multiplied by 10^2) computed between point clouds generated by respective models and reference examples from the validation set (a balanced mix of all seen classes). Shaded color bands indicate maximal and minimal metric values for different settings of hyperparameters. Best viewed in color.

The most challenging setup for PNN is the gradual introduction of new concepts. This task sequence creates a solid baseline, equivalent to replaying 69% of old examples with an unlimited memory buffer. On a positive note, memory-based techniques will not be able to improve it further. In this case, PNN can only compete with this baseline when using ABC. However, one additional fact in support of ABC can be inferred by looking at the generated objects in Figure 1b. Both the HyperCloud baseline and vanilla PNN tend to create objects that resemble a shape mixed between classes (cars with chair legs instead of wheels), whereas concepts generated from the ABC-augmented model more closely resemble the original classes. The JSD metric does not easily capture this difference.

6 Conclusion

In this work, for the first time, we analyzed 3D point cloud generative models in a continual learning scenario. We extended our protocol to sequences of exposures where task boundaries are not clearly defined alongside a typical disjoint single-class incremental evaluation. We have shown that point cloud generators are prone to catastrophic forgetting, similarly to models trained continually in classification tasks. This phenomenon contrasts with the findings of Thai et al. [26], where continual reconstruction tasks did not exhibit such a behavior. These discrepancies might be introduced through the specifics of task definitions or the type of model used.

We also highlight that in continual learning of generative models, an overlooked potential lies in applying classic parameter isolation techniques, such as progressive neural networks. This family of methods is often discarded in supervised image classification due to the required knowledge of the task identity at test time. A different formulation of a generative problem, where the goal is to create diverse objects from all possible classes irrespective of task identity, creates an opportunity for parameter isolation techniques to outperform other typical approaches. This is achieved by simple uniform sampling from different branches of the model at test time.

We have also introduced an adaptation to the learning procedure, named autonomous branch construction (ABC), which performs a selective mapping of training examples to respective model branches based on their reconstruction loss value. This modification proved to be very effective in tasks where new concepts appear gradually and previously seen examples mix with the current task data. Such a technique could also be applied to other parameter isolation techniques, like models pruned through PackNet [17], to mitigate the impact on model size due to the expanding nature of progressive neural networks.

Acknowledgments

This research was funded by Foundation for Polish Science (grant no POIR.04.04 .00-00-14DE/18-00 carried out within the Team-Net program co-financed by the European Union under the European Regional Development Fund), National Science Centre, Poland (grant no 2020/39/B/ST6/01511) and Warsaw University of Technology (POB Research Centre for Artificial Intelligence and Robotics within the Excellence Initiative Program - Research University). The author has applied a CC BY license to any Author Accepted Manuscript (AAM) version arising from this submission, in accordance with the grants' open access conditions.

References

1. Abati, D., et al.: *Conditional channel gated networks for task-aware continual learning.* IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020.
2. Achille, A., et al.: *Life-long disentangled representation learning with cross-domain latent homologies.* Conference on Neural Information Processing Systems (NeurIPS), 2018.
3. Achlioptas, P., et al.: *Learning representations and generative models for 3D point clouds.* International Conference on Machine Learning (ICML), 2018.
4. De Lange, M., Tuytelaars, T.: *Continual prototype evolution: Learning online from non-stationary data streams.* International Conference on Computer Vision (ICCV), 2021.
5. De Lange, M., et al.: *A continual learning survey: Defying forgetting in classification tasks.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 2021.
6. Deja, K., et al.: *Multiband VAE: Latent space partitioning for knowledge consolidation in continual learning.* arXiv:2106.12196, 2021.

7. Gadelha, M., Wang, R., Maji, S.: *Multiresolution tree networks for 3D point cloud processing*. European Conference on Computer Vision (ECCV), 2018.
8. Heusel, M., et al.: *GANs trained by a two time-scale update rule converge to a local Nash equilibrium*. International Conference on Neural Information Processing Systems (NeurIPS), 2017.
9. Hsu, T.M.H., Qi, H., Brown, M.: *Measuring the effects of non-identical data distribution for federated visual classification*. arXiv:1909.06335, 2019.
10. Hsu, Y.C., et al.: *Re-evaluating continual learning scenarios: A categorization and case for strong baselines*. arXiv:1810.12488, 2018.
11. Kang, W.Y., Han, C.H., Zhang, B.T.: *Discriminative variational autoencoder for continual learning with generative replay*, 2019.
12. Khare, S., Cao, K., Rehg, J.: *Unsupervised class-incremental learning through confusion*. arXiv:2104.04450, 2021.
13. Kirkpatrick, J., et al.: *Overcoming catastrophic forgetting in neural networks*. Proceedings of the National Academy of Sciences (PNAS), **114**(13), 3521–3526, 2017.
14. Kuzina, A., Egorov, E., Burnaev, E.: *BooVAE: A scalable framework for continual VAE learning under boosting approach*. arXiv:1908.11853, 2019.
15. Lee, S., et al.: *A neural Dirichlet process mixture model for task-free continual learning*. International Conference on Learning Representations (ICLR), 2020.
16. Lesort, T., et al.: *Generative models from the perspective of continual learning*. International Joint Conference on Neural Networks (IJCNN), 2019.
17. Mallya, A., Lazebnik, S.: *PackNet: Adding multiple tasks to a single network by iterative pruning*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.
18. Masana, M., et al.: *Class-incremental learning: survey and performance evaluation*. arXiv:2010.15277, 2020.
19. Qi, C.R., et al.: *PointNet: Deep learning on point sets for 3D classification and segmentation*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
20. Rao, D., et al.: *Continual unsupervised representation learning*. Conference on Neural Information Processing Systems (NeurIPS), 2019.
21. Rusu, A.A., et al.: *Progressive neural networks*. arXiv:1606.04671, 2016.
22. Seff, A., et al.: *Continual learning in generative adversarial nets*. arXiv:1705.08395, 2017.
23. Shin, H., et al.: *Continual learning with deep generative replay*. Conference on Neural Information Processing Systems (NeurIPS), 2017.
24. Spurek, P., et al.: *Hypernetwork approach to generating point clouds*. International Conference on Machine Learning (ICML), 2020.
25. Stypułkowski, M., et al.: *Conditional invertible flow for point cloud generation*. arXiv:1910.07344, 2019.
26. Thai, A., et al.: *Does continual learning = catastrophic forgetting?* arXiv:2101.07295, 2021.
27. Wortsman, M., et al.: *Supermasks in superposition*. Conference on Neural Information Processing Systems (NeurIPS), 2020.
28. Wu, C., et al.: *Memory replay GANs: Learning to generate images from new categories without forgetting*. Conference on Neural Information Processing Systems (NeurIPS), 2018.
29. Yang, G., et al.: *PointFlow: 3D point cloud generation with continuous normalizing flows*. IEEE/CVF International Conference on Computer Vision (ICCV), 2019.