Neural Architecture for Online Ensemble Continual Learning

Mateusz Wójcik[1,2] **Witold Kościukiewicz**[1,2] **Tomasz Kajdanowicz**[1] **Adam Gonczarek**[2]
[1]Wroclaw University of Science and Technology [2]Alphamoon Ltd., Wrocław
{mateusz.wojcik,witold.kosciukiewicz,tomasz.kajdanowicz}@pwr.edu.pl
adam.gonczarek@alphamoon.ai

Abstract

Continual learning with an increasing number of classes is a challenging task. The difficulty rises when each example is presented exactly once, which requires the model to learn online. Recent methods with classic parameter optimization procedures have been shown to struggle in such setups or have limitations like non-differentiable components or memory buffers. For this reason, we present the fully differentiable ensemble method that allows us to efficiently train an ensemble of neural networks in the end-to-end regime. The proposed technique achieves SOTA results without a memory buffer and clearly outperforms the reference methods. The conducted experiments have also shown a significant increase in the performance for small ensembles, which demonstrates the capability of obtaining relatively high classification accuracy with a reduced number of classifiers.

1 Introduction

Over the last few years, neural networks have become a widely used and effective tool, especially in supervised learning problems [11, 7, 25]. The parameter optimization process based on a gradient descent works well when the data set is sufficiently large and available entirely during the training process. Otherwise, the catastrophic forgetting [9] will occur, which makes neural networks unable to be trained incrementally. The field of continual learning aims to develop methods that enable the accumulation of new knowledge without forgetting previously inferred one.

Currently, the methods that are guaranteed to be most effective across various tasks utilize a memory buffer [18]. While this is a relatively simple and effective approach, it requires constant access to the data. In many practical real-world applications, this disqualifies such methods due to privacy policies or data size [27]. It has also been shown that methods without a memory buffer are not effective in class incremental [31] setup with classic optimization algorithms like e.g. Adam [22].

In this paper, we present a fully differentiable neural architecture for online class incremental continual learning called DE&E. The architecture is inspired by an Encoders and Ensembles (hereafter referred to as E&E) [28] and adapted to the most challenging task-free online class incremental setup. Our method retains advantages of E&E while increasing its accuracy, reducing forgetting, enables end-to-end ensemble training and significantly improving performance when number of parameters is low (small ensembles). We demonstrate that the proposed architecture achieves SOTA results in evaluated scenarios. In summary, our contributions are as follows: 1) we introduced a differentiable KNN layer [34] into the model architecture, 2) we proposed a novel approach to aggregate classifier predictions in the ensemble, 3) we demonstrate the proposed architecture effectiveness by achieving SOTA results on popular continual learning benchmarks without a memory buffer.

6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans.

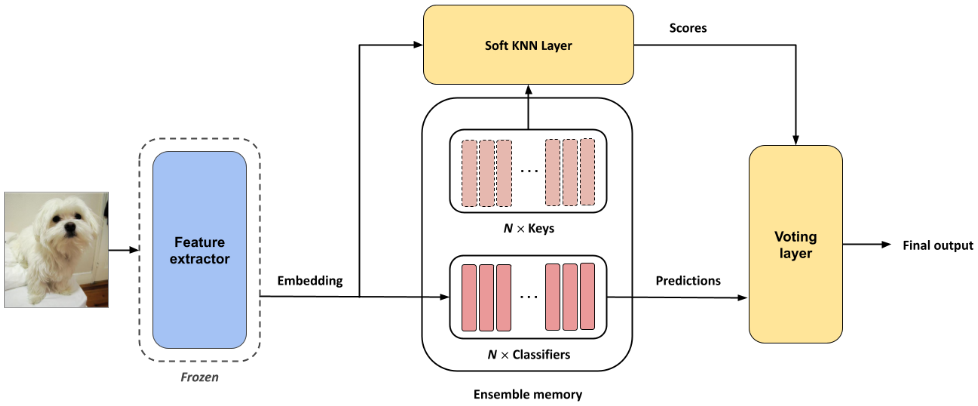


Figure 1: Architecture of the proposed model. Input image is processed by the feature extractor. Obtained embeddings are used to find the most relevant classifiers according to assigned keys. The *soft KNN* layer approximates the *soft KNN* scores. Predictions are weighted in the voting layer by both cosine similarity and *soft KNN* scores. Final output is the class with the highest voting score.

2 Model architecture

Feature extractor. The full model architecture is presented in Figure 1. The first component of the proposed architecture is a multi-layer feature extractor that transforms input data into the embedding space. It can be described by the following mapping $\mathbf{z} = F(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^D$ is an input example and $\mathbf{z} \in \mathbb{R}^M$ is a M-dimensional embedding. The approach we follow assumes the use of a pre-trained model with frozen parameters. Such a procedure makes it possible to completely prevent the extractor from forgetting by isolating feature space learning from the classification process.

Keys and classifiers. We use an ensemble of N classifiers $f_n(\cdot)$, where each of them maps the embedding into a K-dimensional output vector $\hat{\mathbf{y}}_n = f_n(\mathbf{z})$. With each classifier, there is an associated key vector $\mathbf{k}_n \in \mathbb{R}^M$ with the same dimensionality as the embedding. The keys help to select the most suitable models for specialization with respect to the currently processed input example. They are initialized randomly from normal distribution. We use simple single-layer neural networks as classifiers, with fan-in variance scaling as the weight initialization strategy. The network output is activated by a hyperbolic tangent function (*tanh*).

Soft κ-nearest neighbors layer. The standard KNN algorithm is often implemented using ordinary sorting operations that make it impossible to determine the partial derivatives with respect to the input. It removes the ability to use KNN as part of end-to-end neural models. However, it is possible to obtain a differentiable approximation of the KNN model by solving the Optimal Transport Problem [23]. Based on this concept, we add a differentiable layer to the model architecture. We call this layer soft κ-nearest neighbors (*soft KNN*). In order to determine the KNN approximation, we first compute a cosine distance vector $\mathbf{c} \in \mathbb{R}^N$ between the embedding and the keys:

$$c_n = 1 - \cos(\mathbf{z}, \mathbf{k}_n), \tag{1}$$

where $\cos(\cdot,\cdot)$ denotes the cosine similarity. Next, we follow the idea of a soft top-κ operator presented in [34], where κ denotes the number of nearest neighbors. Let $\mathbf{E} \in \mathbb{R}^{N\times 2}$ be the Euclidean distance matrix with the following elements:

$$e_{n,0} = (c_n)^2, \quad e_{n,1} = (c_n - 1)^2. \tag{2}$$

And let $\mathbf{G} \in \mathbb{R}^{N\times 2}$ denote the similarity matrix obtained by applying the Gaussian kernel to $\mathbf{E}$:

$$\mathbf{G} = \exp(-\mathbf{E}/\sigma), \tag{3}$$

where σ denotes the kernel width. The $\exp$ operators are applied elementwise to matrix $\mathbf{E}$.

We then use the Bregman method, an algorithm designed to solve convex constraint optimization problems, to compute L iterations of Bregman projections in order to approximate their stationary points:

$$\mathbf{p}^{(l+1)} = \frac{\boldsymbol{\mu}}{\mathbf{G}\mathbf{q}^{(l)}}, \quad \mathbf{q}^{(l+1)} = \frac{\boldsymbol{\nu}}{\mathbf{G}^{\top}\mathbf{p}^{(l+1)}}, \quad l = 0, \ldots, L-1 \tag{4}$$

where $\boldsymbol{\mu} = \mathbf{1}_N/N$, $\boldsymbol{\nu} = [\kappa/N, (N-\kappa)/N]^{\top}$, $\mathbf{q}^{(0)} = \mathbf{1}_2/2$, and $\mathbf{1}_i$ denotes the i-element all-ones vector. Finally, let $\mathbf{\Gamma}$ denotes the optimal transport plan matrix and is given by:

$$\mathbf{\Gamma} = \mathrm{diag}(\mathbf{p}^{(L)}) \cdot \mathbf{G} \cdot \mathrm{diag}(\mathbf{q}^{(L)}) \tag{5}$$

As the final result $\boldsymbol{\gamma} \in \mathbb{R}^N$ of the soft κ-nearest neighbor operator, we take the second column of $\mathbf{\Gamma}$ multiplied by N i.e. $\boldsymbol{\gamma} = N\mathbf{\Gamma}_{:,2}$. $\boldsymbol{\gamma}$ is a soft approximation of a zero-one vector that indicates which κ out of N instances are the nearest neighbors. Introducing the *soft KNN* enables us to train parts of the model that were frozen until now (A.4.2).

Voting layer. We use both c_n and $\boldsymbol{\gamma}$ to weight the predictions by giving the higher impact for classifiers with keys similar to extracted features. The obtained approximation $\boldsymbol{\gamma}$ has two main functionalities. It eliminates the predictions from classifiers outside κ nearest and weights the result. Since the Bregman method does not always completely converge, the vector κ contains continuous values that are close to 1 for the most relevant classifiers. We make use of this property during the ensemble voting procedure. The higher the κ value for a single classifier, the higher its contribution toward the final ensemble decision. The final prediction is obtained as follows:

$$\hat{\mathbf{y}} = \frac{\sum_{n=1}^{N} \gamma_n c_n \hat{\mathbf{y}}_n}{\sum_{n=1}^{N} c_n} \tag{6}$$

Training To effectively optimize the model parameters, we follow the training procedure presented in [28]. It assumes the use of a specific loss function that is the inner product between the ensemble prediction and the one-hot coded label:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\mathbf{y}^{\top}\hat{\mathbf{y}} \tag{7}$$

Optimizing this criterion yields an advantage of using a *tanh* activation function, significantly reducing catastrophic forgetting. Following the reference method, we also use an optimizer that discards the value of the gradient and uses only its sign to determine the update direction. As a result, the parameters are being changed by a fixed step during the training.

3 Experiments

Setup

Results. The results of the evaluation on MNIST and CIFAR-10 are presented in Table 1. For all setups evaluated, our model performed best improving results of main reference method (E&E) up to 6%. We can also see a significant difference in achieved accuracy between the DE&E approach and baselines. Furthermore, it achieved this results without replaying training examples seen in the past, making it more practical relative to memory based methods (Replay, A-GEM, GEM) with 10 examples stored per experience (one split). For the ensemble of 128 classifiers and MNIST, our architecture achieved results more than 18% better than the best method with a memory buffer.

In addition, we observed that the proposed method significantly improves the performance of small ensembles. The smaller the ensembles, the higher the gain in accuracy. For MNIST and the ensemble of 16 models, the improvement was up to approximately 6% over the E&E. For the 64 classifiers and CIFAR-10, the improvement was about 5%. Figure 2 shows the comparison of the total number of weights of ensembles of different sizes and the achieved classification performance. The proposed method achieves higher results having the same number of parameters. For an ensemble of 1024 classifiers, the accuracy is already very close, suggesting that the gain decreases with large ensembles.

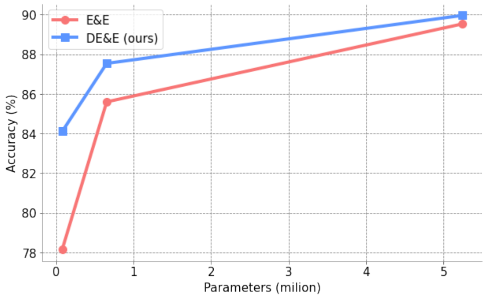


Figure 2: Number of weights in ensembles (16, 128, 1024 classifiers) and achieved accuracy (%) on 10-split MNIST.

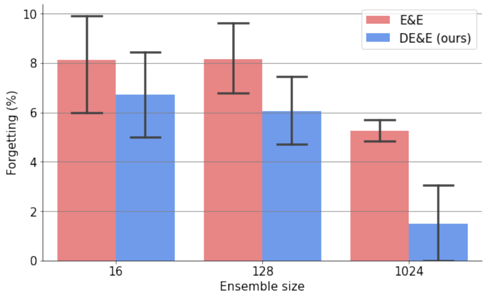


Figure 3: Averaged forgetting rate (the lower the better) for ensembles evaluated on 10-split MNIST.

Table 1: Accuracy (%) and standard deviation for methods evaluated on MNIST and CIFAR-10. For experimental setup details see A.2.

	MNIST (10 splits)		CIFAR-10 (5 splits)	
	$N = 16$	$N = 128$	$N = 64$	$N = 128$
Naive	14.41 ± 5.99	11.63 ± 2.22	19.65 ± 0.33	19.70 ± 0.36
LwF [16]	12.38 ± 3.99	9.88 ± 0.55	19.48 ± 0.55	19.62 ± 0.60
EWC [12]	14.33 ± 4.44	10.97 ± 2.32	19.52 ± 0.29	19.88 ± 0.50
SI [36]	10.18 ± 1.00	17.22 ± 4.64	17.97 ± 2.40	21.32 ± 5.76
CWR* [17]	16.41 ± 5.42	10.38 ± 0.79	18.92 ± 2.97	22.41 ± 2.00
GEM [19] (10 / exp)	67.81 ± 2.61	58.92 ± 6.34	30.75 ± 1.47	29.27 ± 1.46
A-GEM [4] (10 / exp)	53.59 ± 5.21	21.31 ± 15.90	39.86 ± 14.25	36.12 ± 6.19
Replay [5] (10 / exp)	74.49 ± 3.84	69.02 ± 4.90	44.03 ± 3.72	43.82 ± 7.10
E&E [28]	78.16 ± 1.85	85.60 ± 0.52	46.34 ± 1.98	56.24 ± 1.41
DE&E (ours)	$\mathbf{84.19 \pm 1.00}$	$\mathbf{87.54 \pm 0.24}$	$\mathbf{48.78 \pm 1.34}$	$\mathbf{59.36 \pm 0.73}$

An important advantage of the proposed method is a low forgetting rate [3]. We observed significantly reduced forgetting relative to the reference method, as shown in Figure 3. Stronger specialization amplified by the introduced voting method makes classifiers less likely to lose acquired knowledge. The larger the ensemble the relatively less knowledge is forgotten. Empirically, this phenomenon can be explained by the fact that a larger ensemble means better coverage of the key data space, making the models specialize in classifying specific groups of examples. As a result, we protect models against domain shift of input examples, thus making them easier to classify and harder to forget.

4 Conclusions

In this paper, we proposed a neural architecture for online continual learning with training procedure specialized in challenging class incremental problems. The presented architecture introduces a fully differentiable *soft KNN* layer and novel prediction weighting strategy based on the *soft KNN*. This components amplified the influence of most specialized classifiers on the final prediction. As a result, we showed improved accuracy for all of the cases studied and achieved SOTA results. We have shown that it is possible to noticeably improve the quality of classification using the proposed techniques and this effect is observed especially in small ensembles that gained significantly higher performance. As a result, the presented architecture outperforms methods with memory buffer and enables researchers to make further steps towards overrun the current SOTA in class incremental problems.

Acknowledgement

This work was conducted under the Implementation Doctorate programme of Polish Ministry of Science and Higher Education.

References

[1] R. Aljundi, E. Belilovsky, T. Tuytelaars, L. Charlin, M. Caccia, M. Lin, and L. Page-Caccia. Online continual learning with maximal interfered retrieval. *Advances in neural information processing systems*, 32, 2019.

[2] Y. Cao, T. A. Geddes, J. Y. H. Yang, and P. Yang. Ensemble deep learning in bioinformatics. *Nature Machine Intelligence*, 2(9):500–508, 2020.

[3] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 532–547, 2018.

[4] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny. Efficient lifelong learning with a-gem. In *ICLR*, 2019.

[5] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato. Continual learning with tiny episodic memories. *arXiv preprint arXiv:1902.10486, 2019*, 2019.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] T. Doan, S. I. Mirzadeh, J. Pineau, and M. Farajtabar. Efficient continual learning ensembles in neural network subspaces. *arXiv preprint arXiv:2202.09826*, 2022.

[9] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[10] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] J. Kirkpatrick, R. Pascanu, N. C. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

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

[14] S. Lee, J. Ha, D. Zhang, and G. Kim. A neural dirichlet process mixture model for task-free continual learning. *CoRR*, abs/2001.00689, 2020.

[15] Y. Li and Y. Pan. A novel ensemble deep learning model for stock prediction based on stock prices and news. *International Journal of Data Science and Analytics*, 13(2):139–149, 2022.

[16] Z. Li and D. Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[17] V. Lomonaco and D. Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR, 2017.

[18] V. Lomonaco, L. Pellegrini, P. Rodríguez, M. Caccia, Q. She, Y. Chen, Q. Jodelet, R. Wang, Z. Mai, D. Vázquez, G. I. Parisi, N. Churamani, M. Pickett, I. H. Laradji, and D. Maltoni. CVPR 2020 continual learning in computer vision competition: Approaches, results, current challenges and future directions. *CoRR*, abs/2009.09929, 2020.

[19] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6470–6479, Red Hook, NY, USA, 2017. Curran Associates Inc.

[20] Z. Mai, R. Li, J. Jeong, D. Quispe, H. Kim, and S. Sanner. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, 2022.

[21] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[22] S. I. Mirzadeh, M. Farajtabar, R. Pascanu, and H. Ghasemzadeh. Understanding the role of training regimes in continual learning. *Advances in Neural Information Processing Systems*, 33:7308–7320, 2020.

[23] G. Peyré, M. Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[24] T. Plötz and S. Roth. Neural nearest neighbors networks. *Advances in Neural information processing systems*, 31, 2018.

[25] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

[26] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[27] A. Salem, Y. Zhang, M. Humbert, M. Fritz, and M. Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. *CoRR*, abs/1806.01246, 2018.

[28] M. Shanahan, C. Kaplanis, and J. Mitrovic. Encoders and ensembles for task-free continual learning. *CoRR*, abs/2105.13327, 2021.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[30] S. Uddin, I. Haque, H. Lu, M. A. Moni, and E. Gide. Comparative performance analysis of k-nearest neighbour (knn) algorithm and its different variants for disease prediction. *Scientific Reports*, 12, 2022.

[31] G. M. Van de Ven and A. S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

[32] Y. Wen, D. Tran, and J. Ba. Batchensemble: an alternative approach to efficient ensemble and lifelong learning. *arXiv preprint arXiv:2002.06715*, 2020.

[33] M. Wortsman, G. Ilharco, S. Y. Gadre, R. Roelofs, R. Gontijo-Lopes, A. S. Morcos, H. Namkoong, A. Farhadi, Y. Carmon, S. Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. *arXiv preprint arXiv:2203.05482*, 2022.

[34] Y. Xie, H. Dai, M. Chen, B. Dai, T. Zhao, H. Zha, W. Wei, and T. Pfister. Differentiable top-k with optimal transport. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20520–20531. Curran Associates, Inc., 2020.

[35] Y. Yang, H. Lv, and N. Chen. A survey on ensemble learning under the era of deep learning. *arXiv preprint arXiv:2101.08387*, 2021.

[36] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

[37] D. Zoran, B. Lakshminarayanan, and C. Blundell. Learning deep nearest neighbor representations using differentiable boundary trees. *CoRR*, abs/1702.08833, 2017.

A Appendix

A.1 Code

Code is currently available in Github repository (`https://github.com/mateusz-wojcik-97/neural-architecture-for-online-ensemble-cl`).

A.2 Implementation details.

We use PyTorch to both reproduce the E&E results and implement the DE&E method. We use a pre-trained ResNet-50 model as the feature extractor for the CIFAR-10 data set. The model is available in the following GitHub repository, `https://github.com/yaox12/BYOL-PyTorch`, and is used under MIT Licence. For MNIST, we trained a variational autoencoder on the Omniglot data set. We based our implementation of the *soft KNN* layer on the code provided with `https://proceedings.neurips.cc/paper/2020/hash/ec24a54d62ce57ba93a531b460fa8d18-Abstract.html`. All data sets used are public.

Data sets. For model evaluation, we used two popular data sets: MNIST, CIFAR-10 and we also perform some additional experiments using the CIFAR-100 dataset (Sections A.3, A.4.3). The selected data sets are characterized by varying difficulty. MNIST provides images that are significantly easier to classify due to their simple structure. In contrast, CIFAR-10 data set contain slightly larger, color images and provides 10 classes. Each data set was tested using two commonly used configurations: The first one covered a class incremental scenario where each class appears separately (one at a time). The second scenario involved multiple classes appearing at once. Depending on the data set, these configurations varied - for MNIST and CIFAR-10, the 10-split and 5-split approaches were evaluated. By split we mean into how many parts the data set labels were divided. It is worth noting that the 5-split approach can be treated as task-incremental by default when task-id is provided [31]. However, during our research it was taught without providing a task identifier to the model (but we conventionally refer to them as tasks). Such a procedure makes the task even more complicated, because the choice must always be made between all classes.

Feature extractor. To obtain the features for MNIST, a variational autoencoder (VAE) was trained on the Omniglot data set [13]. The VAE architecture follows the one introduced with the E&E method. The VAE feature vector has a size of 512. The training setup is presented in Table 2. To train the VAE, we used the standard reconstruction loss,

$$||\mathbf{x} - \hat{\mathbf{x}}||^2 + \beta KL(\mathbf{z}, \mathcal{N}(0, \mathbf{I})) \tag{8}$$

where $\hat{\mathbf{x}}$ is the decoder output, $\mathcal{N}(0, \mathbf{I})$ is the standard normal distribution, and β is the regularization term for latent loss. We trained the model until the VAE reconstruction loss exceeded the fixed threshold. After training was complete, we froze the encoder weights and extracted them from the trained VAE. Examples of image reconstruction using the trained model are shown in Figure 4.

Table 2: Hyperparameters of the VAE trained on the Omniglot dataset.

Hyperparameter	Value
Learning rate	0.001
Encoding size	512
Input size	28
Batch size	48
Loss threshold	0.02
β	0.001

In order to extract high quality features from CIFAR-10, it was necessary to train a more complex model beforehand. The Resnet-50 model [11] trained on ImageNet data [6] was used. It produces a feature vector with a size 2048. The four-times larger feature vector is necessary to provide quality features for classifying images from such data sets. The BYOL technique [10] was utilized in order to obtain more informative embeddings. BYOL is an approach dedicated to learning

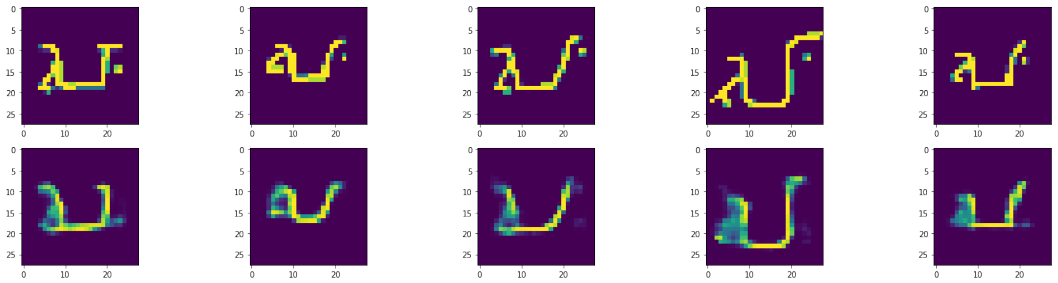


Figure 4: Examples of image reconstruction using a trained VAE. The top row contains the original images from Omniglot while the bottom row shows the reconstructions.

image representations using a self-supervised approach that involves the use of two neural networks. The first, called an online network, learns to predict the representation generated by the second network, called a target. Both networks receive the same image as an input, but with two different transformations applied. The contrastive loss is minimized and the target network parameters are updated as the slow-moving average of the online network.

Baselines. We use Naive, LwF [16], EWC [12], SI [36], CWR* [17], GEM [19], A-GEM [4] and Replay [5] approaches as baselines to compare with our method. We utilize the implementation from Avalanche (`https://avalanche.continualai.org/`), a library designed for continual learning tasks. The main purpose of this comparison was to determine how the proposed method performs against classical approaches and, in particular, against the methods with memory buffer, which gives a significant advantage in class incremental problems. The recommended hyperparameters for each baseline method vary across usages in literature, so we chose them based on our own internal experiments. For a clarity, we keep hyperparameter naming nomenclature from the Avalnache library. For EWC we use $lambda = 10000$. The LwF model was trained with $alpha = 0.15$ and $temperature = 1.5$. For SI strategy we use $lambda = 5e7$ and $eps = 1e - 7$. The hyperparameters of the memory based approach GEM were set as follows: $memory\_strength = 0.5$, $patterns\_per\_exp = 10$, which implies that with every experience (split), 10 examples will we accumulated. This has a particular importance when the number of classes is large. With this setup and 10-split MNIST, memory contains 100 examples after training on all classes. Having a large memory buffer makes achieving high accuracy much easier. For the A-GEM method use the same number of examples in memory and $sample\_size = 10$. All models were trained using Adam optimizer with a $learning\_rate$ of 0.0005 and $batch\_size$ of 32. We chose cross entropy as a loss function and performed one training epoch for each experience. To fairly compare baseline methods with ensembles, as a backbone we use neural network with a similar number of parameters (as in ensemble). Network architectures for each experimental setup are shown in Table 3. All baseline models were trained by providing embeddings produced by feature extractor as an input.

Table 3: Architecture of neural networks used as backbones for baseline models depends on experimental setup.

Dataset	Classifiers	Network layers
MNIST	16	[512, 157, 10]
MNIST	128	[512, 1256, 10]
CIFAR-10	64	[2048, 637, 10]
CIFAR-10	128	[2048, 1274, 10]

We used E&E [28] as the main reference method. It uses an architecture similar to that of a classifier ensemble, however the nearest neighbor selection mechanism itself is not a differentiable component and the weighting strategy is different. In order to reliably compare the performance, the experimental results of the reference method were fully reproduced. Both the reference method and the proposed method used exactly the same feature extractors. Thus, we ensured that the final performance is not affected by the varying quality of the extractor, but only depends on the solutions used in the model architecture and learning method.

Ensembles. Both E&E and our DE&E were trained with the same set of hyperparameters, excluding hyperparameters in the *soft KNN* layer. The setup is shown in Table 4. Every experiment was performed in an online manner, which means one example is shown to the model only once. We use ensembles of sizes 16, 64, 128 and 1024. Based on the size, ensembles have various number of nearest neighbors assigned (Table 5). Depends on the data set, the input batch size was different. For MNIST the batch size was 60. In contrast, for CIFAR-10 we use the batch size of 10 due to larger embedding vector produced by the feature extractor.

The keys for classifiers in ensembles were randomly chosen from the standard normal distribution and normalized using the $L2$ norm. The same normalization was applied to encoded inputs during lookup for matching keys. We used Adam optimizer with a learning rate of 0.0005 to train the keys.

Table 4: Hyperparameters of the DE&E model.

Hyperparameter	Value
Learning rate	0.0001
Weight decay	0.0001
Tanh scaling	250
σ	0.0005
L	400

Table 5: Number of neighbors used for each evaluated ensemble size.

N (ensemble size)	κ (neighbors)
16	4
64	8
128	16
1024	32

Soft KNN. We use the Sinkhorn algorithm to perform the forward inference in *soft KNN*. The Sinkhorn algorithm is useful in entropy-regularized optimal transport problems thanks to its computational effort reduction. The *soft KNN* has $\mathcal{O}(n)$ complexity, making it scalable and allows us to safely apply it to more computationally expensive problems.

The values of *soft KNN* hyperparameters σ and L are also presented in Table 4. We utilize the continuous character of output vector to weight the ensemble predictions. It is worth noting that we additionally set the threshold of the minimum allowed *soft KNN* score to 0.3. It means every element in $\boldsymbol{\gamma}$ lower than 0.3 is reduced to 0. We reject such elements because they are mostly the result of non-converged optimization and do not carry important information. In this way, we additionally secure the optimization result to be as representative as possible.

A.3 Complexity and ablations.

The machine we used had 128 GB RAM, an Intel Core i9-11900 CPU, and an NVIDIA GeForce RTX 3060 GPU with 12GB VRAM. Every experiment was performed using the GPU. The comparison in training time between E&E and DE&E models is shown in Figure 5. For all evaluated data sets, the training time of our model was higher than the time to train the reference method. The results vary between data sets. In case of MNIST, the time to train fully differentiable neural architecture was about four times longer than the E&E. The difference is also noticeable in much more difficult cases (CIFAR-10 and CIFAR-100).

We observed several important aspects that affect the performance of the whole model. Firstly, since the weak learners are single-layer neural networks, the entire feature extraction process relies on a pre-trained encoder that strongly influences the upper bounds of classification accuracy. Models in the ensemble only learn the feature mapping to the class, reducing the complexity and thus reducing the forgetting phenomenon. The conducted research indicates that with the same features, the proposed method is able to obtain higher results than the reference methods. Thus, the described problem can

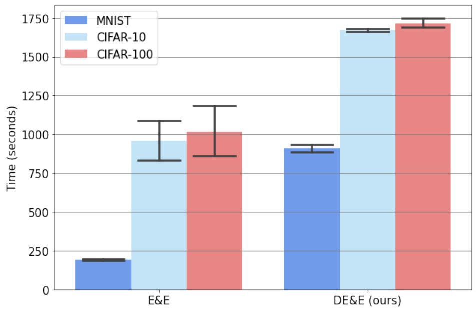
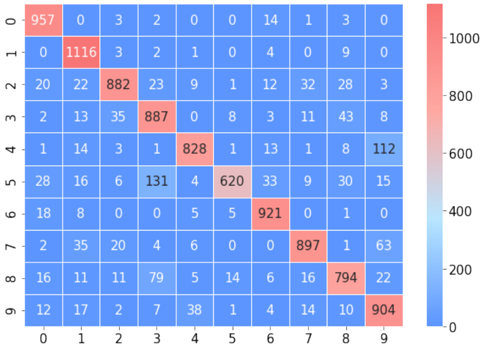


Figure 5: Training time for DE&E ensemble model (128 classifiers). Results for 10-split MNIST, 10-split CIFAR-10 and 20-split CIFAR-100 are shown.



	0	1	2	3	4	5	6	7	8	9
0	957	0	3	2	0	0	14	1	3	0
1	0	1116	3	2	1	0	4	0	9	0
2	20	22	882	23	9	1	12	32	28	3
3	2	13	35	887	0	8	3	11	43	8
4	1	14	3	1	828	1	13	1	8	112
5	28	16	6	131	4	620	33	9	30	15
6	18	8	0	0	5	5	921	0	1	0
7	2	35	20	4	6	0	0	897	1	63
8	16	11	11	79	5	14	6	16	794	22
9	12	17	2	7	38	1	4	14	10	904

Figure 6: Confusion matrix for DE&E ensemble model (128 classifiers) evaluated on the 10-split MNIST.

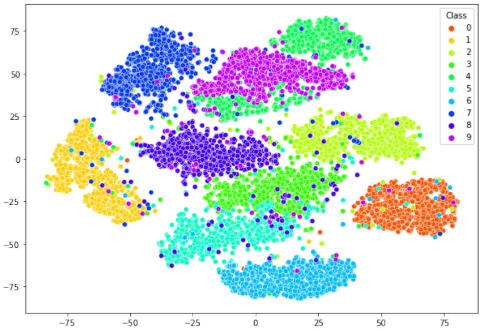


Figure 7: TSNE visualization of MNIST data set encodings (VAE encoder). 10000 examples are shown.

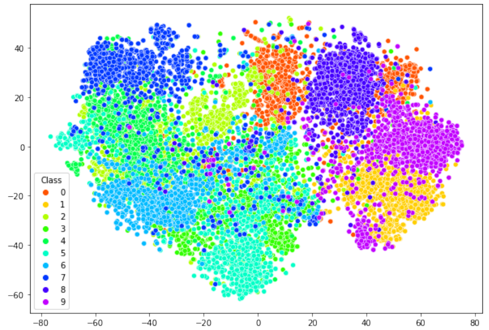


Figure 8: TSNE visualization of CIFAR-10 data set encodings (BYOL ResNet-50). 10000 examples are shown.

be partially overcome by using the architecture proposed in this paper, since the decrease in feature quality has the potential to be compensated by skillful use of other mechanisms.

An apparent drawback of the proposed neural model is the increased training time.The introduction of a differentiable *soft KNN* layer resulted in additional computational effort that clearly impacted the time complexity of the model. *soft KNN* is based on an approximation of the Entropic Optimal Transport problem solution, which is typically computationally expensive. The convergence rate of the algorithm depends on its hyperparameters that should be tuned so that the assumed error tolerance is reached as quickly as possible. Apart from optimization problems, aggregating weak learner predictions is computationally expensive, too. Unlike the reference method, where classifiers from the nearest neighborhood are trained in complete isolation, our method computes the output of all weak learners during each forward pass. However, only the learners that belong to the nearest neighbor group are selected for update (the output of the *soft KNN* layer is used for the selected classifiers to be updated). The adopted weighting procedure makes it possible to simultaneously eliminate the predictions of classifiers that are not nearest neighbors, as well as give appropriate proportions to the predictions that qualify for the group of nearest neighbors. In this case the weighting is not done for κ neighbors but for all N classifiers in the ensemble, which is very time-consuming for large N. Here, we see the field for improvement and more efficient use of the differentiable model capabilities for future work.

We observed that the cause of DE&E prediction errors is not much different than errors made by other models. The most frequently confused classes are those most visually similar to each other. Figure 6 shows the confusion matrix for DE&E (128 classifiers) prediction on the 10-split MNIST test set. Across all evaluations on the MNIST data set, the highest number of errors were made between classes 3 and 5, and also 4 and 9. In contrast, classes 6 and 7 were misclassified only a few times. This indicates a limitation posed by the feature extractor, whose output is not detailed

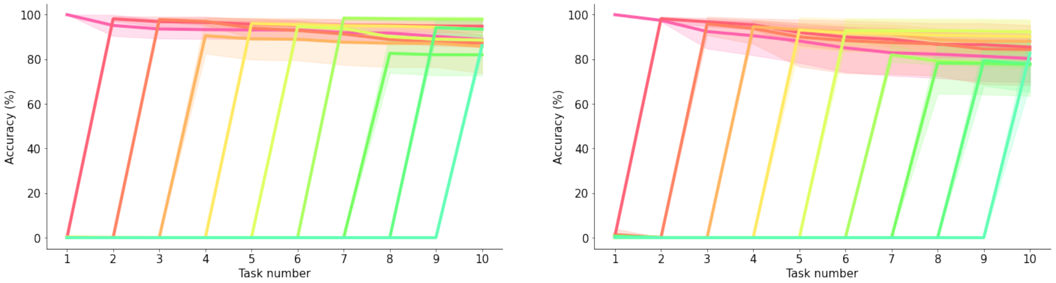


Figure 9: Accuracy (%) performance across training tasks by DE&E ensemble of 1024 (left) and 16 (right) classifiers on 10-split MNIST. The smaller ensemble accuracy decreases much more for consecutive tasks due to no possibility of classifiers specialization.

enough to sufficiently distinguish between highly similar examples. In addition, we also applied the T-SNE nonlinear dimensionality reduction algorithm into MNIST and CIFAR-10 encodings. The output is shown in Figures 7 and 8. The obtained results suggest a clear difficulty in distinguishing the mentioned classes from each other. This phenomenon is compounded on the CIFAR-10 and CIFAR-100 data sets, where the detail of examples is higher and they require much more accurate features. Thus, the use of a more accurate encoder would probably result in an increased classification accuracy performance.

A.4 Other experiments.

A.4.1 Forgetting

In Tables 6, 7, 8, and 9 we also show the forgetting rates for ensembles trained on MNIST and CIFAR-10 data sets. The forgetting measure fluctuates greatly across evaluations, especially on smaller ensembles. Additionally, the forgetting rate is influenced by class order. We observed that lower forgetting rates are particularly noticeable in the simpler MNIST cases. Importantly, forgetting is always reduced in large ensembles (1024 classifiers) using our method. As can be seen in Figure 9, the larger ensemble learns each task better than smaller one. It also has a stable performance on the already seen tasks. The smaller ensemble acquires less knowledge on subsequent tasks and forgets much more.

Table 6: Forgetting (%) and standard deviation (5 runs) for various ensemble sizes evaluated on 5-split MNIST.

	16	128	1024
E&E	9.07 ± 2.85	$\mathbf{5.18 \pm 1.98}$	5.61 ± 0.66
DE&E (ours)	$\mathbf{6.96 \pm 1.95}$	5.20 ± 1.51	$\mathbf{4.90 \pm 1.54}$

Table 7: Forgetting (%) and standard deviation (5 runs) for various ensemble sizes evaluated on 10-split MNIST.

	16	128	1024
E&E	8.13 ± 2.57	8.16 ± 1.90	5.26 ± 0.52
DE&E (ours)	$\mathbf{6.71 \pm 2.18}$	$\mathbf{6.04 \pm 1.81}$	$\mathbf{1.50 \pm 2.32}$

A.4.2 Trainable keys

Table 10 shows the effect of trainable keys on ensemble accuracy. For keys optimization, we used the Adam optimizer and a learning rate of 0.0005. As for previous experiments, a positive effect on accuracy is seen for a small ensemble of 16 classifiers. We observe this in both the 5-split and 10-split setups on the MNIST data set. In contrast, for an ensemble of 128 classifiers, key optimization leads to a decrease in prediction quality. We conclude that with a fewer number of keys it is much more

Table 8: Forgetting (%) and standard deviation (5 runs) for various ensemble sizes evaluated on 5-split CIFAR-10.

	16	128	1024
E&E	$\mathbf{17.28 \pm 3.79}$	15.38 ± 4.38	11.88 ± 1.40
DE&E (ours)	18.40 ± 3.73	$\mathbf{12.63 \pm 1.89}$	$\mathbf{11.78 \pm 3.51}$

Table 9: Forgetting (%) and standard deviation (5 runs) for various ensemble sizes evaluated on 10-split CIFAR-10. Result for the E&E with 128 classifiers is not reported due to technical reasons during evaluation.

	16	128	1024
E&E	$\mathbf{9.89 \pm 11.49}$	-	13.84 ± 2.64
DE&E (ours)	21.58 ± 4.32	$\mathbf{15.77 \pm 3.17}$	$\mathbf{13.82 \pm 2.44}$

Table 10: Impact of trainable keys on model accuracy (%). Experiments are performed on the MNIST data set. Means and standard deviations for 5 runs are shown.

	5-split		10-split	
	$N = 16$	$N = 128$	$N = 16$	$N = 128$
DE&E	85.20 ± 0.46	$\mathbf{87.80 \pm 0.47}$	84.19 ± 1.00	$\mathbf{87.54 \pm 0.24}$
DE&E + trainable keys	$\mathbf{85.30 \pm 0.51}$	87.47 ± 0.22	$\mathbf{84.64 \pm 0.69}$	87.23 ± 0.12

difficult to allow them to specialize. Giving the keys a degree of freedom may lead to an adjustment of their arrangement. However, when the number of keys is higher and the space is covered more densely, correcting them is ineffective.

A.4.3 Large ensembles and experiments on CIFAR-100

As shown in Table 11, in case of the CIFAR-100 and the largest ensemble, no significant improvement was noted for 20-split, but for 100-split the improvement was about 9%. But for smaller ensembles we do not observe significant gains over the E&E (Table 12). This clearly shows that ensembles too small in size (according to problem difficulty) does not give a chance for improvement with the proposed techniques. When there are too few models, the collective intelligence is suppressed by voting noise. However, it is important to note that increasing ensemble size does not always result in improved performance. As we have shown before, even an ensemble of 16 classifiers can perform much more accurate on the MNIST data set just by making proper use of the specialization mechanism. Small ensembles benefit most from the proposed voting method, because their each vote has a relatively greater impact on the decision than for larger models. Thus, a streamlined voting procedure results in proportionally more meaningful effects.

In case of the largest ensemble (1024 classifiers) we observed slight improvement over the E&E method in all setups evaluated (Table 13). Our method achieves higher accuracy in both 5-split and 10-split scenarios for MNIST and CIFAR-10 datasets.

Table 11: Accuracy (%) and standard deviation for models evaluated on CIFAR-100 data set. Both E&E and DE&E ensembles consist of 1024 classifiers.

	20-split	100-split
E&E	$\mathbf{40.34 \pm 0.58}$	31.60 ± 8.12
DE&E (ours)	39.72 ± 8.25	$\mathbf{40.57 \pm 6.00}$

Table 12: Accuracy (%) and standard deviation (5 runs) for smaller ensembles on CIFAR-100.

	$N = 16$	$N = 128$
E&E	7.04 ± 1.56	30.52 ± 0.74
DE&E (ours)	$\mathbf{7.19 \pm 0.54}$	$\mathbf{30.58 \pm 1.06}$

Table 13: Accuracy (%) and standard deviation (5 runs) for models evaluated on MNIST and CIFAR-10 data sets. Both E&E and End-to-End ensembles consist of 1024 classifiers.

	MNIST		CIFAR-10	
	5-split	10-split	5-split	10-split
E&E	89.13 ± 0.23	89.53 ± 0.31	66.27 ± 0.77	67.41 ± 0.80
DE&E (ours)	$\mathbf{89.45 \pm 0.18}$	$\mathbf{90.00 \pm 0.21}$	$\mathbf{67.44 \pm 0.67}$	$\mathbf{67.57 \pm 1.06}$

A.5 Related work

Continual learning methods. Currently, methods with a memory buffer such as GEM [19], A-GEM [4] or MIR [1] usually achieve the highest performance on benchmark tasks using traditional data sets [20]. Because past samples are stored in memory and repeated multiple times during training, forgetting is reduced by constantly refreshing the knowledge acquired in the past. It has been shown that even a very small number of stored examples can significantly reduce network performance degradation [5]. In addition to methods with a memory buffer, a very wide group of approaches based on parameter regularization exists. The most popular ones include EWC [12] or LWF [16]. When receiving a new dose of knowledge, these methods attempt to influence the model parameter updating procedure to be minimally invasive. The lack of a memory buffer results in a smaller memory overhead, but the price paid for this means lower efficiency. Another group of methods are approaches based on the expansion of network parameters like PackNet [21] or Progressive Neural Networks [26]. However, the main limitation of those methods is the significant increase in the number of parameters during training.

Ensemble methods. Ensemble methods are widespread in the field of deep learning [35, 2, 15]. Ensemble techniques have also been used successfully in the field of continual learning, as evidenced by the presence of methods such as BatchEnsemble [32] or CN-DPM [14]. Other contributions present in literature [8] tend to focus strongly on improving model performance rather than increasing model efficiency. Furthermore, ensemble approaches can also be used indirectly through dropout [29] or weights aggregation [33].

KNN. The K Nearest Neighbors algorithm is currently the most frequently used among a variety of machine learning techniques [30]. Despite its very high computational cost for a large number of examples, it is still often used as a baseline in various classification and regression problems. However, KNN is characterized by a lack of differentiability, effectively blocking the exploitation of its advantages in gradient training based neural architectures. This problem has attracted the attention of researchers in recent years, leading to the development of several methods that allow for approximating the output of a conventional KNN guaranteeing the ability to calculate the needed derivatives. One of the first such approaches was the use of Differentiable Boundary Trees [37], where the authors proposed a custom cost function associated with a tree's prediction. Another important work is the introduction of continuous deterministic relaxation of KNN [24], which can be used directly as a neural network layer (N^3 block). This method has shown effectiveness in both classification and image denoising problems. The most recent approach [34] (used by us) that solves the Entropic Optimal Transport problem. Importantly, the introduced form of optimization also allowed to improve the classification results, this time on the CIFAR-10 data set. However, as for each method mentioned above, the computational cost behind accuracy gains is quite significant.