Evolving Deep Neural Networks for Continuous Learning

Bruna Atamanczuk[1], Kurt Arve Skipenes Karadas[2], Bikash Agrawal[3], and Antorweep Chakravorty[2]

[1] Aker BP, Norway
[2] University of Stavanger, Norway
[3] Simplifai AS, Norway

Abstract. Continuous learning plays a crucial role in advancing the field of machine learning by addressing the challenges posed by evolving data and complex learning tasks. This paper presents a novel approach to address the challenges of continuous learning. Inspired by evolutionary strategies, the approach introduces perturbations to the weights and biases of a neural network while leveraging backpropagation. The method demonstrates stable or improved accuracy for the 12 scenarios investigated without catastrophic forgetting. The experiments were conducted on three benchmark datasets, MNIST, Fashion-MNIST and CIFAR-10. Furthermore, different CNN models were used to evaluate the approach. The data was split considering stratified and non-stratified sampling and with and without a missing class. The approach adapts to the new class without compromising performance and offers scalability in real-world scenarios. Overall, it shows promise in maintaining accuracy and adapting to changing data conditions while retaining knowledge from previous tasks.

Keywords: Deep Learning · Artificial Intelligence · Continuous Learning · Evolutionary Algorithms · Evolutionary Strategy

1 Introduction

Continuous Learning (CL) is a sub-field of Machine Learning (ML) that focuses on refining and enhancing models throughout their entire life cycle, enabling them to learn and improve even after deployment. Unlike traditional machine learning approaches, CL recognizes the need for ongoing model adaptation to new data and changing environments. This ensures that models remain relevant and effective, providing businesses with a competitive advantage in dynamic technological landscapes. CL addresses the challenge of adapting models to new data without forgetting previously learned knowledge [15], making it crucial for real-world applications like image classification, semantic segmentation, and object detection [13]. By allowing models to learn new classes incrementally and retain previous knowledge, CL offers a more efficient solution compared to retraining models from scratch.

In this paper, the term "continuous learning" will be utilized, although it is worth noting that the literature may also refer to this concept as "lifelong learning" or "incremental learning". The main idea behind CL is to train ML models sequentially, where new data is added over time allowing the model to learn new tasks. As more data is added, it is expected that the model will be able to incrementally learn and consolidate knowledge without losing accuracy.

When developing continuous learning models, it is crucial to differentiate between various scenarios that may arise. In the context of classification tasks, the following situations can occur when new data is added to the models [14, 15]: 1) In the *instance incremental scenario*, the number of classes remains constant, but new instances become available at each learning stage. 2) The *class incremental scenario* involves the addition of new classes, resulting in changes to the statistical properties of the features. 3) The *instance and class incremental scenario* occurs when both new instances and classes become available simultaneously, introducing new training patterns for both known and unknown classes.

The primary challenge in continuous learning revolves around mitigating catastrophic forgetting, where previously learned information is lost when adapting to new tasks. Various methods and strategies are employed to address this issue in classification tasks. Architectural strategies modify the model's architecture to allow for the learning of new tasks without forgetting the knowledge acquired from previous tasks [15, 17, 20]. Regularization strategies aim to optimize the parameter weights for the tasks without altering the model's architecture [5, 8, 12, 15]. The rehearsal strategy [19] involves retaining a subset of previous data, allowing the model to review old knowledge while learning new information. A more robust approach is the pseudo-rehearsal method [19], which avoids accessing the original training data. Instead, a generator is constructed to learn the distribution of input data and generate batches of pseudo data that closely resemble the distribution of the old data when the model learns new knowledge.

This work focuses on Evolutionary Strategies (ES) and their usage to improve the quality of solutions through adaptive modifications of artificial neural network weights within a continuous learning setup. ES is an approach within the field of Evolutionary Algorithms (EAs) that introduces random perturbations to the current solutions, in order to explore the search space and potentially discover better solutions.

The subsequent section provides a summary of existing research on continuous learning and outlines the associated challenges. The related work section offers an overview of evolutionary strategies, optimization techniques applied to neural networks, which will serve as the foundation for a new approach to continuous learning tasks. The methodology section introduces a novel approach for addressing continuous learning problems and details the experimental setup and evaluation metrics employed. The results section presents the performance and outcomes achieved through the implementation of the proposed evolutionary method on various datasets, with a comprehensive analysis of each dataset's results. In the conclusion, the paper summarizes the main findings and contribu-

tions to the field of class continuous learning, along with suggestions for future research endeavors.

2 Related Work

Evolutionary algorithms have been developed as a family of optimization algorithms that take inspiration from the biological evolution process. This process occurs constantly in nature and increases the diversity in the population of every living species [2].

The basic procedure followed to implement EAs is to create a population of candidate solutions to a problem and use the main principles of evolutionary methods, such as mutation, reproduction and natural selection to evolve this population over a number of generations [2, 4, 7]. In general, the primary focus of EAs lies in adapting these concepts to suit the specific characteristics of the problem addressed. Mathematically, the problem is modelled as the population and then a cost function is applied to act as the environment. The main goal is to find the solutions with the highest fitness, which is determined by the cost function [2].

In general, EAs encompass various approaches such as *Evolutionary Strategy* (ES), *Evolutionary Programming* (EP), and *Genetic Algorithm* (GA). These algorithms share a common objective of leveraging biological evolution mechanisms to enhance computational problem-solving capabilities [4]. In the case of ES, the algorithm aims to discover better solutions to a problem by introducing random perturbations to the current solution. This process, known as mutation, is a key characteristic of ES [6]. Through mutation, offspring solutions are generated and evaluated based on their fitness. The fittest individuals are selected to form the basis for the next generation, while less fit individuals are either discarded or have a lower chance of contributing to the next generation.

The first algorithms for ES were presented in the early 1960s by the researchers Rechenberg and Schwefel [1, 6], using a Gaussian distribution with zero mean and σ standard deviation, $\mathcal{N}(0, \sigma^2)$ to create the mutation of the offspring solutions [6]. In the recent versions of this approach, however, ES is portrayed as a black-box stochastic optimization technique [23], with less emphasis on its connection to biological evolution [21]. Intuitively, the optimization process can be described as an operation of "guess and check", where the general idea is that random parameters are initially chosen and then subsequently adjusted through two steps: 1) random tweaks are made to the guess, and 2) slight adjustments are then made towards more successful tweaks [21].

The optimization process, in this case, can be seen as a form of *Reinforcement Learning* (RL) [21] and have been successfully applied to two RL benchmark tasks: the Atari game-playing and the Multi-Joint dynamics with Contact (MuJoCo) control tasks. On a higher level, the first task consists of designing a model to successfully achieve superhuman results in classic Atari games, whereas the second task refers to a physics engine that is widely used for developing continuous control agents. For both cases, the main idea is the same: to train a

function to describe the behaviour of an agent that interacts with some given environment [21]. This function is usually a neural network, that rewards the agent if it takes any of the allowed actions, for example, wins a game.

3 Designing evolutionary neural networks

The conventional training process of a neural network relies on the use of the Back Propagation (BP) algorithm, which employs gradient descent to find the closest optimal solution starting from a random point. However, due to the use of gradient, the BP algorithm presents some drawbacks such as the potential for getting stagnated in local minima and slow convergence [16]. In general, EAs have been employed to find optimal values for neural networks as an alternative to the BP algorithm. Evolutionary strategies have shown promising results in various reinforcement learning tasks [22], but their application in continuous learning optimization remains unexplored, to the best of the authors' knowledge.

4 Proposed approach

The proposed approach centers around a novel method to enhance the performance of neural networks under evolving data conditions. The core idea involves incorporating perturbations into the weights and biases of the network. Initially, a model M_0 is trained using the available training data. As more data becomes available at different time periods, the model is duplicated, and random noise is introduced to the weights and biases, to create mutated copies, $\{M_{0_1}, \ldots, M_{0_N}\}$ where N is the number of desired mutations. Each newly created model is then trained using the additional data, and its accuracy is assessed. This process is repeated for a predetermined number of iterations. The accuracy of each mutated model serves as input for calculating a weighted average $\bar{W}$ of the weights and biases. Thereafter, this information is utilized to generate a new model, M_{ES}, which is subsequently employed to make predictions on unseen data. The proposed procedure is outlined in Algorithm 1.

Algorithm 1 Proposed approach

for each mutation M_{0_N} **do**
  Clone model M_0
  Draw z_i from the Uniform[0, 1] distribution
  $W_i = W_i * z_i$ ▷ Mutating weights
  $b_i = b_i * z_i$ ▷ Mutating biases
  Train model using the new training data
  Evaluate the model
  Save accuracy
end for
Calculate the weighted average $\bar{W}$ of weights and biases using Equation 1
Compile and evaluate the offspring model, M_{ES}, utilizing test dataset

As described in Algorithm 1, the weighted average of the weights and biases is obtained by Equation 1.

$$\bar{W} = \frac{\sum_{i=1}^{N} W_i \cdot acc_i}{\sum_{i=1}^{N} acc_i} \tag{1}$$

Where $\bar{W}$ represents the average weights and biases matrices. Additionally, W_i and acc_i denote the weights and biases, as well as the accuracy of the mutated models $\{M_{0_1}, \ldots, M_{0_N}\}$.

5 Experimental setup

To perform the experiments included in this paper, Python 3.10 was used. The libraries employed were standard data science libraries such as Keras, Matplotlib, NumPy, scikit-learn and TensorFlow. The code was run in an NVIDIA Tesla T4 GPU from Google Colab. Three different datasets were used for training and evaluating the evolutionary strategy:

- **MNIST** [11] dataset consists of handwritten digits having 10 classes, from 0 to 9. It contains 60,000 training images and 10,000 test images. Each image is a grayscale image in 28 x 28 pixels.
- **Fashion-MNIST** [24] dataset is originating from Zalando. It consists of 60,000 images in the training set and 10,000 images in the test set. Each image is a grayscale image in 28 x 28 pixels. There are 10 different classes.
- **CIFAR-10** [10] dataset consists of images of various transportation means and animals. It comprises 60,000 color images with shape 32 x 32 pixels in 10 classes. There are 50,000 images for training and 10,000 images for the test set.

The datasets underwent minimal preprocessing due to the absence of missing values or outliers. Input images were normalized by dividing them by 255, and categorical labels were converted to binary vectors using one-hot encoding. The training data was divided into subsets using an 80-20 split, simulating scenarios where data becomes available at different time intervals. In some cases, one class was intentionally excluded from the larger training subset and allocated to a smaller subset. The data splitting was performed with and without stratification. Overall, four scenarios were analyzed for each dataset: "All classes, stratified", "All classes", "Missing class, stratified", and "Missing class". Table 1 summarizes the dimensions of the training data for each scenario, providing an overview of the variations introduced by the preprocessing and partitioning strategies.

Furthermore, for each dataset a CNN model was developed. For the MNIST and Fashion-MNIST datasets, the same model architecture was used. It consisted of a CNN model with a 2D Convolutional layer with 32 filters, a kernel size of 3x3, and an input shape of (28, 28, 1). It was followed by a MaxPooling operation with a pool size of (2, 2). The second layer was another 2D Convolutional layer with 64 filters, a kernel size of 3x3, and ReLU activation. It was again followed

Table 1. Dimensions of subsets after splitting

Dataset	Model	Large subset	Small subset
MNIST	All classes, with and without stratification	48,000	12,000
	Missing class, with and without stratification	43,266	16,734
Fashion-MNIST	All classes, with and without stratification	48,000	12,000
	Missing class, stratified	43,200	16,800
	Missing class	43,145	16,855
CIFAR-10	All classes, with and without stratification	40,000	10,000
	Missing class, stratified	36,000	14,000
	Missing class	35,996	14,004

by a MaxPooling operation with a pool size of (2, 2). The model then includes a fully connected layer with 128 units and ReLU activation. Finally, an output layer with softmax activation consisting of 10 units was added to predict class probabilities.

The architecture used for the CIFAR-10 dataset consisted of a 2D Convolutional layer with 32 filters, a kernel size of 3x3, an input shape of (32, 32, 3), and ReLU activation. It was followed by another 2D Convolutional layer with 32 filters, a kernel size of 3x3, and ReLU activation. A MaxPooling operation was then applied. A Dropout layer with a rate of 0.25 is added to reduce overfitting. The second layer consisted of a 2D Convolutional layer with 64 filters, a kernel size of 3x3, and ReLU activation. This was followed by another 2D Convolutional layer with 64 filters, a kernel size of 3x3, and ReLU activation. Another MaxPooling operation was applied, followed by a Dropout layer with a rate of 0.25 to further prevent overfitting. Next, a Flatten layer was added to convert the 2D feature maps into a 1D vector. This was followed by a fully connected layer with 512 units and ReLU activation. Again, a Dropout layer with a rate of 0.5 was introduced. Finally, an output layer consisting of 10 units and softmax activation was added.

For each of the neural network architectures described, a model was initialized with random weights and biases. Thereafter, the model M_0 was created by applying TensorFlow's `clone_model` function. The model M_0 was then trained using the larger subset of training data. Then, the trained model was used to create mutated copies, $\{M_{0_1}, \ldots, M_{0_N}\}$. This was achieved by introducing random noise, drawn from the Uniform[0, 1] distribution, to the model's weights and biases. Afterwards, these mutations were trained using the smaller subset of training data. Here, a validation split of 10% was used. The validation accuracy was monitored during the training process and further used to calculate the weighted average of the weights and biases for the mutated models. Also, an `EarlyStopping` callback was defined with patience of 3.

Finally, a new model was created from the weighted averages and evaluated on the test data. The final model is later denoted as ES model (M_{ES}) or offspring

model. To assess the performance and effectiveness of the continuous training approach using evolutionary strategy, clones of M_0 were mutated 3, 5, 10, 15 and 50 times, respectively.

Prior to conducting the experiments detailed above, baseline models with identical architectures to the ES models were trained on the entire training data for the purpose of comparison. A replica of the model initialized with random weights and biases, referred to as $M_{baseline}$, was trained using the complete training set.

6 Evaluation metrics

In order to assess the performance and effectiveness of the models developed in this work, the metrics described below were employed. It is important for the reader to understand that the objective was not solely focused on achieving the highest performance for the datasets under consideration. Instead, the primary aim was to investigate the application of evolutionary strategy in the context of continuous learning.

6.1 Confusion Matrix

The confusion matrix provides a clear and concise overview of how well a classifier's predictions match the actual outcomes. Typically, the predicted classes are organized horizontally, while the actual labels are arranged vertically, although this order can be reversed [9]. The elements on the diagonal of the matrix offer insight into the classifier's performance by showcasing accurate predictions, while the off-diagonal elements indicate the instances where the classifier made incorrect predictions.

The confusion matrix is often denoted by $\mathbf{A}(i, j)$, which can be used to calculate several evaluation metrics. However, in this report, accuracy has been selected as the metric of choice due to its simplicity. The authors suggest that readers take into account additional evaluation metrics based on their specific use case. While the confusion matrix provides valuable insights into a classifier's performance, it may not capture all aspects of interest.

6.2 Accuracy

Accuracy serves as a metric to assess a classifier's ability to accurately classify samples. It is determined by calculating the ratio of correctly classified samples to the total number of samples [9]. Mathematically, accuracy can be calculated as:

$$Accuracy = \frac{\sum_{i=1}^{M} A(i,i)}{\sum_{i=1}^{M} \sum_{j=1}^{M} A(i,j)} \tag{2}$$

where $A(i, i)$ represent the diagonal elements and the $A(i, j)$ refer to the off-diagonal elements of the confusion matrix.

7 Results

The results are presented in Table 2 and visually represented in Fig. 1 to enable an overall performance comparison. In the missing class cases, class number 6 was excluded from the larger subsets of the training data, and added back to the smaller portion of training data. All models were evaluated using the test data.

It can be noticed that the accuracy remains rather stable regardless of the number of mutations used. It is also worth noticing that accuracy of the baseline model and the model M_0 were higher than any of the ES models considering all classes. However, when there is a class missing, the ES models learn the new patterns, achieving a high accuracy than the M_0 model. On the other hand, the baseline model still achieved higher performance. Moreover, it is essential to acknowledge that developing a model trained on the entire dataset, may not be feasible or scalable in real-world scenarios. The comparison to the M_0 model is more relevant in this context.

Table 2. Summary of performance of the models

				ES models				
Dataset	**Model**	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
MNIST	All classes (stratified)	0.9903	0.9885	0.9886	0.9888	0.9895	0.9896	**0.9899**
	All classes	0.9907	0.9890	**0.9899**	0.9893	0.9893	0.9896	0.9892
	Missing class (stratified)	0.9895	0.8959	0.9869	0.9881	**0.9886**	0.9881	0.9878
	Missing class	0.9895	0.8960	0.9877	0.9877	**0.9878**	0.9877	0.9878
Fashion-MNIST	All classes (stratified)	0.9081	0.9121	0.9061	**0.9073**	0.9064	0.9051	0.9072
	All classes	0.9066	0.9000	0.9022	0.8945	**0.9046**	0.9026	0.9044
	Missing class (stratified)	0.9022	0.8579	0.8812	0.8847	0.8900	**0.8923**	0.8903
	Missing class	0.9043	0.8546	0.8921	**0.8952**	0.8951	0.8914	0.8940
CIFAR-10	All classes (stratified)	0.7846	0.7669	0.7618	0.7613	**0.7691**	0.7611	0.7649
	All classes	0.7886	0.7718	0.7612	0.7666	0.7692	0.7662	**0.7733**
	Missing class (stratified)	0.7843	0.6902	0.7098	0.7247	0.7271	**0.7381**	0.7227
	Missing class	0.7902	0.6896	0.7317	0.7401	0.7341	**0.7503**	0.7337

The results reveal several important findings as presented in Fig. 1. Firstly, the accuracy of the models remains consistent when utilizing evolutionary strategies for continuous learning, demonstrating their ability to maintain performance throughout the learning process. Secondly, the evolutionary strategy models exhibit enhanced performance when dealing with datasets containing missing

classes compared to the initial model, showcasing the algorithm's capacity to learn and adapt to new knowledge. Additionally, the baseline model achieves higher accuracy, as expected, especially when there is a missing class, due to its utilization of all available data and consistent class distributions. However, it is worth noting that such models may encounter feasibility and scalability challenges in practical scenarios.

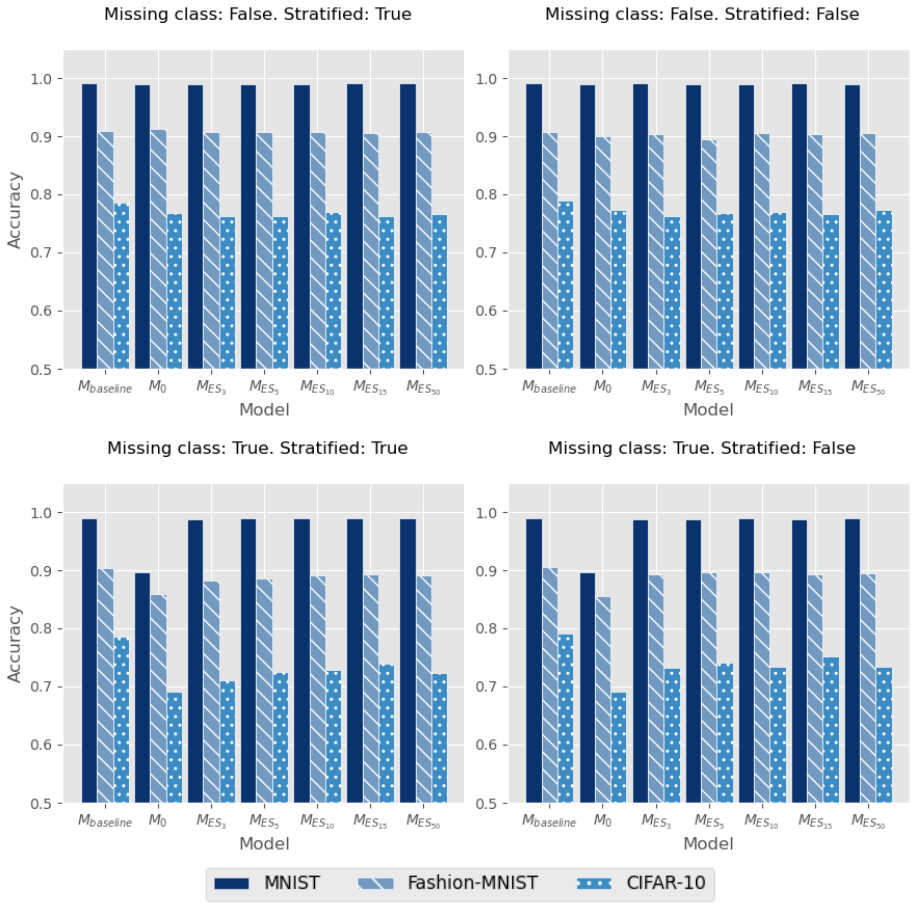

Fig. 1. Summary of accuracy for the models considering the different splits and datasets

Another interesting result emerges when looking at the confusion matrices obtained for the models. In this paper, for the sake of simplicity, the results are shown only for the Fashion-MNIST dataset and omitted for the remaining cases. However, a similar rationale can be extrapolated to all cases where there was a missing class.

Figure 2 presents the confusion matrices for the baseline model and the top-performing ES model for the Fashion-MNIST dataset, considering a missing class and stratified split. Here, there is a notable difference as class 6 is not present in the model M_0, which was used to develop the M_{ES} model. Moreover, this particular class proves to be challenging to predict, even for the baseline model trained with all available data. The ES model demonstrates enhanced performance specifically in classes 3 and 4, with an observed increase of 8.8% and 8.0%, respectively in the correctly classified samples (true positives). Additionally, there is an increase in false positives for classes 3 and 4, compared to the baseline model. However, the number of false negatives decreased for the same classes. Nevertheless, the ES models are capable of achieving reasonable accuracy, even when encountering a new class and facing a slight class imbalance resulting from the inclusion of new data in the training dataset.

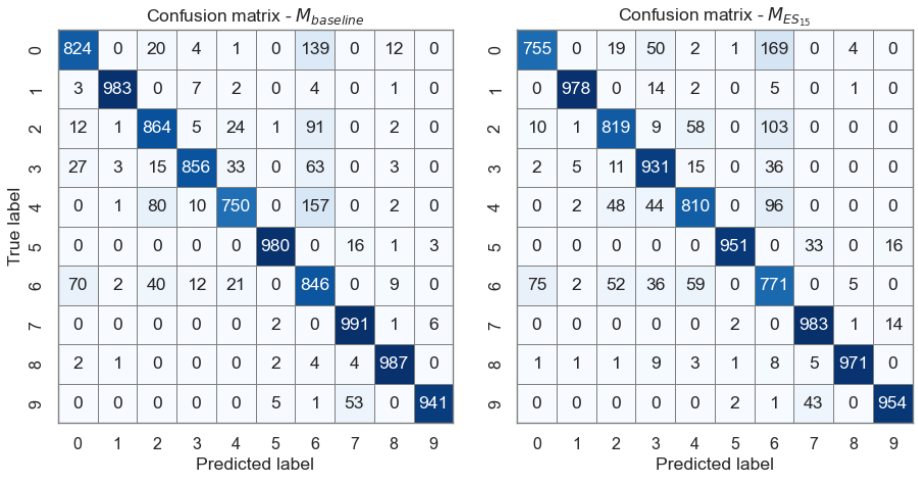

Fig. 2. Summary of accuracy for the models considering the different splits and datasets

Focusing on the different scenarios analyzed, it is worth examining the variations between the best-performing ES model and the baseline model. In the context of *All classes* scenarios, the maximum difference between the baseline model and the best-performing ES model was 1.6%, as observed specifically in the CIFAR-10 dataset. On the other hand, when considering the *Missing class* scenarios, the maximum difference between the baseline model and the best-performing ES model was 4.3%, also observed in the CIFAR-10 dataset.

However, it is important to recognize that creating a model trained on the entire dataset might not be practical or scalable in real-world situations. Thus, a more relevant comparison would be between the M_0 model and the most successful ES model, M_{ES}. Notably, for the *All classes* scenarios, the maximum difference between the best-performing ES model and the M_0 model was

a mere 0.5%. Moreover, the outcomes became more significant when observing the *Missing class* scenarios. Across the datasets, the disparity between M_0 and the best-performing ES model varied. In the case of the MNIST dataset, both the stratified and non-stratified splits displayed a remarkable margin of approximately 9.2% in favor of the best-performing M_{ES} model. For the Fashion-MNIST dataset, the stratified case exhibited a difference of 3.4%, while the non-stratified case showed a difference of 4.1%. In comparison, the CIFAR-10 dataset, being more complex, demonstrated even larger variations of 4.8% and 6.1% for the stratified and non-stratified splits, respectively.

The aim of this paper was to demonstrate the effective application of evolutionary strategies in situations where training models on the complete dataset are impractical due to evolving data distributions or the availability of additional data. Usually, training models on entire datasets require significant time and computational resources. Moreover, accumulated data over the years can become massive. Thus, posing challenges in terms of cost and time needed for training the entire dataset. In summary, these findings highlight the efficacy and feasibility of employing evolutionary strategies for continuous learning. The presented approach facilitates faster and more efficient progress in production, as they ensure accuracy, adaptability to changing data, and enhanced predictions for specific classes.

8 Conclusion

This paper introduces an innovative solution to tackle the challenges of continuous learning, a rapidly advancing field in machine learning. Unlike traditional continuous learning methods that typically require retraining the entire model or making architectural changes to prevent obsolescence over time, this work proposes a new approach inspired by evolutionary strategy. The effectiveness of the suggested method was evaluated on three distinct datasets, and the results showed consistent or even improved accuracy over time, without experiencing catastrophic forgetting. Although the continuous learning models slightly trailed the baseline model in terms of overall performance, they exhibited promising outcomes, demonstrating their capability and potential in handling evolving data scenarios.

To summarize, the presented evolutionary strategy-based approach shows great potential for real-world applications under changing data conditions. By introducing perturbations and leveraging the average weights and biases of mutated models, this method showcases the ability to maintain accuracy and adapt to new data while retaining knowledge from previous tasks.

Future work

In order to advance the current research, further investigations should be conducted regarding the cloning and mutation process of the model, M_0, along with the calculation of weighted averages based on accuracy. One potential approach

involves establishing a threshold for accuracy, whereby underperforming models are discarded during the weighted average calculation. Moreover, it is worth considering the utilization of a different metric, other than accuracy, for evaluating the ES model, particularly when addressing problems involving class imbalance [18]. Numerous metrics, such as the ones proposed by [3] could be considered in this regard.

Subsequent research efforts should focus on the application of the proposed approach to real-life datasets, as they typically offer a larger and more diverse range of data. This will allow the creation of multiple ES models at different time steps. Furthermore, it is suggested to test the approach using alternative data splits, rather than adhering to the conventional 80-20 rule employed in this study. For instance, a split ratio of 70-20-10 could be adopted to allow for the examination of multiple time steps. Additionally, the potential impact of training data size on the effectiveness of continuous training should be taken into account.

It is also worth exploring hyperparameter optimization techniques, as they are crucial to enhance the performance of the models. The models can be fine-tuned by systematically optimizing the hyperparameters to achieve better results.

Finally, it is recommended to explore the application of the proposed approach in regression problems. While the current study focuses on classification tasks, investigating its effectiveness in regression scenarios would provide valuable insights into its potential applicability across various domains.

References

1. Beyer, H.G., Schwefel, H.P.: Evolution strategies–a comprehensive introduction. Natural computing **1**, 3–52 (2002)
2. Câmara, D.: 1 - evolution and evolutionary algorithms. In: Câmara, D. (ed.) Bio-inspired Networking, pp. 1–30. Elsevier (2015). https://doi.org/https://doi.org/10.1016/B978-1-78548-021-8.50001-6, https://www.sciencedirect.com/science/article/pii/B9781785480218500016
3. Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., Maltoni, D.: Don't forget, there is more than forgetting: new metrics for continual learning. arXiv preprint arXiv:1810.13166 (2018)
4. Ding, S., Li, H., Su, C., Yu, J., Jin, F.: Evolutionary artificial neural networks: a review. Artificial Intelligence Review **39**(3) (2013)
5. Dutt, A.: Continual learning for image classification. Ph.D. thesis, Université Grenoble Alpes (ComUE) (2019)
6. Eiben, A.E., Smith, J.E.: Evolution Strategies, pp. 71–87. Springer Berlin Heidelberg, Berlin, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05094-1_4
7. Fogel, D.B.: An introduction to simulated evolutionary optimization. IEEE transactions on neural networks **5**(1), 3–14 (1994)
8. Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of

Sciences **114**(13), 3521–3526 (2017). https://doi.org/10.1073/pnas.1611835114, https://dx.doi.org/10.1073/pnas.1611835114
9. Kotu, V., Deshpande, B.: Chapter 8 - model evaluation. In: Kotu, V., Deshpande, B. (eds.) Data Science (Second Edition), pp. 263–279. Morgan Kaufmann, second edition edn. (2019). https://doi.org/https://doi.org/10.1016/B978-0-12-814761-0.00008-3, https://www.sciencedirect.com/science/article/pii/B9780128147610000083
10. Krizhevsky, A.: Learning multiple layers of features from tiny images (2009)
11. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010)
12. Li, Z., Hoiem, D.: Learning without forgetting (2017)
13. Liu, Y., Hong, X., Tao, X., Dong, S., Shi, J., Gong, Y.: Model behavior preserving for class-incremental learning. IEEE Transactions on Neural Networks and Learning Systems pp. 1–12 (2022). https://doi.org/10.1109/tnnls.2022.3144183, https://dx.doi.org/10.1109/tnnls.2022.3144183
14. Lomonaco, V., Maltoni, D.: Core50: a new dataset and benchmark for continuous object recognition. In: Conference on Robot Learning. pp. 17–26. PMLR. https://doi.org/10.48550/arxiv.1705.03550, https://dx.doi.org/10.48550/arxiv.1705.03550
15. Luo, Y., Yin, L., Bai, W., Mao, K.: An appraisal of incremental learning methods. Entropy **22**(11), 1190 (2020)
16. Mirjalili, S.: Evolutionary Feedforward Neural Networks, pp. 75–86. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-93025-1_6
17. Polikar, R., Upda, L., Upda, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on **31**, 497 – 508 (12 2001). https://doi.org/10.1109/5326.983933
18. Powers, D.M.: Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. arXiv preprint arXiv:2010.16061 (2020)
19. Robins, A.: Catastrophic forgetting, rehearsal, and pseudorehearsal. Connection Science **7**(2), 123–146 (1995)
20. Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R.: Progressive neural networks (2022)
21. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017), https://openai.com/research/evolution-strategies
22. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv pre-print server (2017). https://doi.org/None arxiv:1703.03864, https://arxiv.org/abs/1703.03864
23. Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., Schmidhuber, J.: Natural evolution strategies. The Journal of Machine Learning Research **15**(1), 949–980 (2014)
24. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. CoRR **abs/1708.07747** (2017), http://arxiv.org/abs/1708.07747