FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme / specialisation: Applied Data Science	The spring semester, 2023 Open
Authors: Bruna Atamanczuk Kurt Arve Skipenes Karadas	
Supervisor at UiS: Antorweep Chakravorty Co-supervisor: External supervisor(s): Bikash Agrawal (Simplifai.ai)	
Thesis title: Evolving Deep Neural Networks for Continuous Learning: Addressing Challenges and Adapting to Changing Data Conditions without Catastrophic Forgetting	
Credits (ECTS): 30	
Keywords: Deep Learning; Artificial Intelligence; Continuous Learning; Evolutionary Algorithms; Evolutionary Strategy	Pages: 80 + appendix: 26 Stavanger, June 15th, 2023

Bruna Atamanczuk
Kurt Arve Skipenes Karadas

Evolving Deep Neural Networks for Continuous Learning: Addressing Challenges and Adapting to Changing Data Conditions without Catastrophic Forgetting

Master's thesis in Applied Data Science
Supervisor: Antorweep Chakravorty
Co-supervisor: Bikash Agrawal
June 2023

University of Stavanger
Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

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 thesis 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 16 scenarios investigated without catastrophic forgetting. The experiments were conducted on three benchmark datasets, MNIST, Fashion-MNIST and CIFAR-10. Furthermore, different deep learning models were used to evaluate the approach, such as MLP and CNN. The data was split considering stratified and non-stratified sampling and with and without missing classes. The approach adapts to new classes 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

ACKNOWLEDGEMENTS

We are immensely grateful to our supervisors, Antorweep Chakravorty and Bikash Agrawal, for their exceptional guidance and support during our research. Their expertise and unwavering belief in our abilities have been invaluable. The biweekly meetings provided a platform for productive discussions, inspiring ideas, and constructive feedback, fostering a positive and motivating atmosphere that fueled our progress and pushed us to excel.

Antorweep and Bikash's insights in the field of data science have enriched our understanding and helped us navigate through challenges. Their availability and responsiveness have been remarkable, always addressing our queries and concerns with patience and support. We express our deepest gratitude to Antorweep and Bikash for their unwavering support, guidance, and motivation throughout our research journey. Your contributions have been instrumental in shaping the direction and success of our thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

A list of all abbreviations in alphabetical order is presented below.

AI	Artificial Intelligence
ANN	Artificial Neural Network
BP	Back Propagation
CL	Continuous Learning
CNN	Convolutional Neural Network
DNN	Deep Neural Networks
EA	Evolutionary Algorithm
EP	Evolutionary Programming
ES	Evolutionary Strategies
EWC	Elastic Weight Consolidation
FN	False Negative
FP	False Positive
GA	Genetic Algorithm
LwF	Learning without Forgetting
ML	Machine Learning
MLP	Multilayer Perceptron
MuJoCo	Multi-Joint dynamics with Contact
PNN	Progressive Neural Networks
RL	Reinforcement Learning
TN	True Negative
TP	True Positive

CHAPTER

ONE

INTRODUCTION

Continuous learning is the practice of refining and improving machine learning models throughout their entire life cycle. In a traditional Machine Learning (ML) life cycle, a model is built with training data, deployed into production, and periodically improved over time. However, continuous learning takes this process a step further by acknowledging that models can continue to learn and improve even after deployment. This is achieved through ongoing refinement, retraining, and adaptation of the model to new data and changing environments. Continuous learning ensures that machine learning models remain relevant, effective, and up-to-date, providing businesses with a competitive edge in today's fast-paced technological landscape.

1.1 Motivation

Continuous Learning (CL) is an important topic in machine learning because it addresses the challenge of continuously adapting a model to new data without forgetting previously learned knowledge [1]. This is a crucial requirement for many real-world applications, such as image classification, semantic segmentation and object detection [2].

Traditionally, the approach used to update a model once more data is available was to retrain it from scratch. This can result in models becoming computationally expensive and time-consuming [2]. CL offers a more efficient solution by allowing a model to learn new classes incrementally, without the need to discard previous knowledge. Furthermore, CL also addresses problems such as class imbalance and catastrophic forgetting. The former is related to the difficulty of learning minority classes due to a lack of labelled data [3], and the latter is related to the tendency of Deep Neural Networks (DNN) to forget previously learnt tasks once new information is incorporated

[4]. CL allows the model to learn new classes over time, and to retain the previous knowledge, which can help improve the overall performance of the model.

Another motivation for working with CL is the increasing availability of large datasets, and deep learning techniques, making it possible to develop powerful models for a variety of tasks.

In summary, class continuous learning is a valuable area of research not only because it addresses the real-world challenges of adapting models to new data, but also aims at improving performance, and reducing the computational costs associated with retraining models from scratch. In addition, it is important to continue developing and enhancing these methods to make them more efficient, practical, and useful for a wider range of applications.

1.2 Problem definition

The continuous influx of data into a given environment poses a significant challenge to ML models that must remain up-to-date and effective. Specifically, as new and existing classes frequently emerge within the data stream, the ML models can become outdated and require retraining to adapt to changing conditions. While stakeholders may choose to retrain these models from scratch, such an approach may prove both inefficient and infeasible as data volume increases over time.

To address this challenge, there is a need to explore alternative technologies that enable continuous learning and can facilitate efficient and scalable updates to ML models in the face of changing data conditions. By leveraging such approaches, it may be possible to avoid complete retraining and enable more effective management of the continuous flow of data in the environment.

1.3 Data

All data utilized in this work is sourced from TensorFlow's Python library, which is openly available.

1.4 Objectives

The objectives of this work are:

1. Discuss the challenges of continuous learning, such as catastrophic forgetting and class imbalance.
2. Provide an overview of common approaches used for class continuous learning.
3. Experiment with a new approach inspired by the evolutionary strategy to solve the described problem.
4. Test a new approach on different datasets and evaluate results.

1.5 Structure of the work

This thesis is organized into six chapters.

1. Introduction: The introduction presents the topic and its importance in real-world applications.
2. Literature review: In this chapter, the existing research on continuous learning is summarized, and the challenges related to this approach are presented.
3. Related work: This chapter presents an overview of optimization techniques applied to neural networks known as evolutionary strategies which will be further used to develop a new approach to continuous learning tasks.
4. Methodology: A novel approach for solving continuous learning problems is presented as well as the experimental setup and the metrics used to evaluate the performance of the algorithm used are described.
5. Results: The results and performance achieved by utilizing the new continuous learning approach across various datasets are presented. The outcomes obtained from each dataset are carefully examined and evaluated to assess the effectiveness of the proposed evolutionary method.
6. Conclusion: The main findings of the thesis and its contributions to the field of class continuous learning are summarized. Additionally, suggestions for future research are given.

CHAPTER

TWO

LITERATURE REVIEW

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. Traditional approaches often suffer from limitations when faced with large-scale datasets or scenarios where data distribution changes over time. In contrast, continuous learning enables Machine Learning (ML) models to learn constantly from new data instances, adapt to changing environments, and expand their knowledge progressively. This chapter presents an overview of concepts used to build continuous learning models.

2.1 Continuous learning

Over the last decades, data has become widely available. This increasing availability of data has promoted the development of powerful ML models. Traditionally these models are trained with static, identically distributed and well-labelled data [1]. However, for most real-world applications new data is constantly changing, resulting in the models becoming obsolete over time. In order to respond to the changes in data, the models must be retrained. Defining how and when to do this is far from trivial.

Continuous learning (CL) is the sub-field of ML that tackles such questions. This field is usually referred to as continuous learning, lifelong learning, or incremental learning. For the purpose of this report, we will be using the term continuous learning. We acknowledge that these terms are synonymous within the context of this thesis. CL refers to building models to learn continually from a stream of data. In this setting, data is frequently changing which might cause its distribution to diverge from the original training data, resulting in a reduction of the model's accuracy over time. To avoid this, the model must be retrained to learn new knowledge whilst remembering

the previous one. The continuous training line of research is concerned with automating the training process once new data becomes available [5]. In order to develop a successful continuous training pipeline, [5, 6] recommend answering the following questions:

- *When should a model be retrained?*
  Here one should decide on the retraining schedule based on the triggers defined to spot performance changes, data changes or even define a periodic retraining schedule.
- *How much data is needed to retrain the model?*
  The data is chosen based on a fixed window, dynamic window or selected based on re-sampling strategies.
- *What should be retrained?*
  Batch (offline) vs. incremental (online) learning.
- *When to deploy the model after retraining?*
  A/B testing.

2.2 Continuous learning process

Continuous learning aims to design models that continuously learn new knowledge without forgetting the previous one. 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. This idea is illustrated below:

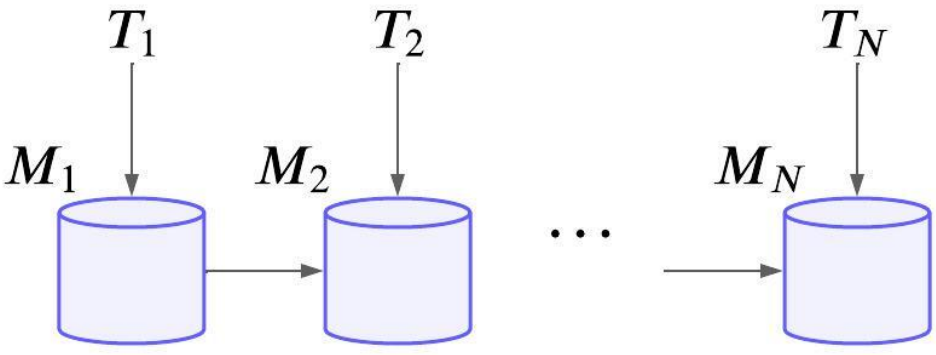


Figure 2.2.1: Continuous learning process

In Figure 2.2.1 the data is divided into tasks represented by $\{T_1, T_2, \ldots, T_N\}$, and T_i can be denoted as $T_i = \{(x_1^i, y_1^i), \ldots, (x_i^i, y_i^i)\}$. At each time interval, a new model M_i is trained using the new data T_i and the previous model M_{i-1} [1]. As more data is added, it is expected that the model will be able to incrementally learn and consolidate knowledge without losing accuracy.

2.3 Continuous learning scenarios

In addition to the conceptual model, it is important to distinguish between different scenarios that can occur when developing continuous learning models. These scenarios serve as means to contextualize the topic and provide insight into the challenges encountered in continuous learning. For classification tasks, the following can happen once new data is added to the models [1, 7]:

- **Instance incremental scenario**: in this scenario, the number of classes is fixed and new instances become available in each learning stage. This leads to changes in the relationship between features and outcomes, resulting in the appearance of a new predictive pattern [8].

- **Class incremental scenario**: in this case, new classes become available causing the statistical properties of the features to change [8]. This leads to the introduction of new training patterns as data is added.

- **Instance and class incremental scenario**: this is the scenario that better reflects the real world. In instance and class incremental scenarios, new instances and classes become available resulting in the addition of new training patterns for both known and unknown classes.

2.4 Challenges in continuous learning

2.4.1 Class imbalance

Class imbalance is one of the challenges investigated in the field of ML [9]. It occurs when the instances of one class outnumber the instances of the remaining classes. This causes the majority class to introduce bias to the learning algorithm affecting its performance. According to [10], class imbalance is one of the most crucial causes of catastrophic forgetting.

2.4.2 Catastrophic forgetting

A continuous learning system aims to imitate human learning, where knowledge is acquired gradually throughout one's life and utilized as necessary. Although the Artificial Intelligence (AI) community has investigated this concept, applying it to neural networks leads to the issue of *catastrophic forgetting*. This phenomenon refers to the model's inability to retain previously learned information as it adjusts to new data, which arises when neural networks struggle to retain old knowledge while simultaneously acquiring new information. This problem was described by [11, 12], in the late 1980s.

The occurrence of this problem is actually a special case of the stability-plasticity dilemma described in [13]. In order for an organism to adapt to new changes, its adaptive mechanisms have to be stable and plastic at the same time. A too-stable system cannot learn new information, while a too-plastic system forgets everything old and learns everything new. As a result of this, if the model is too plastic, catastrophic forgetting occurs. Despite the cause of catastrophic forgetting not being studied analytically, it is known that training neural networks on new samples alters the already established representations of internal features leading to this issue [14]. For this reason, various studies focus on reducing catastrophic forgetting in order to enable continuous learning [15].

2.5 Continuous learning strategies

The main challenge of continuous learning is to avoid the problem of *catastrophic forgetting*. There are different methods or *strategies* used to solve the problem of catastrophic forgetting in classification tasks. These are explored below, along with their advantages and drawbacks. It is worth emphasizing that some of the methods presented do not fall into a particular strategy.

2.5.1 Architectural strategy

The architectural strategy operates under the assumption that distinct learning tasks require different sets of parameters. The general idea behind this approach is that the architecture of the model is modified to allow for new tasks to be learned without forgetting the information learned from previous tasks. One approach is to train various models for each incremental task and then use a selector to determine which model to use [1]. Amongst the most known methods used in this scenario are Learn++[16] and Progressive Neural Networks (PNN) [17].

The Learn++ algorithm is a type of ensemble ML algorithm that aims to enhance prediction accuracy by combining the results of several classifiers. Its primary focus is on learning from sequential data streams. The Learn++ algorithm works by training various classifiers on distinct subsets of the input data. Subsequently, each classifier generates a prediction, which is then integrated with the predictions of the other classifiers using a weighted voting scheme. These weights are continually adjusted based on the classifier's performance on the present input data and the preceding data streams. The advantages of Learn++ include the ability to learn new classes, a small number of parameters and a short training time. The main pitfall is that it can suffer from data imbalance when learning new classes.

The PNN deals with catastrophic forgetting by gradually adding new tasks to the network without disrupting the acquired knowledge from previous tasks. This is done by fixing the parameters of the previous network once a new task is added. In practice, a new network is trained on the data of the new task and on the output from the previous network(s). This approach seems very effective against catastrophic forgetting. However, as the amount of tasks increases, so does the number of parameters, which may require a large amount of memory to train the model, particularly when dealing with complex tasks or large datasets.

2.5.2 Regularization strategy

The regularization strategies focus on finding the best set of parameter weights for the tasks without modifying the architecture of the models [15]. The main idea behind regularization involves increasing the error during the training phase of ML models in order to decrease the generalization error. This is usually achieved by adding a penalty to the weights of a network, which encourages them to remain small as to avoid the risk of overfitting.

In this setting, a special regularization term is added to the loss function, thus preventing catastrophic forgetting. The most representative methods in this category are Elastic Weight Consolidation (EWC) [4] and Learning without Forgetting (LwF) [18].

EWC uses a regularization technique to prevent the network from forgetting important information from previous tasks while still allowing it to learn new tasks. The algorithm works by assigning a penalty term to the weights of the network that have a large impact on the performance of the previous tasks. This penalty acts as a regularization term during training, effectively limiting the amount by which these weights can change during subsequent

training on new tasks. This regularization technique avoids catastrophic forgetting without the need to store old data or expand the network, resulting in reduced storage requirements compared to other methods [1].

LwF uses a different approach, known as distillation, to prevent the network from forgetting important information from previous tasks while still allowing it to learn new tasks. The distillation involves training one network on a large dataset and then using the output to generate soft targets, which are probability distributions over the classes of the dataset. These soft targets are then used to train a new network, which learns to match the soft targets of the previous network. This concept is applied in LwF, where the algorithm works by training a new network on a new task while simultaneously distilling the knowledge from the previously learned network. This process helps to prevent catastrophic forgetting by ensuring that the new network does not overwrite the knowledge from the previous network [1].

In general, the approaches that use regularization strategies are limited as a continuous learning method as they cannot increase the model's capacity. As a result, after adding a certain number of tasks, a saturation point is reached causing the average generalization performance to suffer.

2.5.3 Rehearsal and pseudo-rehearsal strategy

Another strategy that exists to eliminate or at least minimise the catastrophic forgetting problem is the rehearsal method. There exist several forms of rehearsal mechanisms, namely recency, random and sweep rehearsal. Based on a paper by Robins ([19]), *sweep rehearsal* is the most successful approach. This method allows the model to review the old knowledge whenever it learns new knowledge by retaining a subset of the previous data.

However, rehearsal in general may have a limitation in that previously learned information may not be available for retraining. A solution to this problem is the pseudo-rehearsal method. In this technique, rehearsal is provided without having to access the original training data. As a solution to the plasticity-stability dilemma, a generator is constructed to learn how input data is distributed. The generator will create batches of pseudo data that are very similar to the old data in distribution when the model learns new knowledge [19].

The rehearsal method involves retaining a subset of previous data, which the model reviews whenever it learns new knowledge. On the other hand, the pseudo-rehearsal method constructs a generator that learns the distribution

of the input data [1]. This generator produces a batch of pseudo data that is similar to the old data in distribution when the model learns new knowledge, thereby allowing the model to address the plasticity-stability dilemma. In the retraining stage, the model is supervised by both new data and pseudo data to update its knowledge.

2.5.4 A note on different strategies

It is important to emphasize that the aforementioned approaches have both advantages and disadvantages with regard to memory usage, execution time, and knowledge retention. For example, techniques that allow the model architecture to grow in order to retain knowledge can become increasingly expensive due to memory usage. On the other hand, fixed architecture techniques can be faster to run, although they can only accommodate a fixed number of tasks due to the limited amount of parameters used. Furthermore, rehearsal and pseudo-rehearsal strategies generally assume that the data distribution remains stationary over time. However, in real-world scenarios, the underlying data distribution might change or drift over time. These strategies may struggle to adapt to new data distributions, potentially leading to a decrease in the overall performance of the algorithms.

CHAPTER

THREE

RELATED WORK

3.1 Optimization of neural networks using evolutionary algorithms

This work focuses on **Evolutionary Strategies** (ES) and their usage to improve the quality of solutions through adaptive modifications of Artificial Neural Network (ANN) weights within a continuous learning setup. ES is an approach within the field of evolutionary algorithms that introduces random perturbations to the current solutions, in order to explore the search space and potentially discover better solutions. To provide a comprehensive understanding of the process undertaken in this thesis, this chapter presents an overview of key concepts associated with ES.

3.1.1 Evolutionary algorithms

Evolutionary algorithms (EAs) have been developed as a family of optimization algorithms that take inspiration from the biological evolution process. The core idea of EAs is to create a population of individuals exposed to environmental pressures, leading to evolution [20]. This is commonly referred to as *survival of the fittest* [21]. Through this mechanism, surviving individuals adapt to the environment by acquiring favourable traits, that are measured in terms of *fitness*. The fitness of an individual measures the extent of its adaptation to the environment [21].

Over successive generations, the recombination of genes and mutation promote the diversification of the population. This results in a population with different characteristics that are better suited to survive in a particular environment. The evolution process occurs constantly in nature and increases the diversity in the population of every living species [21].

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 [20, 21, 22]. The evolution of the population is obtained by successively applying these operators. 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. The basic execution steps of a generic evolutionary algorithm are explained below [21]:

- **Initialization**: In this step, the initial population is created, which can be of any size varying from a couple of individuals to thousands.

- **Evaluation**: Following the initialization of the population, each individual is evaluated according to the cost function. The evaluation aims to determine how much an individual fulfils the problem requirements. In other words, how well an individual can solve the target problem.

- **Selection**: This is the process of choosing which individuals in a population will be used to create the next generation. The selection is based on increasing overall fitness, which is achieved by selecting the individuals that are best suited to solve the target problem and discarding the ones that are not well adapted.

- **Reproduction**: During the reproduction or **crossover** phase, the genetic material of two individuals is combined to create the offspring. The main intention of reproduction is to combine the traits of fit individuals, thus increasing the fitness of the offspring.

- **Mutation**: Mutation is one of the most important drivers of change in the genetic pool. The mutation causes small changes in the individual's traits, which might increase (or reduce) its fitness. If the mutation benefits the individual it will be kept by the selection mechanism, otherwise, it will be removed within the next generations. In general, without mutation, the population would not present any improvement over time, as all the combinations that would ever be possible would be already available in the first iterations.

Together, these operators allow EA to explore the search space and converge on optimal solutions. The specific implementation of these operators can vary depending on the problem being solved and the particular algorithm used.

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 [22]. Nonetheless, they differ in terms of their specific methodologies. ES primarily focuses on individual-level behavioural modifications, EP centers on population-level behavioural adjustments, and GA places emphasis on chromosome operations. Furthermore, the field of evolutionary algorithms also encompasses other methodologies, including Genetic Programming and Memetic Algorithm, among others [22].

3.1.2 Evolutionary strategy

Evolutionary strategy is a member of the evolutionary algorithm family. The general idea behind this algorithm is to discover better solutions to a problem by introducing random perturbations to the current solution. The process of adding random perturbation to solutions is referred to as mutation, and it is one of the strongest characteristics of ES [23]. The mutation generates the offspring solutions which are evaluated based on their *fitness*, as described previously. 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 [23, 24], using a Gaussian distribution with zero mean and σ standard deviation, $\mathcal{N}(0, \sigma^2)$ to create the mutation of the offspring solutions. Furthermore, Eiben and Smith [23], describe the following steps to implement the ES:

Algorithm 1 Generic ES algorithm

```
set t=0;
Create initial point ⟨x_1^t, ..., x_n^t⟩ ∈ ℝ^n;
while TERMINATION CONDITION not satisfied do
    Draw z_i from the Gaussian distribution for all i ∈ {1, ..., n} independently;
    y_i^t = x_i^t + z_i^t for all i ∈ {1, ..., n};
    if (f(x_i^t) ≤ f(y_i^t)) then
        x^{t+1} = x^t;
    else
        x^{t+1} = y^t;
    end if
    Set t = t + 1
end while
```

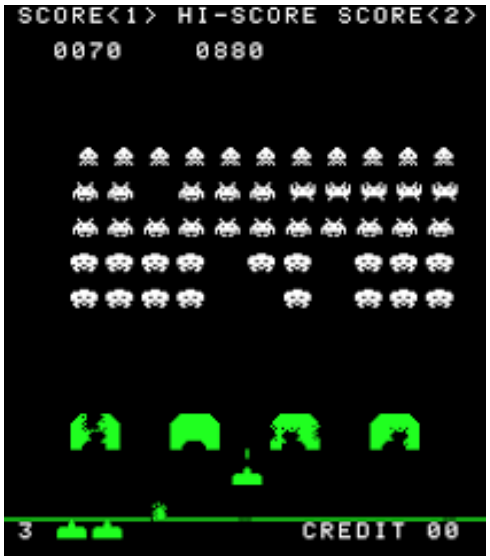
In Algorithm 1, x_i^t represents the initial solutions (or initial population), z_i^t is the random noise responsible to cause the mutation of the offspring, y_i^t denotes the offspring and t represents the generation of the population. In addition, $f(x_i^t)$ and $f(y_i^t)$ are the cost functions used to evaluate each generation of offspring.

In the recent versions of this approach, however, ES is portrayed as a black-box stochastic optimization technique [25], with less emphasis on its connection to biological evolution [26]. 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 [26].

The optimization process, in this case, can be seen as a form of *Reinforcement Learning* (RL) and it is described by the following steps [26]:

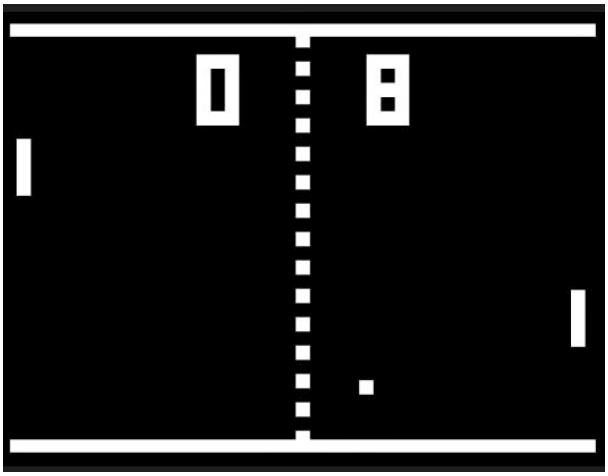
1. A parameter vector w is selected
2. Then, a group of modified parameter vectors $(w_1, w_2, \ldots, w_n)$ is created by adding random noise to w
3. Each modified parameter vector $(w_1, w_2, \ldots, w_n)$ is evaluated and a reward system is created to assign weights to the best-performing vectors.
4. The updated parameter vector is obtained by taking a weighted average of the n vectors, with higher weights assigned to more successful candidates.

The steps described above have, in fact, been successfully applied to two RL benchmark tasks: the Atari game-playing and the Multi-Joint dynamics with Contact (MuJoCo[1]) control tasks. On a higher level, the first task consists of designing a model to successfully achieve superhuman results in classic Atari games, such as Space Invaders or Pong, which are shown in Figure 3.1.1 and Figure 3.1.2.



Source: https://en.wikipedia.org/wiki/Space_Invaders

Figure 3.1.1: Classical Atari game: Space Invaders.



Source: https://www.gameblast.com.br/2014/05/pong-atari-blast-from-the-past.html

Figure 3.1.2: Classical Atari game: Pong

[1]https://mujoco.org/

The second task refers to a physics engine that is widely used for developing continuous control agents. The MuJoCo, offers a comprehensive platform for training simulation agents or robots. Within this environment, various simulations are available to test the capabilities of robotic control agents in tasks like walking, crawling, and other physics-based control challenges. The different MuJoCo environments are presented in Figure 3.1.3.

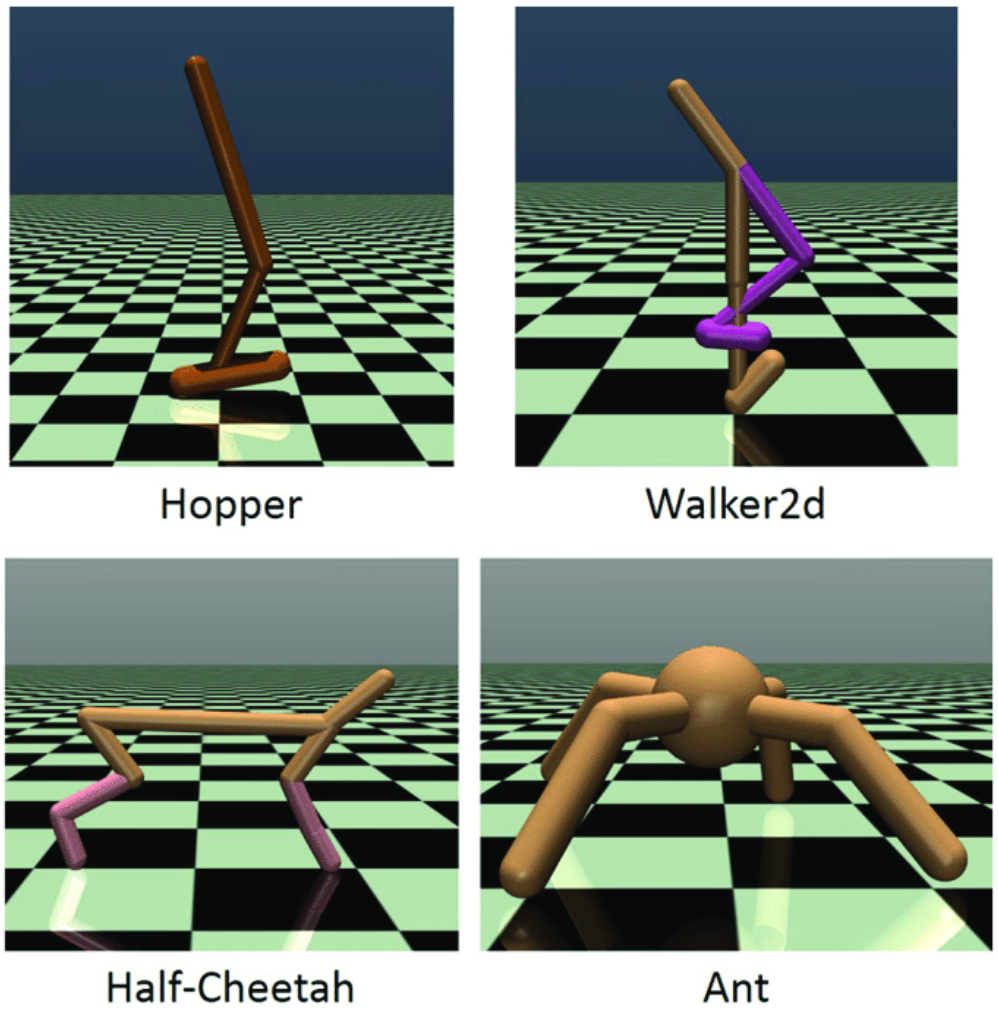


Figure 3.1.3: MuJoCo simulated environments: Hopper-v2, Walker2D-v2, Half-Cheetah-v2, and Ant-v2 [27]

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 [26]. This function is usually a neural network, that rewards the agent if it takes any of the allowed actions, for example, wins a game.

3.1.3 Designing evolutionary neural networks

The training process of a neural network involves finding optimal values for its parameters, which can be seen as an optimization problem. The conventional approach 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 [28]. In general, evolutionary algorithms 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 [29], but their application in continuous learning optimization remains unexplored, to the best of the authors' knowledge.

CHAPTER

FOUR

METHODOLOGY

This chapter presents a comprehensive description of the proposed approach, as well as a methodology for evaluating its performance. It begins by outlining the experimental setup, which includes the specific software, and datasets used for the experiments. Thereafter, the evaluation metrics used to measure the performance of the approach are presented. Finally, the experimental procedure used to develop the method is described.

4.1 Experimental setup

In the experiments included in this thesis, Python 3.10 was used for conducting the experiments. Libraries employed were standard data science libraries such as Keras[1], Matplotlib[2], NumPy[3], scikit-learn[4] and TensorFlow[5]. The code was run in a NVIDIA Tesla T4 GPU from Google Colab. Three different datasets were used for training and evaluating the evolutionary strategy:

- **MNIST** [30] 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** [31] 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.

[1] `https://keras.io/`
[2] `https://matplotlib.org/`
[3] `https://numpy.org/`
[4] `https://scikit-learn.org/`
[5] `https://www.tensorflow.org/`

- **CIFAR-10** [32] 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 required little preprocessing as they have no issues related to missing values, outliers etc. All input images were normalized by dividing them by 255. During the training of the neural networks, no data augmentation was used. In addition, the categorical labels were converted to binary vectors by applying one-hot encoding.

Given that each dataset consists of both training and test data, the preprocessing steps employed were standardized across them. In order to simulate practical applications where data becomes available at different time intervals, the training data was further divided by using an 80-20 split. The splits were done with and without stratification. Furthermore, in the context of the larger subset of training data, one class was intentionally excluded and allocated to the smaller subset. The scenario where no class was omitted was also taken into account, as shown in Table 4.1.1.

Table 4.1.1: Splitting strategies for the training data

Model	**Missing class**	**Stratified**
All classes (stratified)	False	True
All classes	False	False
Missing class (stratified)	True	True
Missing class	True	False

Table 4.1.2 summarizes the new dimensions of the training data after splitting. The noticeable reader might see that the dimensions are different for the two last datasets depending on whether the data is stratified or not. Both the training data for Fashion-MNIST and CIFAR-10 dataset contains exactly 6000 and 5000 images of each class, respectively. For the MNIST digit dataset, this varies. For more details regarding the class distributions, please refer to Appendix C.

Table 4.1.2: Dimensions of subsets after splitting

Dataset	Model	Large subset	Small subset
MNIST	All classes, incl. stratified	48,000	12,000
	Missing Class, incl. stratified	43,266	16,734
Fashion-MNIST	All classes, incl. stratified	48,000	12,000
	Missing Class, stratified	43,200	16,800
	Missing Class	43,145	16,855
CIFAR-10	All classes, incl. stratified	40,000	10,000
	Missing Class, stratified	36,000	14,000
	Missing Class	35,996	14,004

4.2 Experimental procedure

The primary concept underlying this approach involves the introduction of perturbations to the weights and biases within a neural network. Initially, a model was trained using the larger subset of the training data. Subsequently, the model was duplicated, and random noise was introduced to the weights and biases to simulate the mutation process. Each newly created model was then trained using the smaller subset (i.e. fresh data in a practical setup), and its accuracy was evaluated. This iterative process was repeated for a specified number of iterations, and the accuracy of each mutated model was employed to calculate a weighted average for the weights and biases. This information was utilized to generate a new model that was subsequently applied to predict the classes of unseen data. The experimental procedure is illustrated in Figure 4.2.1.

For each of the neural network architectures described in Appendix D, a model was initialized with random weights and biases. Thereafter, a clone of this model, hereby defined as 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, $\{M_{0_1}, \ldots, M_{0_N}\}$ where N is the number of desired mutations. 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`.

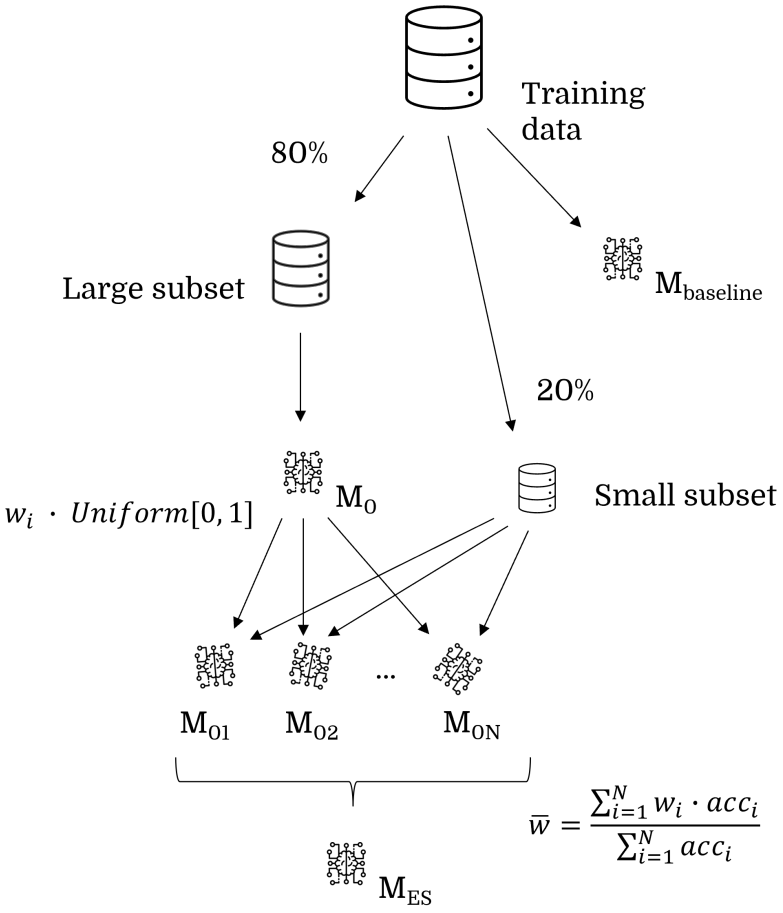


Figure 4.2.1: Evolutionary strategy process

The Equation 4.1 was applied to obtain the weighted averages of the weights and biases.

$$\bar{w} = \frac{\sum_{i=1}^{N} w_i \cdot acc_i}{\sum_{i=1}^{N} acc_i} \tag{4.1}$$

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 $\{M_{0_1}, \ldots, M_{0_N}\}$.

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.

The experimental procedure is outlined in Algorithm 2.

Algorithm 2 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 smaller subset of training data and validation split of 10%
  Evaluate the model
  Save accuracy
end for
Calculate the weighted average of weights and biases using Equation 4.1
Compile and evaluate the offspring model, M_{ES}, utilizing test dataset

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.

4.3 Evaluation metrics

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

4.3.1 Confusion Matrix

The confusion matrix is a convenient tool used to evaluate the performance of a classifier. Essentially, it 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 [33]. 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.

Figure 4.3.1 visually represents the fundamental concept of the confusion matrix. By examining individual classes, valuable insights can be gained regarding potential outcomes and the model's performance evaluation. Each class presents four possible outcomes: True Positive (TP), False Positive (FP), False Negative (FN), and True Negative (TN) [34].

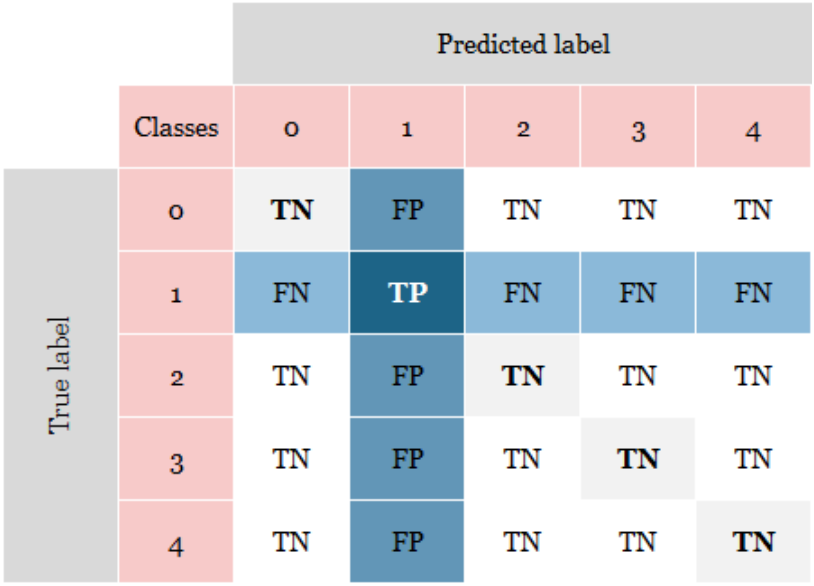


Figure 4.3.1: Multi-class confusion matrix [34]

In Figure 4.3.1, TP signifies that the model accurately predicted the corresponding class. In this case, class 1 was correctly predicted as 1. Conversely, FP indicates that the model assigned an incorrect label, such as class 0 being misclassified as 1. FN indicates that the model misclassified a different class as 1. Lastly, TN indicates that the model correctly identified the sample as not belonging to class 1 when the true label of the sample was indeed not 1.

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 chosen 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.

4.3.2 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 [33]. Mathematically, accuracy can be calculated as:

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

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

CHAPTER

FIVE

RESULTS

This chapter offers an overview of the results obtained from the previously described experiments. The results are presented in a list and visually represented to enable an overall performance comparison. The confusion matrix for each model is provided for both the baseline and best-performing ES models. The former is shown on the left-hand side, while the latter is displayed on the right. For simplicity, the losses can be found in Appendix E. Furthermore, class number 6 was excluded from the larger subsets of the training data, and added back to the smaller portion of training data, when applicable. This process aims to imitate a real-world situation in a production setting where new classes are introduced. This allows the model to be trained on additional data in an incremental manner. In addition, in cases where it is relevant, the data was divided using either stratified or non-stratified sampling methods. The former ensures that the model receives an adequate number of examples from each category, resulting in improved generalization and performance. On the other hand, the latter approach may result in an imbalanced distribution of classes, with some categories having considerably fewer samples compared to others. The authors have observed that this mirrors real-world applications more accurately. Lastly, the models were evaluated using the test data.

5.1 MNIST - MLP

The MNIST dataset was used to train a Multilayer Perceptron (MLP) model as summarized in Table D.1 in the Appendix. The same model architecture was employed to develop the $M_{baseline}$, M_0 and five M_{ES} models. The accuracy obtained is presented in Table 5.1.1.

Table 5.1.1: Accuracy of MNIST dataset (MLP)

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
All classes (stratified)	0.9817	0.9792	0.9754	0.9754	0.9760	**0.9761**	0.9756
All classes	0.9797	0.9803	0.9744	0.9741	0.9749	**0.9753**	0.9751
Missing class (stratified)	0.9809	0.8846	0.9754	0.9764	**0.9767**	0.9763	0.9766
Missing class	0.9810	0.8872	0.9742	0.9743	0.9742	0.9744	**0.9748**

It is possible to notice that the resulting accuracy values obtained for all the cases evaluated demonstrate consistent stability regardless of the number of mutations applied. No signs of catastrophic forgetting were evident. The results for each case are explained in detail below.

All classes, stratified split

Figure 5.1.1 provides a detailed presentation of the performance of the MLP on the MNIST dataset for all classes, using stratified split. It showcases the results for the initial model M_0, the subsequent M_{ES_N} models, and the accuracy of the baseline model (represented by a dashed line). When examining the performance across all classes with training data split in a stratified manner, it becomes apparent that the ES models achieve nearly identical results (accuracy ranging from 0.9754 to 0.9761), regardless of the number of mutations used to generate the offspring models. It is also worth noting that both the baseline model and the M_0 model exhibit slightly better performance, 0.9817 and 0.9792 respectively.

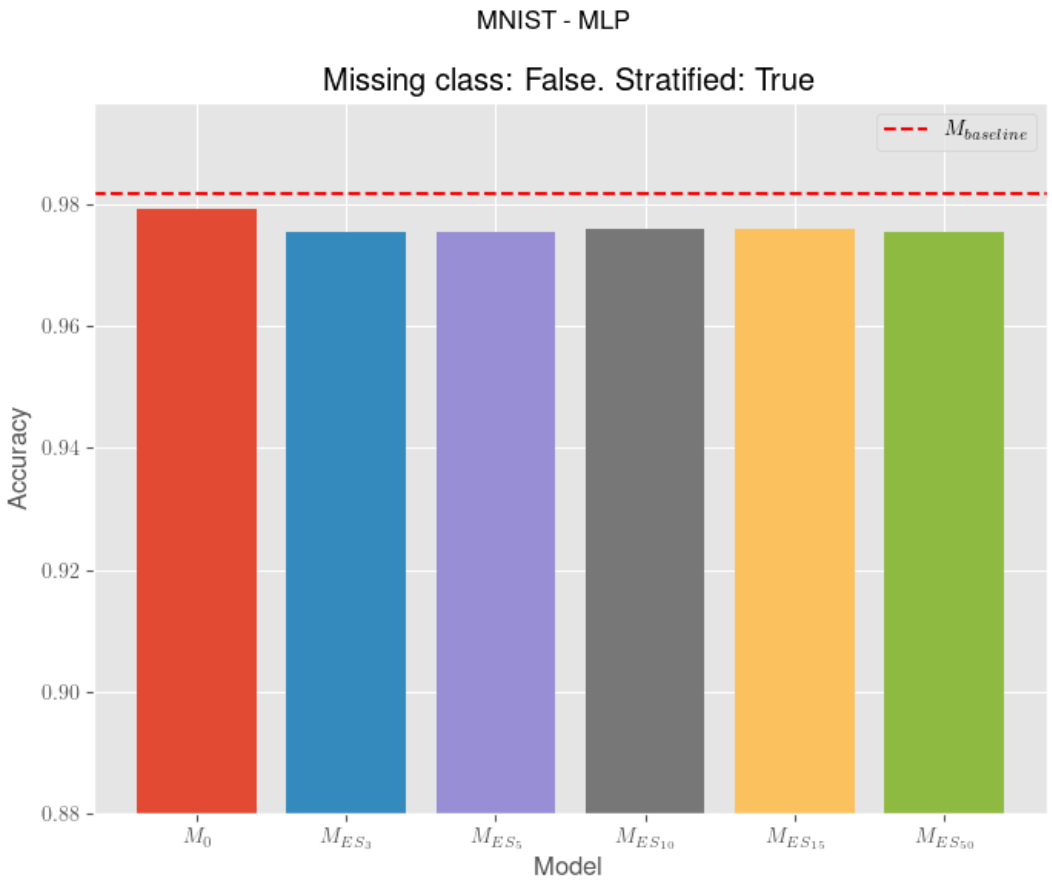


Figure 5.1.1: Accuracy of MNIST - MLP considering all classes (stratified)

Additionally, Figure 5.1.2 illustrates the confusion matrices obtained from the first experiment. It showcases the performance of the baseline model and the best-performing ES model, in this case, $M_{ES_{15}}$. In most classes, the baseline outperforms the offspring model. However, it is possible to notice that the ES model performs slightly better in classes 3 and 7. Misclassifications in the offspring model were most noticeable where class 5 was predicted as class 3.

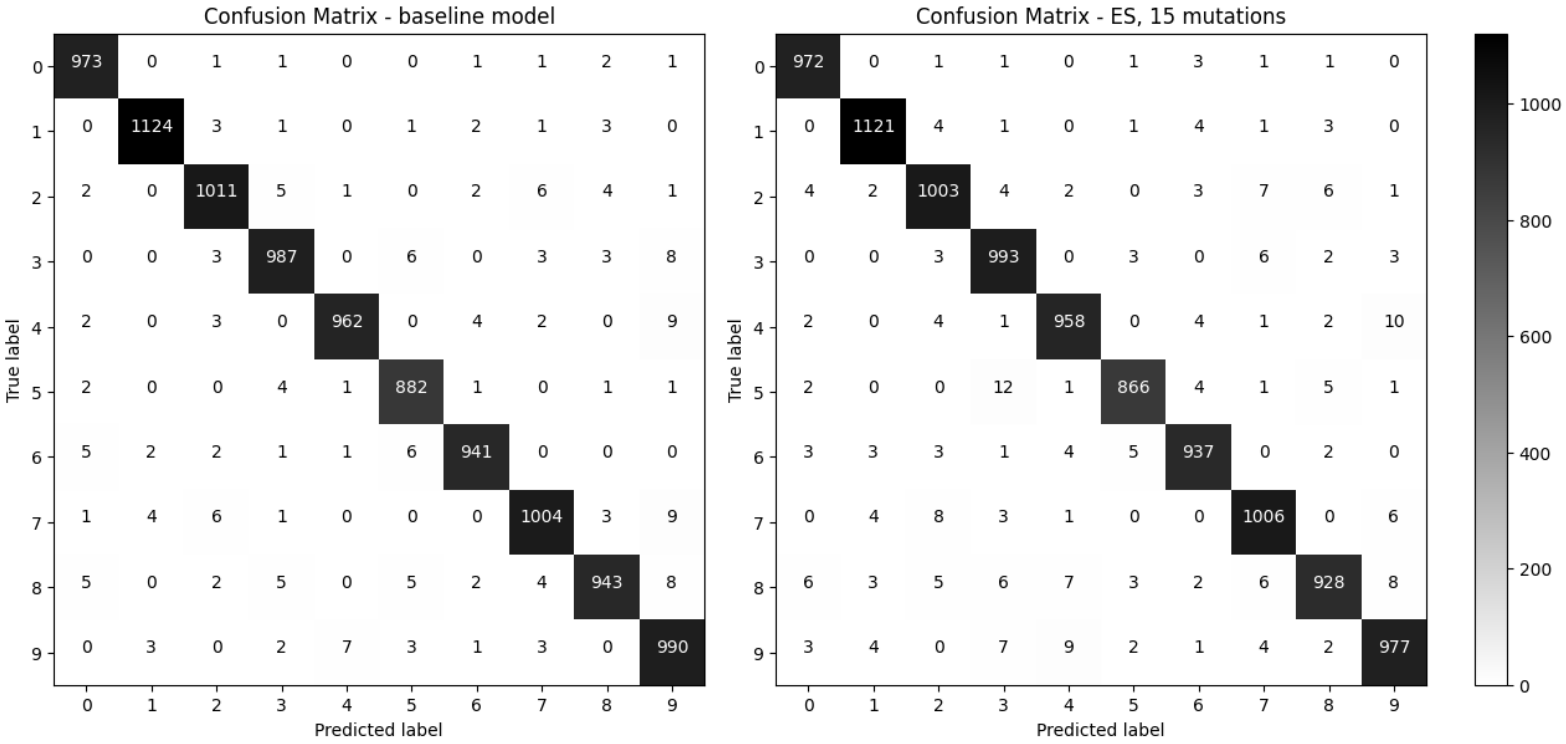


Figure 5.1.2: Confusion matrices for MNIST - MLP, all classes (stratified)

All classes, non-stratified split

Figure 5.1.3 illustrates the evaluation of all classes in the MNIST dataset utilizing a random split of the training data, *without* stratification. Regardless of the number of mutations used, the ES models consistently demonstrate almost identical performance, with the lowest accuracy being 0.9741 and the highest being 0.9753. Similar to the previous case, both the baseline model and the model M_0 exhibit relatively better performance. However, in this instance, the model M_0 achieves slightly higher accuracy compared to the baseline model, 0.9803 and 0.9797 respectively.

Figure 5.1.3: Accuracy of MNIST - MLP considering all classes

The confusion matrices depicted in Figure 5.1.4 provide insights into the accuracy of both the baseline model and the best-performing ES model, encompassing all classes without stratification. Notably, there are significant performance differences in classes 2, 4, and 8. In general, the baseline model outperforms the ES model, with the exception of class 8. It is observed that the offspring model demonstrates fewer misclassifications of class 9 as 4 compared to the baseline model but exhibits misclassifications as class 8. Furthermore, it is worth noting that class 3 is misclassified as 7 by the offspring model, whereas the baseline model did not exhibit any misclassifications in this particular case.

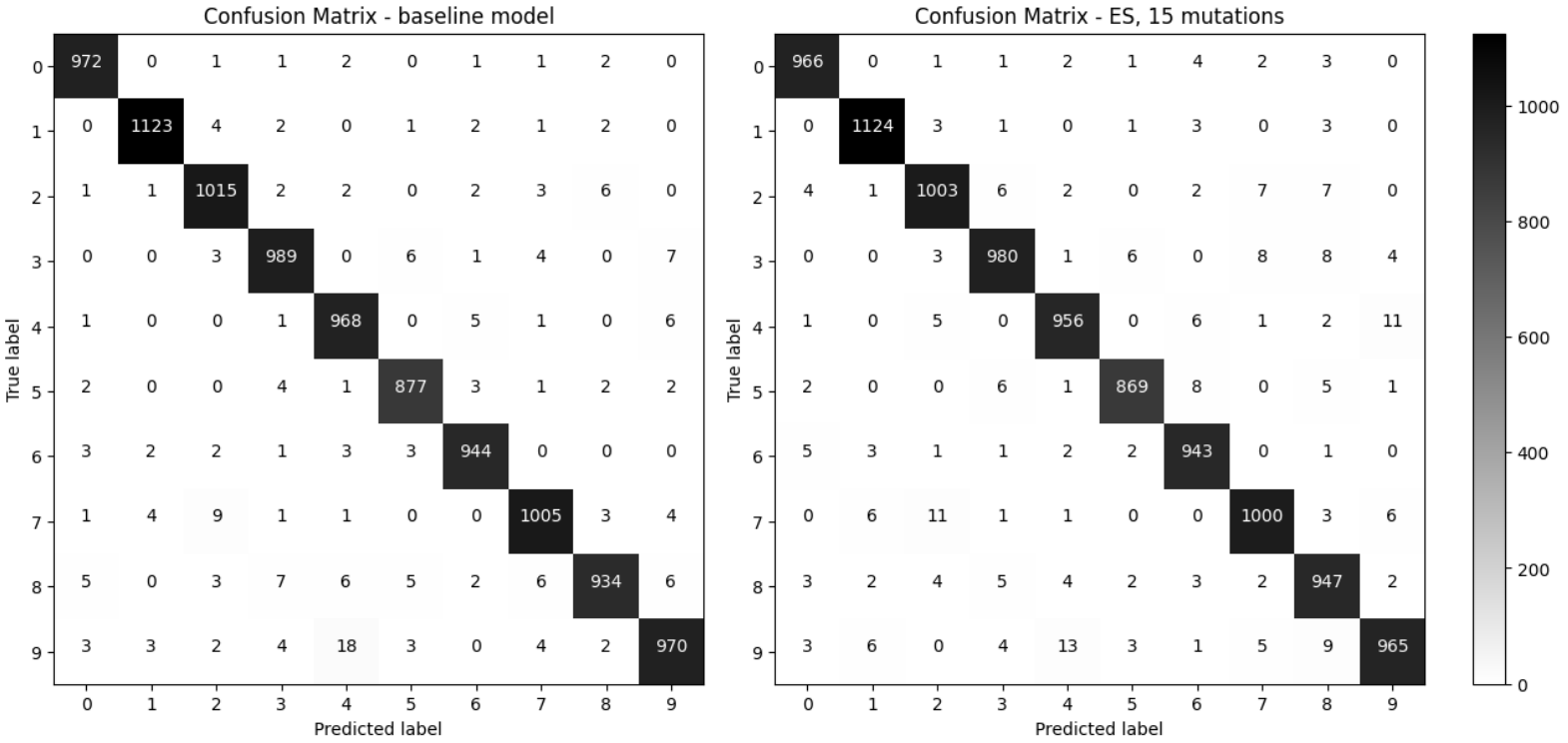


Figure 5.1.4: Confusion matrices for MNIST - MLP, all classes

Missing class, stratified split

Figure 5.1.5 showcases the performance when considering a missing class and a stratified split of data. A notable difference can be observed in the performance of the M_0 model compared to the ES models, 0.8846 for the M_0 model and 0.9767 for the best-performing ES model. This difference was expected since a single class was excluded from the training data for the M_0 model. As a result, the missing class was misclassified, leading to a decrease in overall accuracy. However, it is worth noting that the ES models perform remarkably well, nearly matching the performance of the baseline model. This suggests that the models were capable of adapting to new data, even when its distribution changed and a new class was introduced.

Figure 5.1.5: Accuracy of MNIST - MLP, missing class (stratified)

Figure 5.1.6 displays the confusion matrices for both the baseline model and the best-performing ES model, offering an assessment of their resilience when a new class is introduced to the training data. Notably, there are significant differences in the performance for correctly predicted classes in classes 2, 8, and 9. In general, the baseline model outperforms the ES model, with the exception of class 8. Additionally, there are no noticeable differences in the misclassification patterns between the two matrices. However, it is evident from the obtained confusion matrix that the ES model swiftly learns the new pattern, indicating its ability to adapt and incorporate new information effectively.

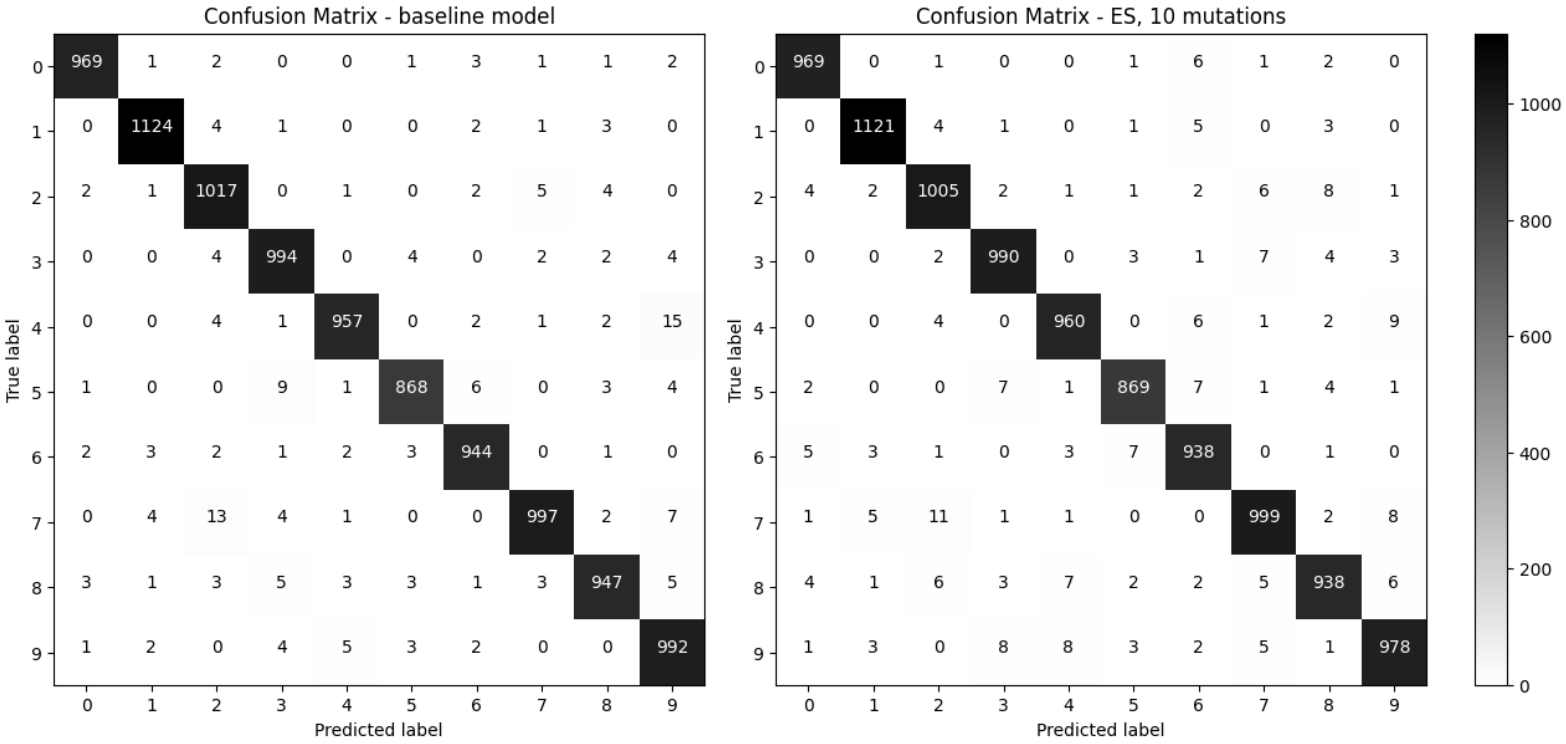


Figure 5.1.6: Confusion matrices for MNIST - MLP, missing class (stratified)

Missing class, non-stratified split

Figure 5.1.7 displays the performance of the models, while Figure 5.1.8 compares the performance of the baseline model and the best-performing ES model when considering the training data with a single class removed and a non-stratified split. In this scenario, the ES models outperformed the M_0 model, achieving respective results of 0.9748 and 0.9810 for the best-performing ES model. Furthermore, all ES models attained similar results, indicating consistent performance.

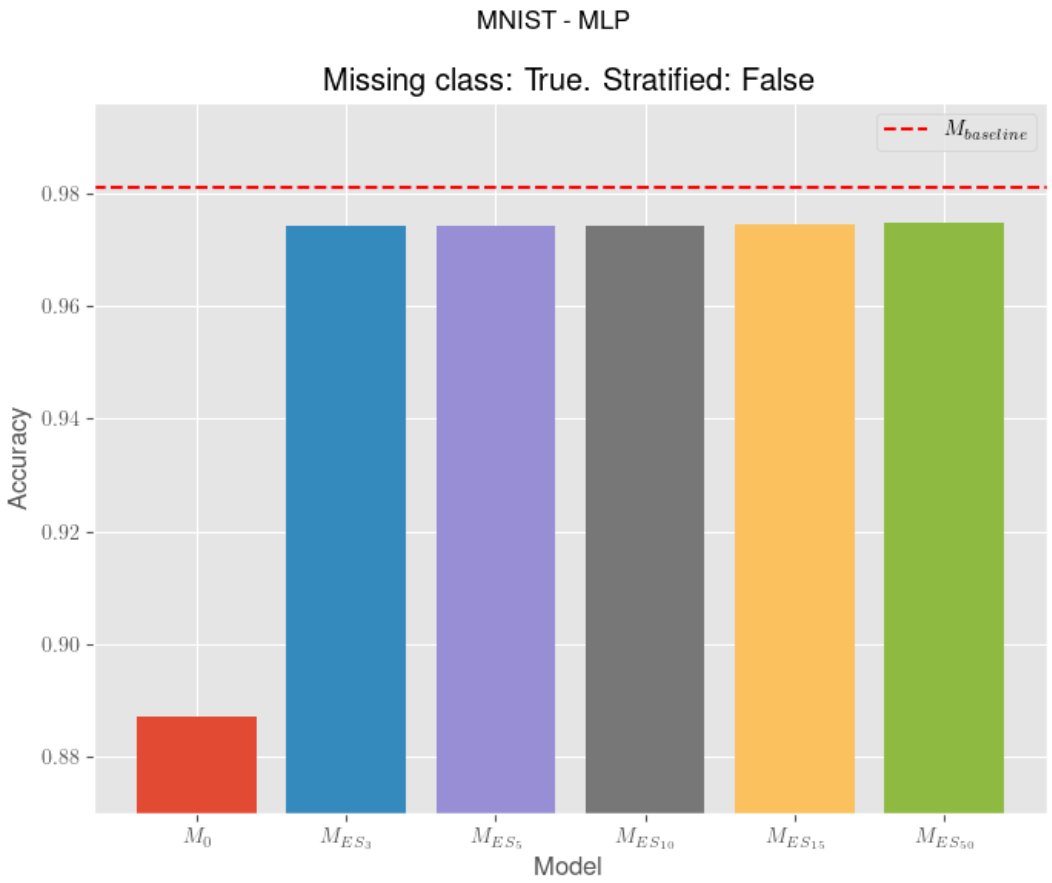


Figure 5.1.7: Accuracy of MNIST - MLP, missing class

When examining the confusion matrices in Figure 5.1.8, it is evident that the ES model performs admirably well, despite some notable disparities, especially in classes 8 and 9, where the baseline model exhibits superior performance. The ES model, on the other hand, demonstrates a reduced number of misclassifications for class 4, particularly in cases where it predicts class 9. Additionally, a noticeable distinction between the models is their handling of class 9, which is occasionally mispredicted by the ES model as class 4.

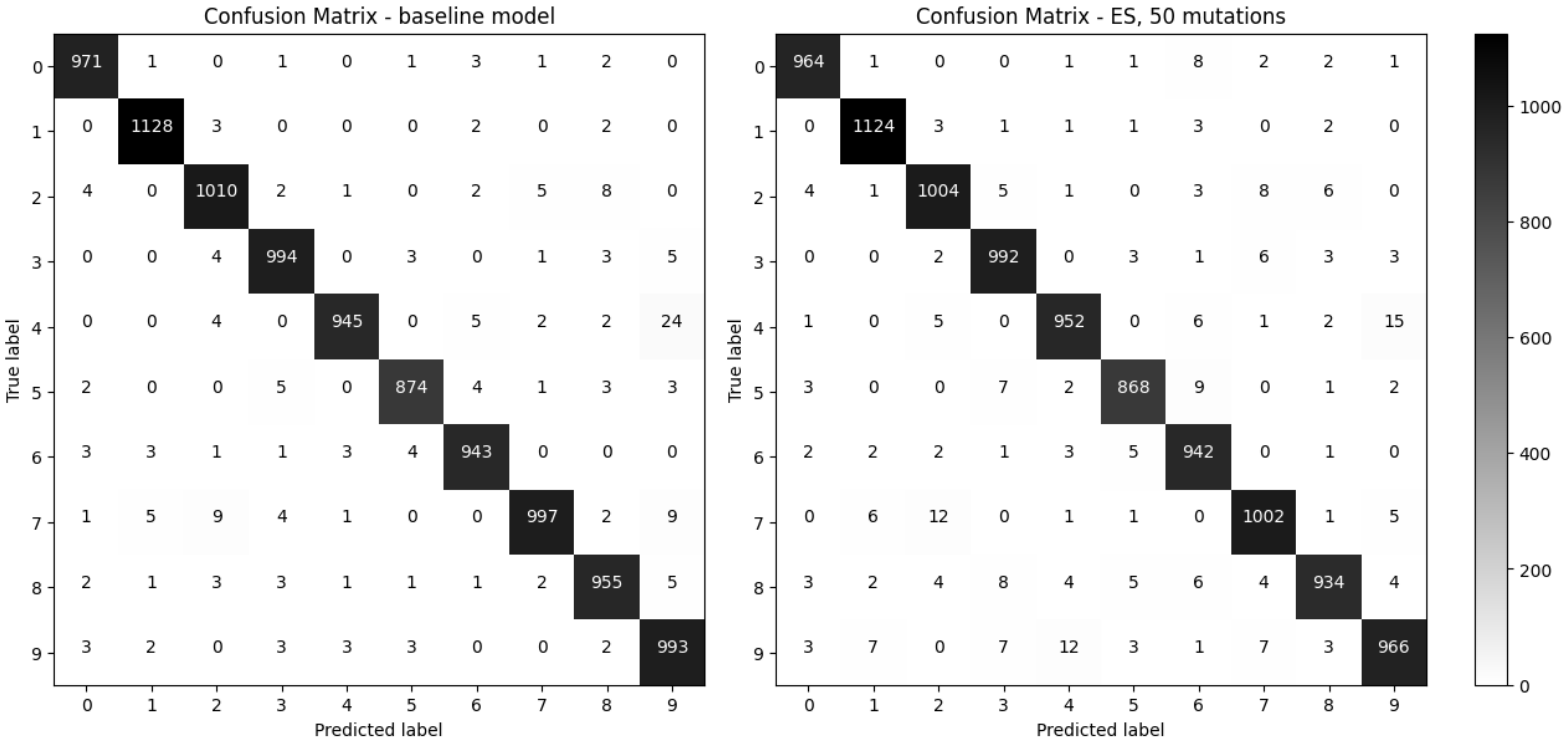


Figure 5.1.8: Confusion matrices for MNIST - MLP, missing class

5.2 MNIST - CNN

Table 5.2.1 presents the accuracy of the ES models, as well as the baseline model and the M_0 model, utilizing a Convolutional Neural Network (CNN). The same architecture was employed to develop the models described, see Table D.2 in the Appendix for more details. Remarkably, the accuracy of the ES models remains consistent regardless of the number of mutations applied. When a random class is removed, it is notable that the ES model quickly adapts to the new patterns, resulting in an increase in accuracy. This suggests that catastrophic forgetting has not occurred, as the model retains its previously learned knowledge while incorporating new information.

Table 5.2.1: Accuracy of MNIST dataset (CNN)

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
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

The performance of the MNIST dataset using CNN is also presented in more detail below.

All classes, stratified split

The results of training a CNN model on the MNIST dataset, utilizing all classes and a stratified split, are showcased in Figure 5.2.1. The accuracy of the ES models were marginally higher, with the best-performing model achieving an accuracy of 0.9899 and the worst performer achieving an accuracy of 0.9886, surpassing the accuracy of the model M_0 at 0.9885. However, the ES models' accuracy remained just below that of the baseline model, which achieved 0.9903 accuracy. Additionally, it is possible to observe that the accuracy tended to increase as the number of mutations used in the ES models was augmented.

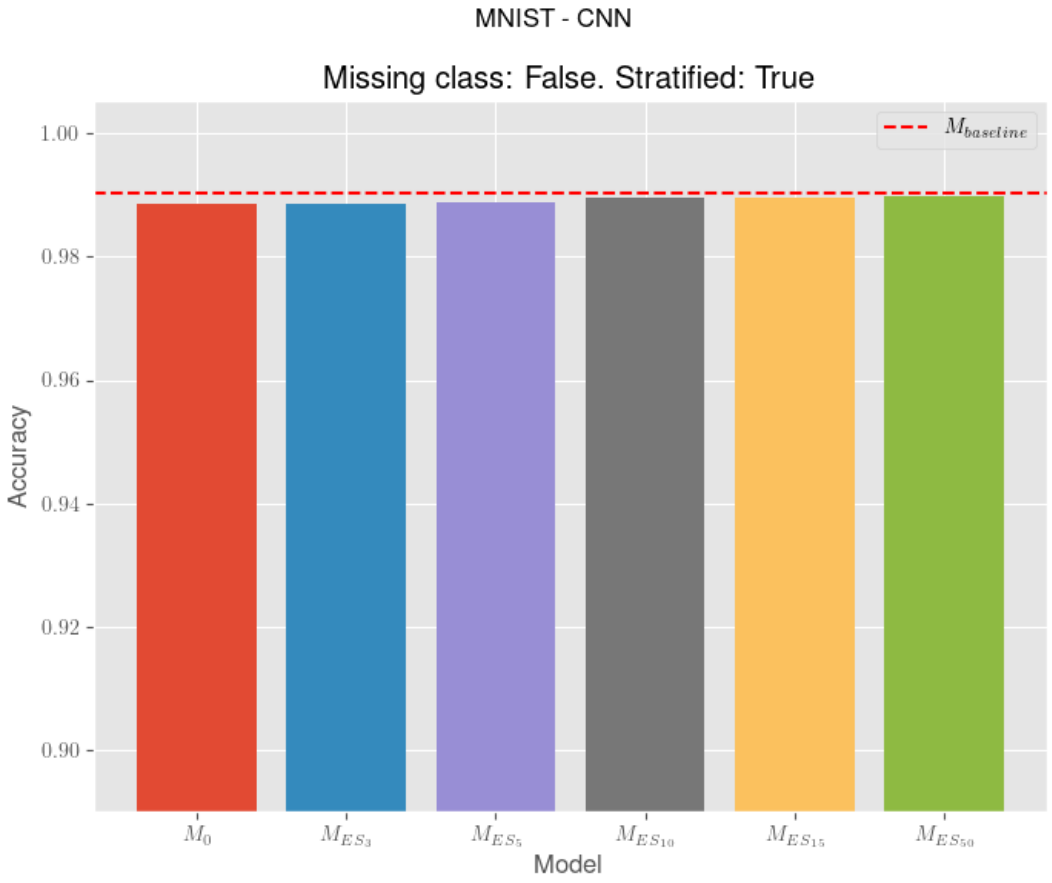


Figure 5.2.1: Accuracy of MNIST - CNN considering all classes (stratified)

The confusion matrices depicted in Figure 5.2.2 show that the classification accuracy is close to each other. However, it is worth noticing that the ES model predicted class number 5 better than the baseline model. There are no other noticeable differences in misclassification patterns between the two matrices.

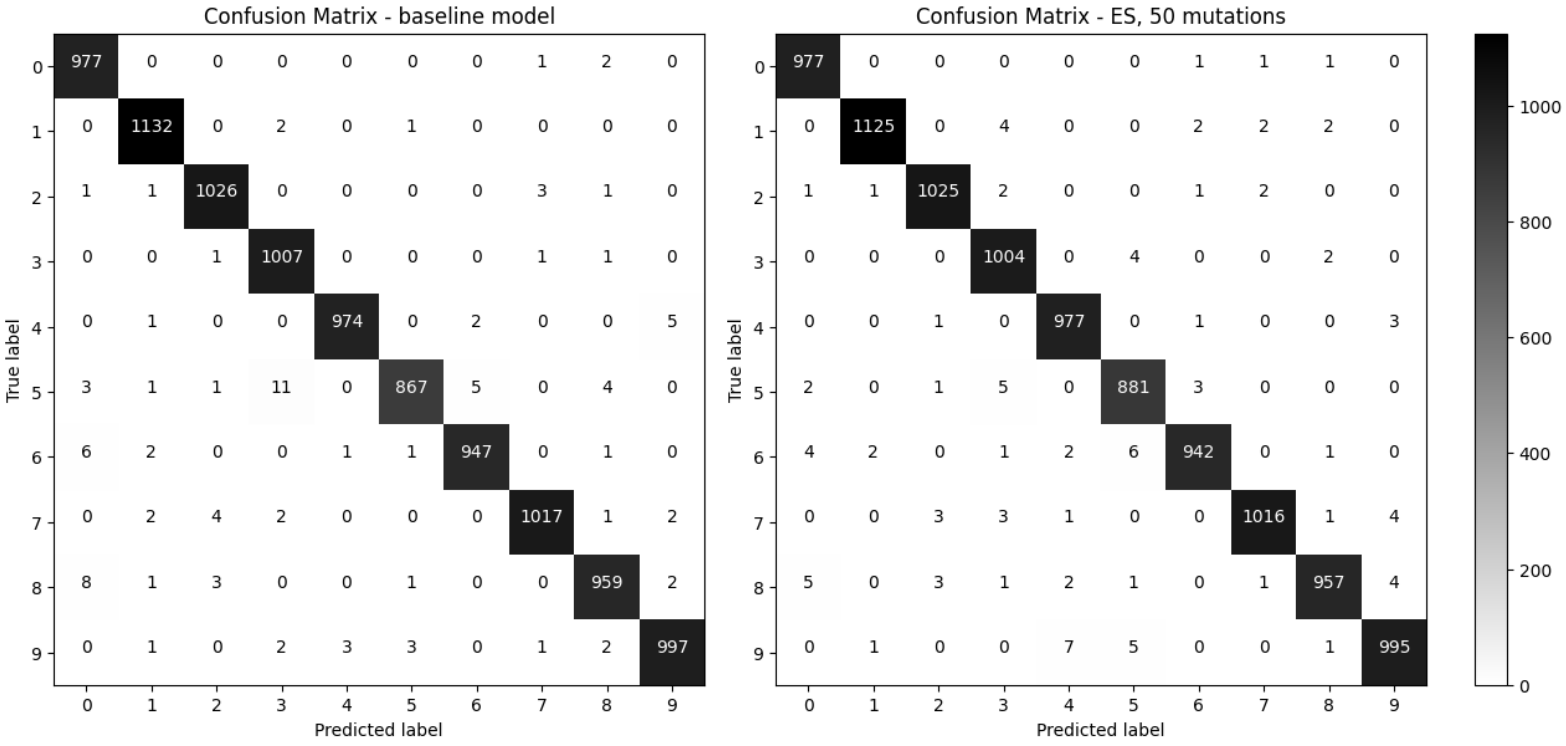


Figure 5.2.2: Confusion matrices for MNIST - CNN, all classes (stratified)

All classes, non-stratified split

Figure 5.2.3 presents the outcomes of training a CNN on the MNIST dataset, considering all classes, and using a training data split without stratification. Notably, the ES models achieved comparable results, with the best-performing model attaining an accuracy of 0.9899 and the worst-performing model achieving an accuracy of 0.9892. Similarly, the M_0 model achieved an accuracy of 0.9890. It is apparent that all models exhibited accuracy close to that of the baseline model, which achieved an accuracy of 0.9907.

Figure 5.2.3: Accuracy of MNIST - CNN considering all classes

In addition, Figure 5.2.4 shows the confusion matrices for the baseline model and the best-performing ES model. The most significant difference observed in this comparison appears in class number 9. Notably, the ES model demonstrates superior classification accuracy for classes 0, 1, 5, 6, and 8, which collectively account for half of all classes in the dataset.

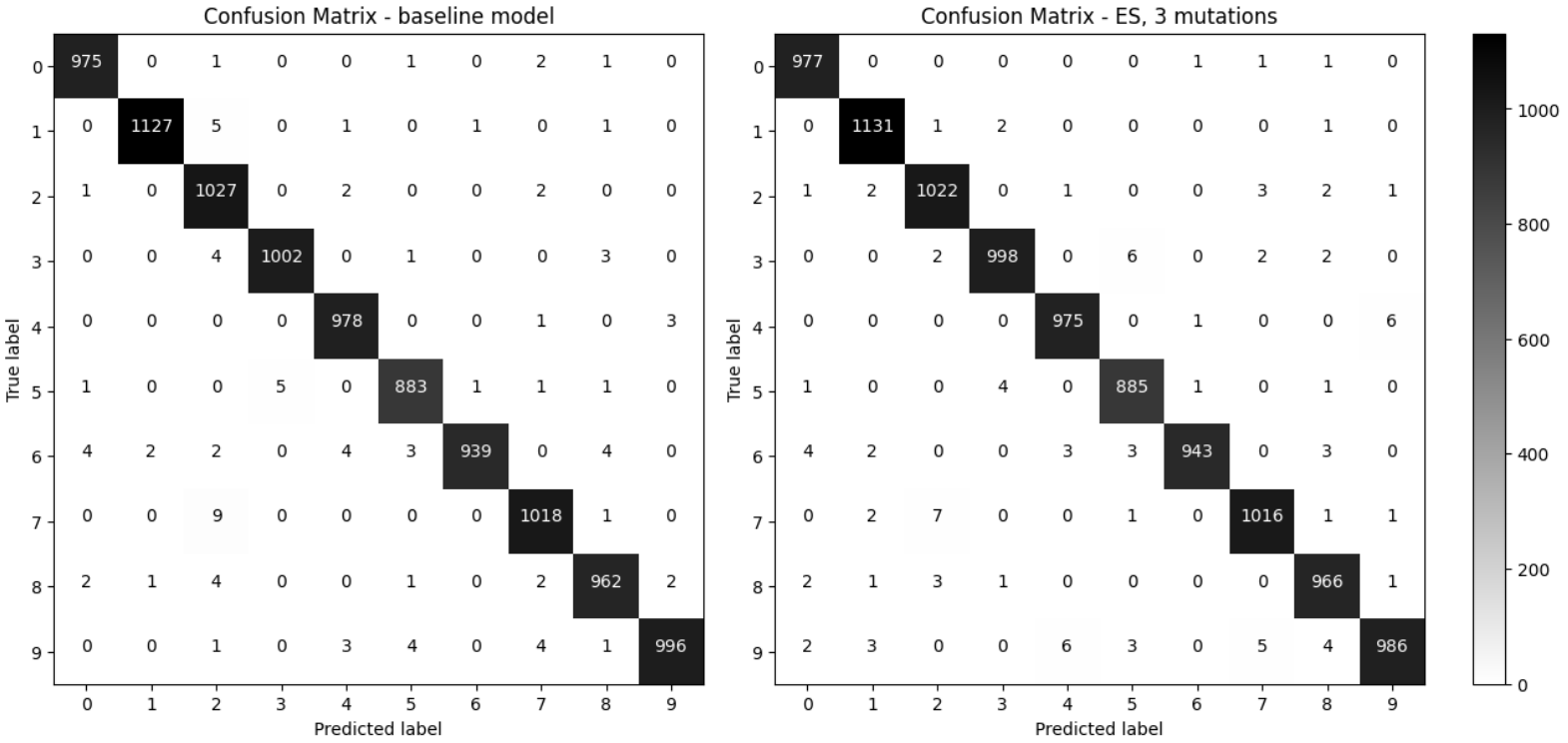


Figure 5.2.4: Confusion matrices for MNIST - CNN, all classes

Missing class, stratified split

In this scenario, a stratified split of the training data was performed, along with the removal of a specific class. A significant difference in accuracy can be observed between the M_0 model and the ES models. The accuracy for the M_0 model was 0.8959, while the best-performing ES model achieved an accuracy of 0.9886, as depicted in Figure 5.2.5. The disparity arises from the absence of class 6 during the training of the M_0 model, leading to misclassification of these samples in the test set.

Additionally, it is worth noting that the accuracy of all ES models closely resembled that of the baseline model, which was 0.9895. This indicates that the ES models successfully learned the new class and effectively adapted to the changes in data distribution during the mutation process.

Figure 5.2.5: Accuracy of MNIST - CNN, missing class (stratified)

Figure 5.2.6 shows the overall classification performance for the baseline model and the best-performing ES model. It is possible to notice that the models' performance was quite similar.

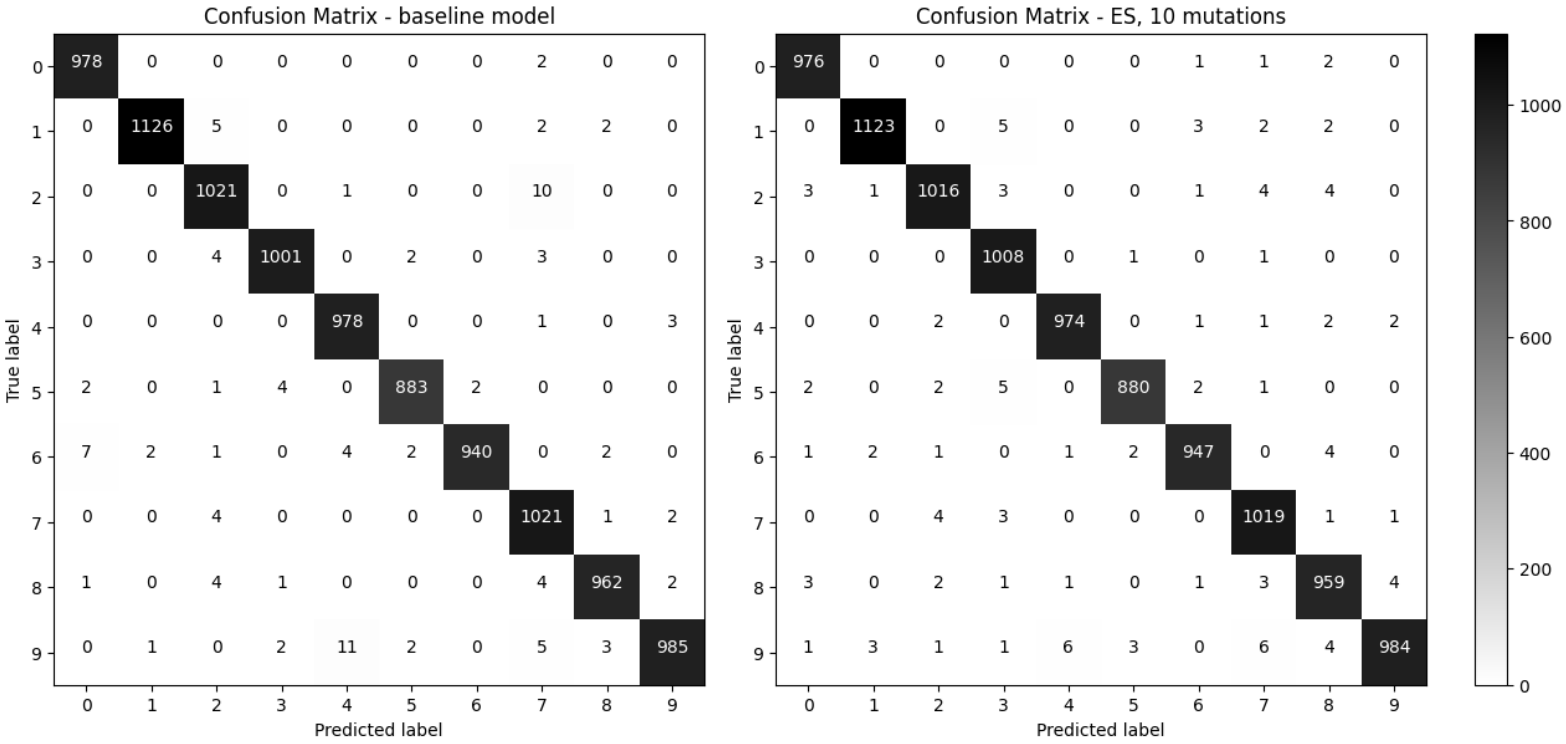


Figure 5.2.6: Confusion matrices for MNIST - CNN, missing class (stratified)

Missing class, non-stratified split

The performance of the models, in the case where the training data was split without stratification and one class was removed, is presented in Figure 5.2.7. Similar to the previous case, the ES models demonstrated a performance close to that of the baseline model, which was 0.9895. In this case, the best-performing ES model achieved an accuracy of 0.9878 and the "worst-performing" ES model exhibited an accuracy of 0.9877. These results showed increased performance when compared to the model M_0, which yielded an accuracy of 0.8960.

Figure 5.2.7: Accuracy of MNIST - CNN, missing class

The comparison between the confusion matrices for the baseline model and the best-performing ES model is presented in Figure 5.2.8. Notably, the baseline model exhibited superior performance in classes 2, 7, and 9. Conversely, the ES model surpassed the baseline model in classes 5 and 7, demonstrating an enhanced capability in predicting these specific categories. Additionally, no discernible differences in the misclassification patterns are observed between the two matrices.

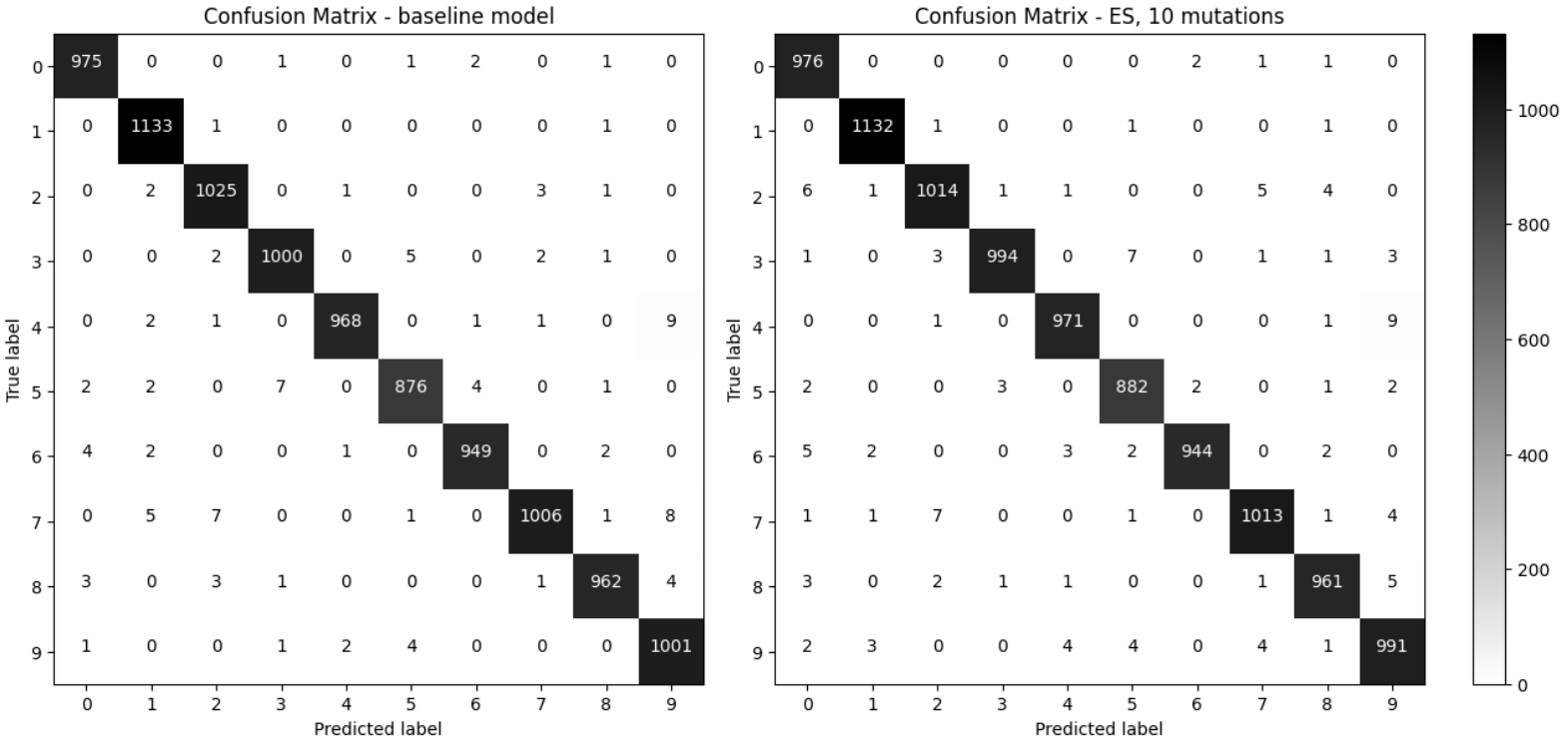


Figure 5.2.8: Confusion matrices for MNIST - CNN, missing class

5.3 Fashion-MNIST - CNN

In this section, a CNN model was developed for the Fashion-MNIST dataset. All cases were evaluated using an identical architecture, which is presented in Table D.2 in the Appendix. The accuracy for the $M_{baseline}$ model, as well as the M_0 model and the ES models is presented in Table 5.3.1.

Table 5.3.1: Accuracy of Fashion-MNIST dataset

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
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

The results for the Fashion-MNIST dataset are presented in detail below.

All classes, stratified split

Figure 5.3.1: Accuracy of Fashion-MNIST considering all classes (stratified)

The results of models trained on all classes with a stratified data split are illustrated in Figure 5.3.1. It is evident that in this particular scenario, the model M_0 exhibits slightly higher accuracy compared to both the baseline model and the ES models. The best-performing ES model achieved an accuracy of 0.8952, while the worst-performing model achieved 0.8914. In addition, the accuracy for the baseline model was 0.9043 and the accuracy for the model M_0 was 0.8546. It is important to notice that the ES models demonstrate commendable performance, with some closely resembling the accuracy of the baseline model.

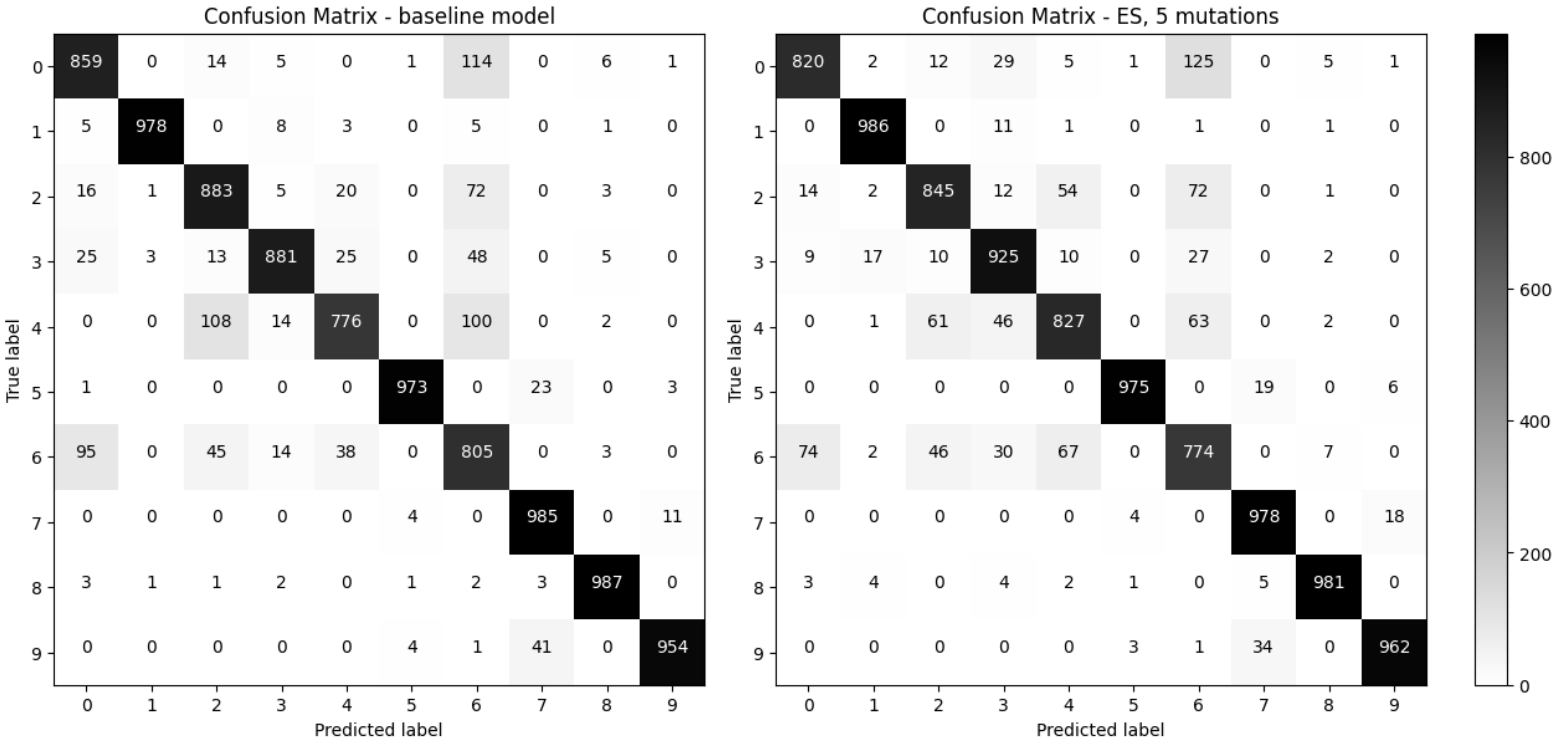


Figure 5.3.2: Confusion matrices for Fashion-MNIST, all classes (stratified)

The confusion matrices presented in Figure 5.3.2 provide a visual representation of the baseline model as well as the best-performing ES model. The overall misclassifications observed in this dataset were more pronounced compared to the previous dataset, given its increased complexity. Notably, the ES model demonstrates improved performance specifically in classes 3 and 4, where the observed increase in the correct classified samples (true positives) was 5.0% and 6.6%, respectively. In addition, the number of false positives increased for classes 3 and 4, compared to the baseline model. However, the number of false negatives decreased for the same classes. A similar pattern can be observed in the other scenarios analysed in this work. In every case, there were *at least* two instances where the ES models outperformed the baseline model in predicting specific classes.

All classes, non-stratified split

Figure 5.3.3 displays the models' accuracy when all classes are taken into account and the training data is split without stratification, which may result in class imbalance. The accuracy of the ES models changed slightly more than what has been seen previously. The best-performing ES model achieved an accuracy of 0.9046, while the worst-performing model attained an accuracy of 0.8945. Interestingly, some ES models surpassed the accuracy of M_0, which was 0.9000. However, it is important to note that all models displayed lower accuracy than the baseline model, which obtained an accuracy of 0.9066.

Figure 5.3.3: Accuracy of Fashion-MNIST considering all classes

Figure 5.3.4 further reinforces the observed trend. The confusion matrices indicate that class 6 poses a greater challenge for the ES model when the data is not stratified. Conversely, the ES model demonstrates improved predictions for class 4. This discrepancy is attributed to the absence of a stratified split, which introduces variations in the distribution of data types across classes.

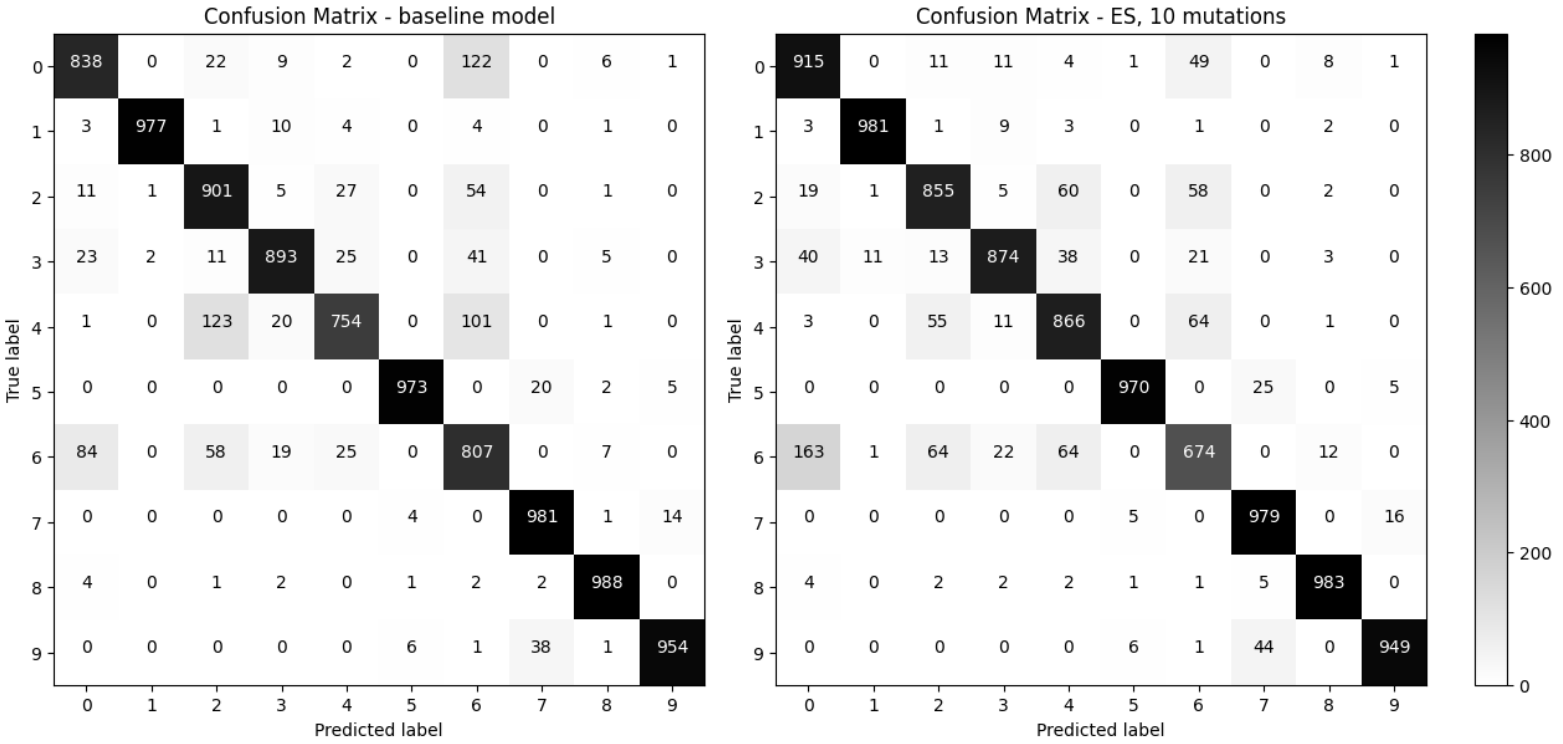


Figure 5.3.4: Confusion matrices for Fashion-MNIST, all classes

Missing class, stratified split

In this case, it was considered a missing class and stratified split. The results are illustrated in Figure 5.3.5. The model M_0 achieved the lowest accuracy (0.8579) due to the missing class. Interestingly, the accuracy of the ES models seemed to increase with the number of mutations up until $M_{ES_{50}}$. Here, the best-performing model attained an accuracy of 0.8923, while the worst-performing achieved an accuracy of 0.8903. Furthermore, the baseline model exhibited the highest overall accuracy, which was 0.9022.

Figure 5.3.5: Accuracy of Fashion-MNIST, missing class (stratified)

Figure 5.3.6 presents similar trends to the previous confusion matrices for the given dataset. However, there is a notable difference as class 6 is not present in model M_0. 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.

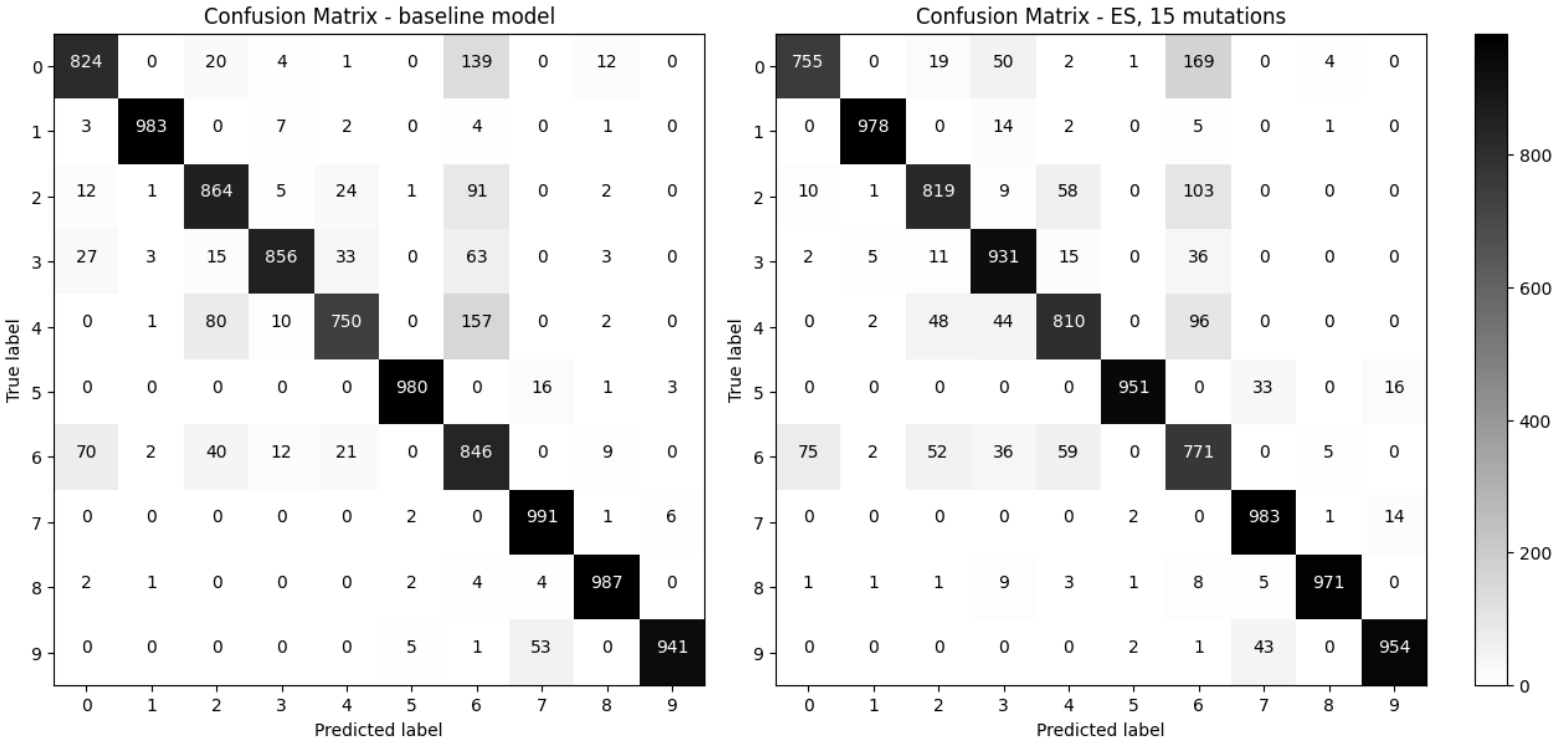


Figure 5.3.6: Confusion matrices for Fashion-MNIST, missing class (stratified)

Missing class, non-stratified split

Figure 5.3.7 presents the performance of the models when considering a missing class and a non-stratified split. It is clear that the ES models, with the best accuracy at 0.8952 and the worst-performing at 0.8914, outperformed the M_0 model, which achieved an accuracy of 0.8546. This indicates the successful adaptation of the ES models to the new class. As anticipated, the baseline model exhibited even higher accuracy, reaching 0.9043.

Figure 5.3.7: Accuracy of Fashion-MNIST, missing class

The confusion matrices depicted in 5.3.8 provide similar insights to the previous ones. It is noteworthy that this model (M_{ES_5}) exhibited the highest accuracy in predicting class 6 among all the cases presented in this section, for the Fashion-MNIST dataset. As expected, the baseline model consistently outperformed all other models. On the other hand, it is crucial to acknowledge that developing a model trained on the complete dataset may pose feasibility and scalability challenges in real-world contexts. Given these considerations, a more relevant comparison can be drawn between the ES models and the model M_0, which, in this context, was outperformed by the outlined evolutionary approach.

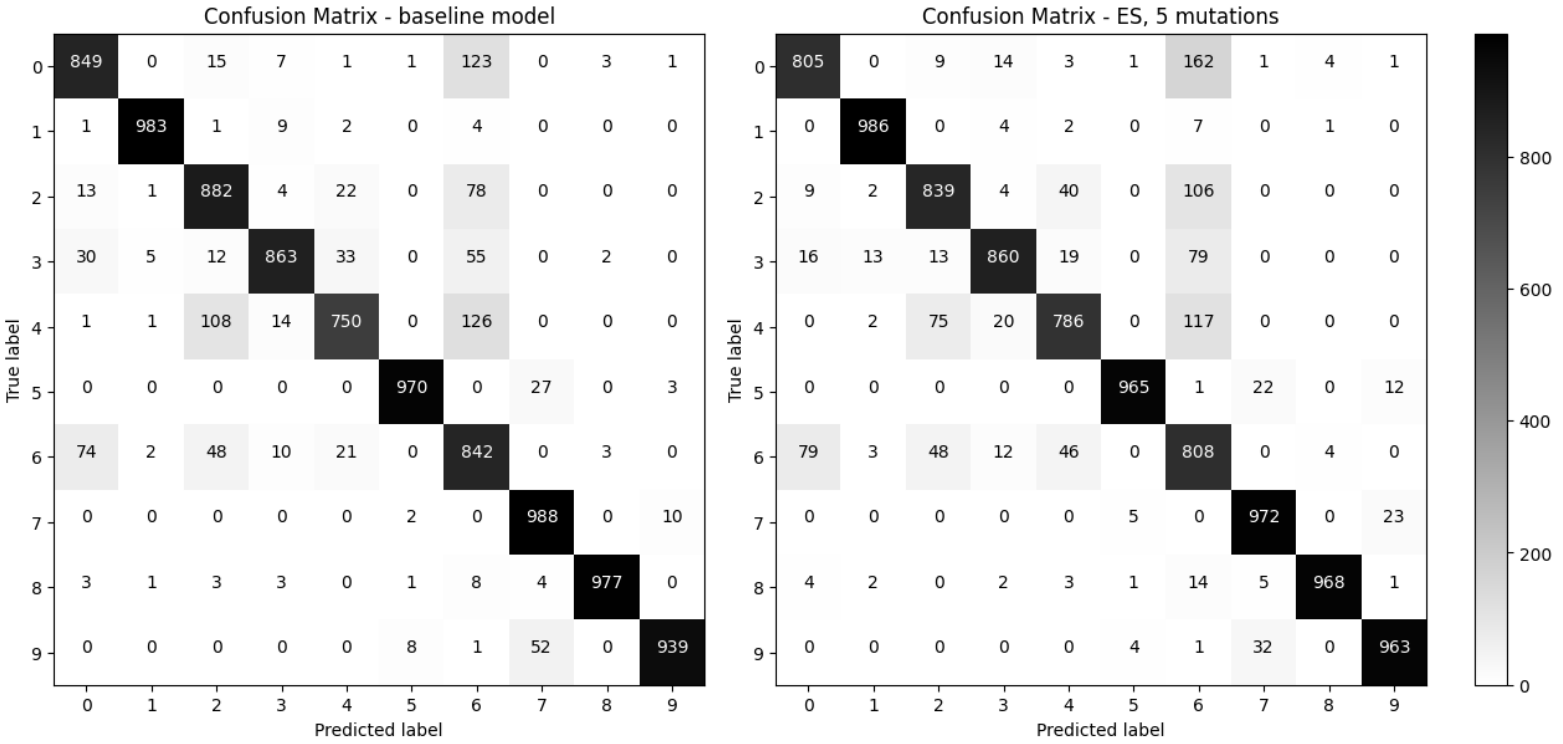


Figure 5.3.8: Confusion matrices for Fashion-MNIST, missing class

5.4 CIFAR-10 - CNN

The results obtained from the CNN model on the CIFAR-10 dataset are summarized in Table 5.4.1. All experiments utilized the identical model architecture, as outlined in Table D.3 in the Appendix. For the CIFAR-10 dataset, the accuracy levels displayed greater variability compared to the previously discussed cases. It is worth noting that this dataset is the smallest among those analyzed in this study. Therefore, the fluctuations and relatively lower accuracy values could also be influenced by the limited amount of available data.

Table 5.4.1: Accuracy of CIFAR-10 dataset

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
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 performance of the CIFAR-10 dataset using CNN is presented in more detail below.

All classes, stratified split

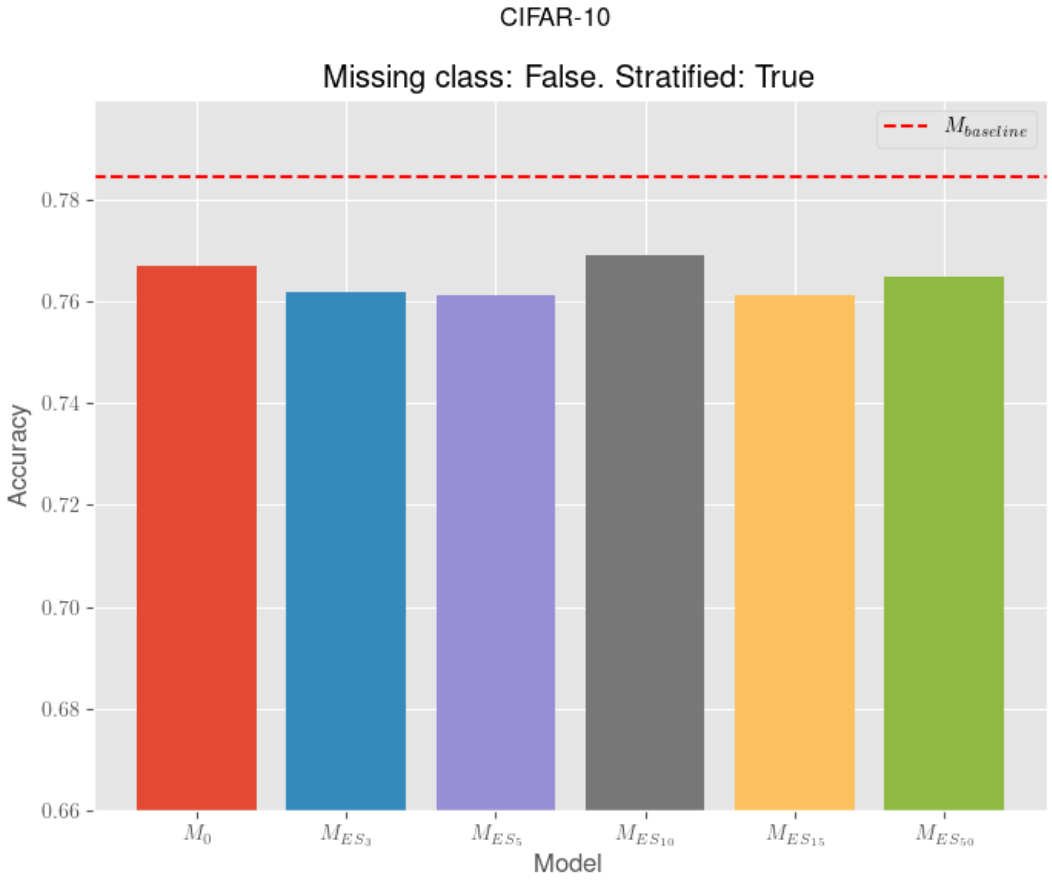


Figure 5.4.1: Accuracy of CIFAR-10 considering all classes (stratified)

Figure 5.4.1 shows the models' accuracy when all classes are taken into account and the data is split in a stratified manner. The results demonstrate that the M_0 model was surpassed by just one of the ES models. Specifically, the M_0 model achieved an accuracy of 0.7669, whereas the top-performing ES model achieved an accuracy of 0.7691. Furthermore, the baseline model exhibited an accuracy of 0.7846.

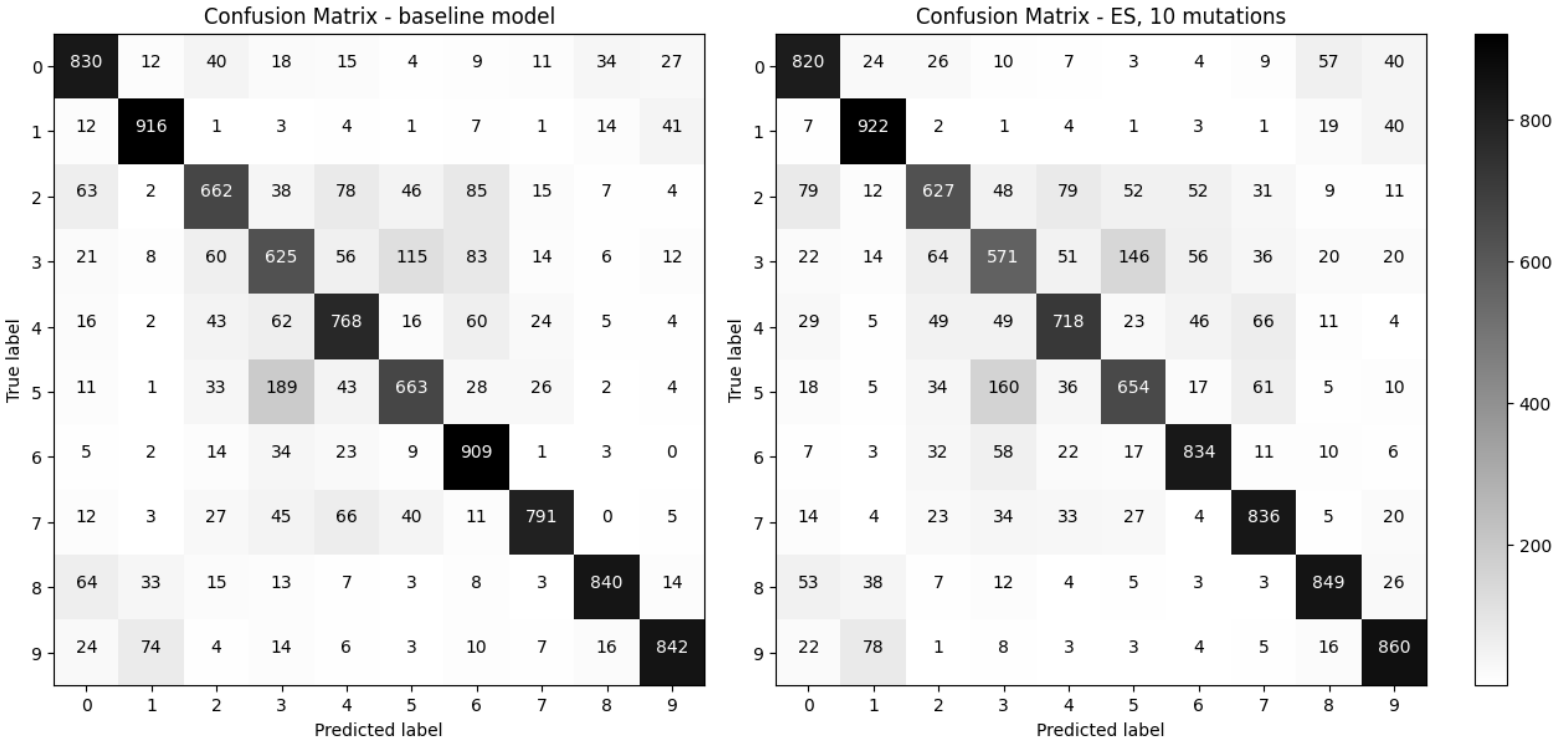


Figure 5.4.2: Confusion matrices for CIFAR-10, all classes (stratified)

The confusion matrices for the CIFAR-10 dataset are displayed in Figure 5.4.2, where it is possible to analyse the results for the baseline model and the best-performing ES model. Similar to the observations made for the Fashion-MNIST dataset, it is evident that predicting the classes for this dataset accurately was more challenging. This aligns with the information provided in Table 5.4.1, which indicates lower accuracy values. Notably, class 6 appears to be slightly more challenging to predict correctly.

All classes, non-stratified split

Figure 5.4.3: Accuracy of CIFAR-10 considering all classes

The outcomes of training a CNN on the CIFAR-10 dataset, considering all classes and employing a training data split without stratification, are presented in Figure 5.4.3. The baseline model achieved an accuracy of 0.7886, while the M_0 model attained an accuracy of 0.7718. Moreover, the accuracy of the ES models appeared to improve as the number of mutations applied increased, except for the model $M_{ES_{15}}$ which attained an accuracy of 0.7662, and was inferior to the accuracy of the model $M_{ES_{10}}$ that achieved 0.7692. Nonetheless, the model $M_{ES_{50}}$ emerged as the best-performing ES model, with an accuracy of 0.7733. In this context, it is worth emphasizing that the accuracy values for the ES models remained relatively stable throughout the training process of the CNN models. This observation suggests that the employed strategy could be a viable alternative to conventional continuous learning approaches.

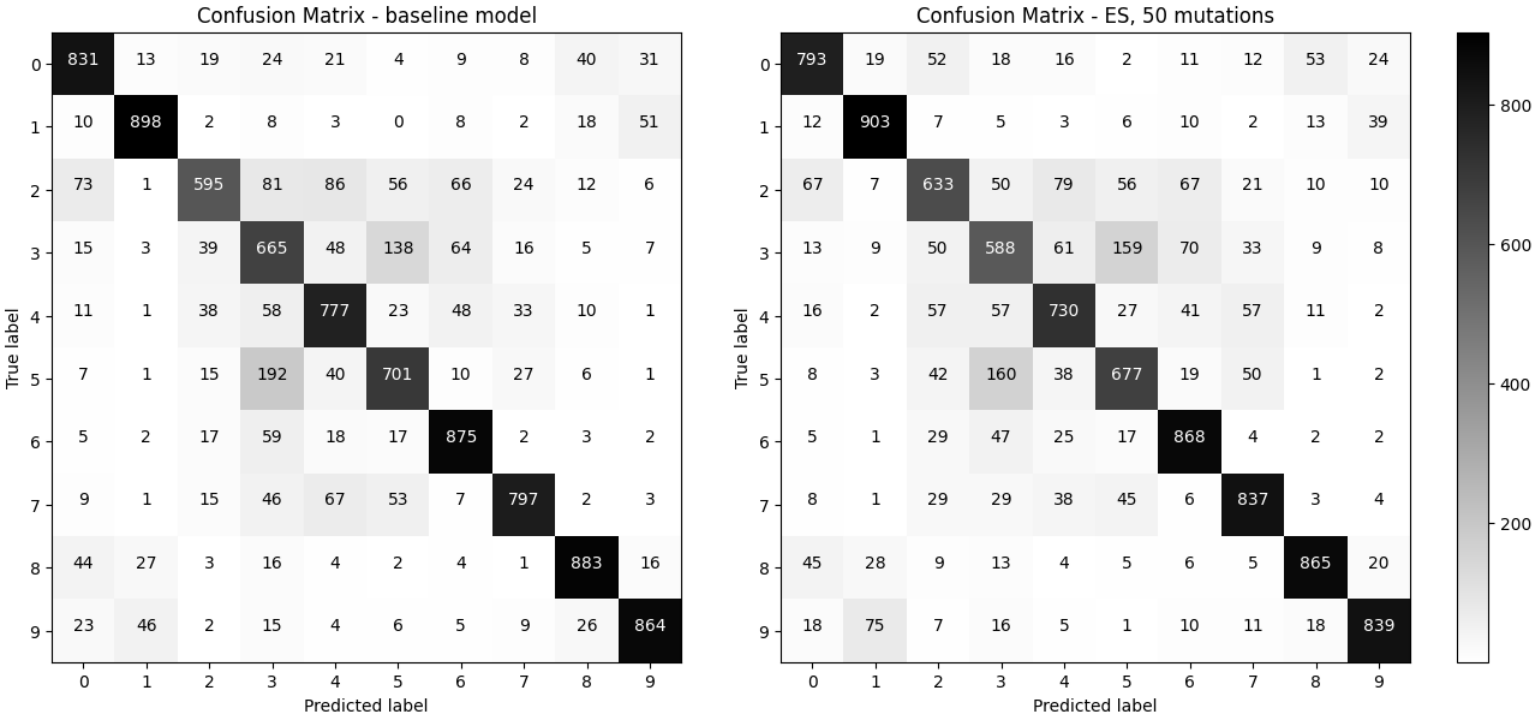


Figure 5.4.4: Confusion matrices for CIFAR-10, all classes

Figure 5.4.4 offers a comparison between the confusion matrices of the baseline model and the best-performing ES model. Despite the baseline model achieving the highest accuracy, it is evident that the ES model demonstrates a higher frequency of correct predictions for classes 1, 2, and 7 compared to the baseline model.

Missing class, stratified split

Figure 5.4.5 presents the performance of the models when the training data is stratified and a class is removed. The accuracy consistently improves with the increasing number of mutations until reaching 15 mutations ($M_{ES_{15}}$). The M_0 model achieved an accuracy of 0.6902, while the best-performing ES model surpassed it with an accuracy of 0.7381. Notably, this case exhibited the most significant discrepancy between the ES models and the baseline model, which achieved an accuracy of 0.7843. Nonetheless, the ES models demonstrate noticeable improvement compared to the M_0 model, highlighting the adaptability of the approach to new and unseen data.

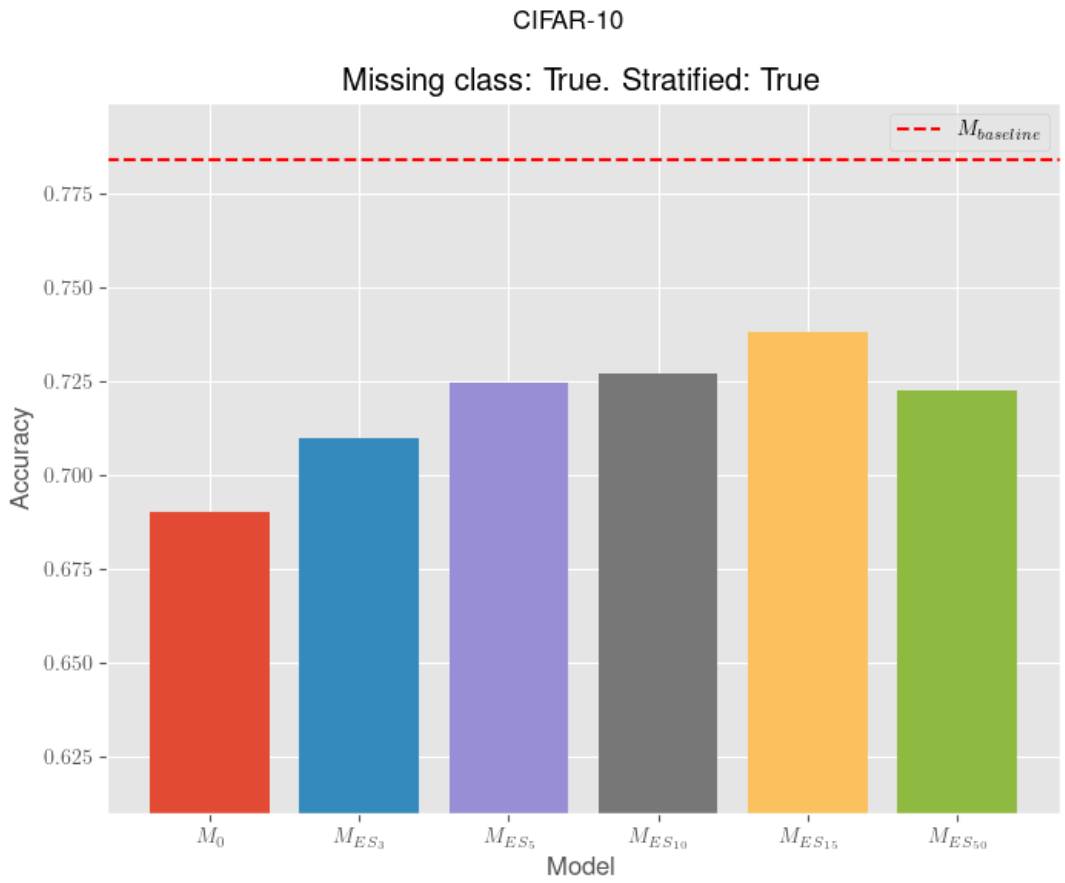


Figure 5.4.5: Accuracy of CIFAR-10, missing class (stratified)

When analyzing the confusion matrices in Figure 5.4.6, it is evident that the ES model exhibited a noticeable decrease in accuracy specifically for class 6 when compared to the baseline model. The most significant variations in misclassification patterns between the two matrices were observed for classes 1 and 6, while classes 8 and 9 displayed minimal differences.

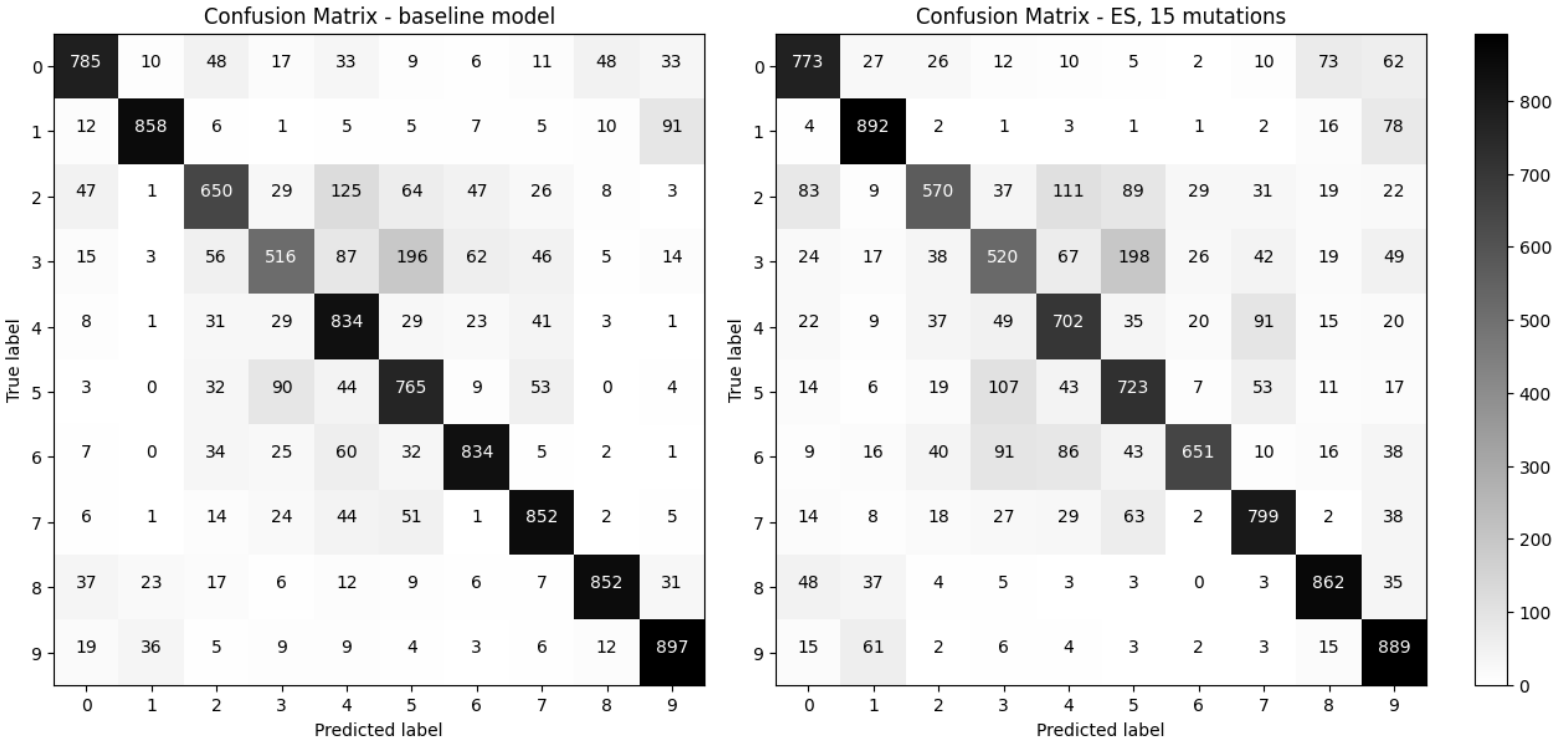


Figure 5.4.6: Confusion matrices for CIFAR-10, missing class (stratified)

Missing class, non-stratified split

Figure 5.4.7 showcases the accuracy values for the models in the scenario of a missing class and a non-stratified split. The ES models demonstrated superior performance compared to the M_0 model. The best-performing ES model achieved an accuracy of 0.7503, while the least-performing ES model attained an accuracy of 0.7317. In contrast, the accuracy of the M_0 model was 0.6896. Despite the baseline model exhibiting a higher accuracy at 0.7902, the ES models showed promising results considering the evolving nature of the data due to the addition of a new class.

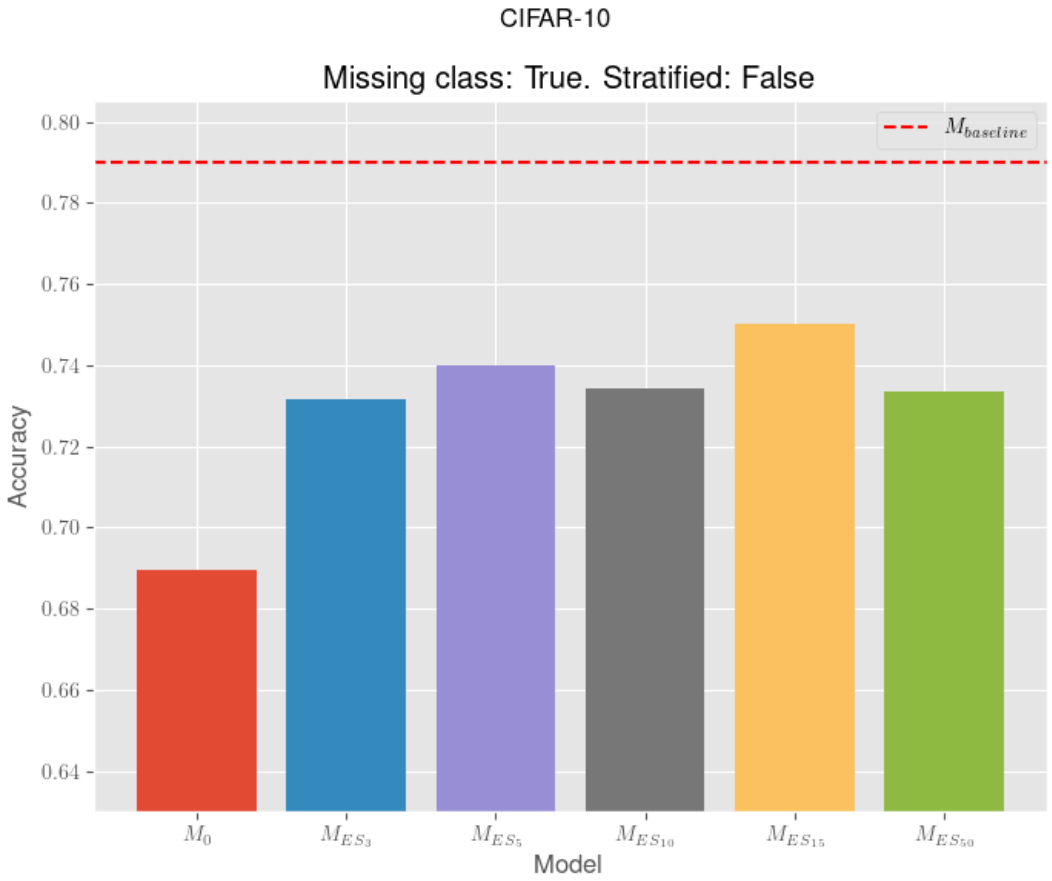


Figure 5.4.7: Accuracy of CIFAR-10, missing class

Additionally, Figure 5.4.8 presents the confusion matrices for the baseline model and the top-performing ES model. It is notable that the ES model's performance experienced a slight decline in comparison to the baseline model, particularly in predicting results for class 6. This outcome is a result of the strategy employed to simulate the scenario where unseen data becomes available over time. However, it is crucial to acknowledge that all ES models maintained a significantly stable accuracy, as mentioned earlier.

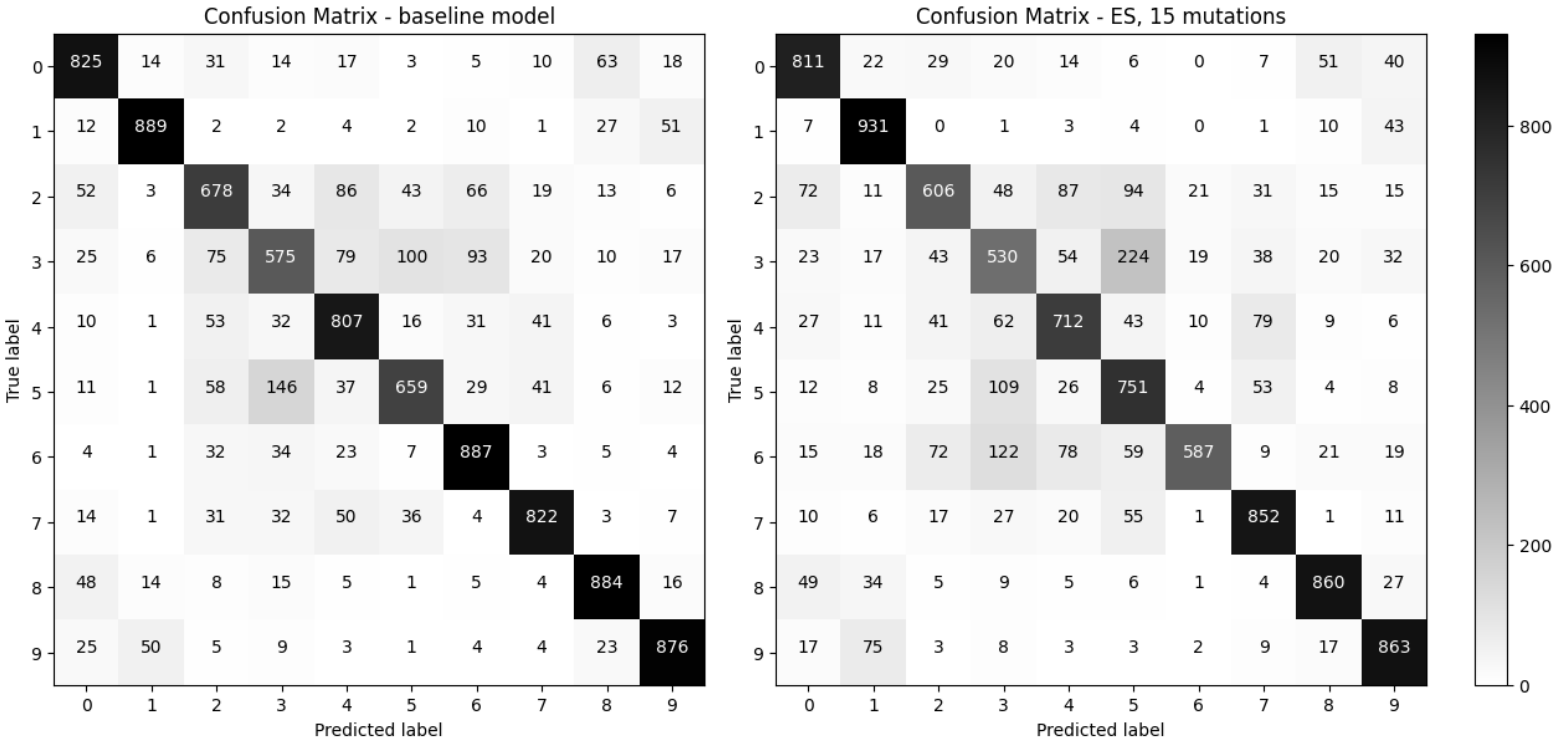


Figure 5.4.8: Confusion matrices for CIFAR-10, missing class

5.5 Summary

The results aggregated by each of the split scenarios analyzed in the thesis are presented in Figure 5.5.1.

The results reveal several important findings. Firstly, the accuracy of the models remains consistent when utilizing evolutionary strategies for continuous learning, demonstrating their ability to maintain accuracy 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.

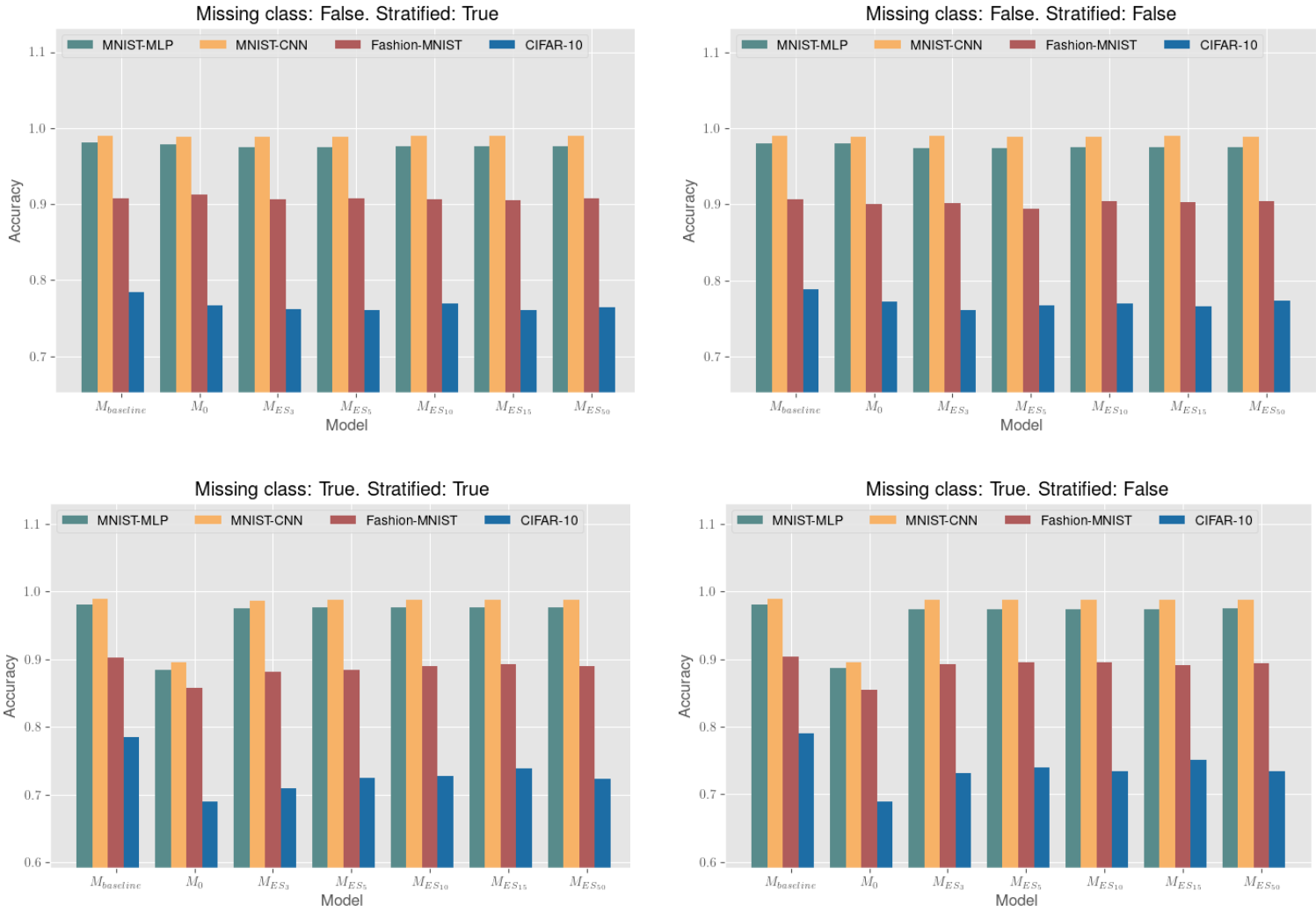


Figure 5.5.1: Summary of accuracy for the models considering the different splits and datasets

The thesis aims to illustrate that evolutionary strategies can be effectively applied in situations where training models on the entire dataset is unfeasible due to evolving data distributions or the availability of additional data. Moreover, the evolutionary strategy models outperform the baseline model in predicting specific classes, as indicated by the confusion matrices, highlighting their potential for achieving superior performance for certain classes. Overall, these results emphasize the effectiveness and practicality of evolutionary strategies for continuous learning, as they maintain accuracy, adapt to changing data, and improve predictions for specific classes.

CHAPTER

SIX

CONCLUSIONS

This thesis has presented a novel approach to address the challenges of continuous learning, a rapidly evolving field within machine learning. While traditional methods often require retraining the entire model (or making architectural changes) to prevent it from becoming obsolete over time, in this work, a new method inspired by evolutionary strategy is proposed.

The main idea behind this approach is to introduce perturbations to the weights and biases of a neural network. Rather than replacing backpropagation as usually suggested, evolutionary strategies are used as a complementary step. The initial model is trained using backpropagation on a portion of the dataset to find the optimal parameters for the task. Then, the model is replicated, and random noise is applied to its weights and biases to simulate the mutation process. Each new model is fitted with new data and its accuracy is computed.

The process is repeated for a number of iterations, and the accuracy of each mutated model is used to compute a weighted average for the new weights and biases. This information is then used to create a new model that contains the average weights and biases of all the mutated models. The proposed approach was tested on three different datasets, and the results demonstrated that the accuracy remained stable or even increased over time, without exhibiting signs of catastrophic forgetting. The continuous training models, while falling slightly behind the baseline model in terms of overall performance, demonstrate promising results, suggesting their capability and potential in handling evolving data scenarios.

Furthermore, the experiments considered different scenarios, specifically instance incremental, when M_0 and M_{0_N} were both trained on data consisting of all classes, and instance and class incremental, when M_0 was trained with a missing class and M_{0_N} was trained on all classes. Overall, the best

performance was obtained for the instance incremental scenario. Given that no class is added over time, the task complexity was relatively lower in this case. In the instance and class incremental scenario, the adopted strategy attempted to simulate the real world where new data and classes become available over time. This was done by splitting the training data into two subsets and further removing one of the classes from the larger subset and adding it to the smaller one. Consequently, this introduced class imbalance in the smaller subset.

The introduction of a new class during the training of the mutated models (M_{0_N}), showcased the ability of the proposed approach to adapt to new classes without compromising performance. In all cases, no catastrophic forgetting was observed, and in some instances, the accuracy improved as the number of mutations increased. It is important to notice that the accuracy levels achieved for the MNIST and Fashion-MNIST datasets were similar to the baseline model. In the case of the CIFAR-10 dataset, the evolutionary strategy yielded an average accuracy level approximately 5.6% lower than that achieved by the baseline model. However, it is essential to acknowledge that constructing a model trained on the entire available dataset may not be feasible or scalable in real-world scenarios. When compared to the M_0 model, which serves as a more relevant comparison in this context, the continuous training approach resulted in an average increase in accuracy of 5.4%. Additionally, it is worth noting that the optimal number of models required for achieving high accuracy may vary depending on the specific task, and hyperparameter tuning is recommended.

In conclusion, the presented approach based on evolutionary strategy holds promise for real-world applications that demand adaptability to changing data conditions. By incorporating perturbations and leveraging the average weights and biases of mutated models, the proposed method demonstrates the potential 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 pre-trained model, M_0, along with the calculation of weighted averages based on accuracy. It is recommended that alternative methods be explored for determining the offspring models derived from the trained mutated models. 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 [35]. Numerous metrics, such as the ones proposed by [36] 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] Yong Luo et al. "An appraisal of incremental learning methods". In: *Entropy* 22.11 (2020), p. 1190. ISSN: 1099-4300.

[2] Yu Liu et al. "Model Behavior Preserving for Class-Incremental Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–12. ISSN: 2162-237X. DOI: 10.1109/tnnls.2022.3144183. URL: https://dx.doi.org/10.1109/tnnls.2022.3144183.

[3] Eden Belouadah and Adrian Popescu. "IL2M: Class incremental learning with dual memory". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 583–592.

[4] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526. ISSN: 0027-8424. DOI: 10.1073/pnas.1611835114. URL: https://dx.doi.org/10.1073/pnas.1611835114.

[5] Matteo Testi et al. "MLOps: A Taxonomy and a Methodology". In: *IEEE Access* 10 (2022), pp. 63606–63618. ISSN: 2169-3536. DOI: 10.1109/access.2022.3181730. URL: https://dx.doi.org/10.1109/access.2022.3181730.

[6] A. Komolafe. *Retraining Model During Deployment: Continuous Training and Continuous Testing*. 2023. URL: https://neptune.ai/blog/retraining-model-during-deployment-continuous-training-continuous-testing (visited on 02/11/2023).

[7] Vincenzo Lomonaco and Davide Maltoni. "Core50: a new dataset and benchmark for continuous object recognition". In: *Conference on Robot Learning*. PMLR, pp. 17–26. DOI: 10.48550/arxiv.1705.03550. URL: https://dx.doi.org/10.48550/arxiv.1705.03550.

[8] A. P. McMahon. *Machine Learning Engineering with Python*. Packt, Nov. 2021.

[9] Shaza M Abd Elrahman and Ajith Abraham. "A review of class imbalance problem". In: *Journal of Network and Innovative Computing* 1.2013 (2013), pp. 332–340.

[10] Suresh Kumar Amalapuram et al. "On Handling Class Imbalance in Continual Learning based Network Intrusion Detection Systems". In: ACM. DOI: 10.1145/3486001.3486231. URL: https://dx.doi.org/10.1145/3486001.3486231.

[11] M. McCloskey and N. J. Cohen. "Catastrophic Interference in connectionist networks: The sequential learning problem". In: *The psychology of learning and motivation* 24 (1989), pp. 109–165.

[12] R. Ratcliff. "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions". In: *Psychological Review* 97 (1990), pp. 285–308.

[13] S. Grossberg. "How does a brain build a cognitive code?" In: *Psychological Review* 81 (1980), pp. 1–51.

[14] Hanul Shin et al. "Continual learning with deep generative replay". In: *Advances in neural information processing systems* 30 (2017).

[15] Anuvabh Dutt. "Continual learning for image classification". PhD thesis. Université Grenoble Alpes (ComUE), 2019.

[16] Robi Polikar et al. "Learn++: An incremental learning algorithm for supervised neural networks". In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 31 (Dec. 2001), pp. 497–508. DOI: 10.1109/5326.983933.

[17] Andrei A. Rusu et al. *Progressive Neural Networks*. 2022. arXiv: 1606.04671 [cs.LG].

[18] Zhizhong Li and Derek Hoiem. *Learning without Forgetting*. 2017. arXiv: 1606.09282 [cs.CV].

[19] Anothony Robins. "Catastrophic Forgetting, Rehearsal, and Pseudorehearsal". In: *Connection Science* 7.2 (1995), pp. 123–146.

[20] David B Fogel. "An introduction to simulated evolutionary optimization". In: *IEEE transactions on neural networks* 5.1 (1994), pp. 3–14.

[21] Daniel Câmara. "1 - Evolution and Evolutionary Algorithms". In: *Bio-inspired Networking*. Ed. by Daniel Câmara. Elsevier, 2015, pp. 1–30. ISBN: 978-1-78548-021-8. DOI: https://doi.org/10.1016/B978-1-78548-021-8.50001-6. URL: https://www.sciencedirect.com/science/article/pii/B9781785480218500016.

[22] Shifei Ding et al. "Evolutionary artificial neural networks: a review." In: *Artificial Intelligence Review* 39.3 (2013).

[23] A. E. Eiben and J. E. Smith. "Evolution Strategies". In: *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 71–87. ISBN: 978-3-662-05094-1. DOI: 10.1007/978-3-662-05094-1_4. URL: https://doi.org/10.1007/978-3-662-05094-1_4.

[24] Hans-Georg Beyer and Hans-Paul Schwefel. "Evolution strategies–a comprehensive introduction". In: *Natural computing* 1 (2002), pp. 3–52.

[25] Daan Wierstra et al. "Natural evolution strategies". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 949–980.

[26] Tim Salimans et al. *Evolution strategies as a scalable alternative to reinforcement learning*. 2017. URL: https://openai.com/research/evolution-strategies (visited on 05/23/2023).

[27] Eiji Uchibe. "Cooperative and Competitive Reinforcement and Imitation Learning for a Mixture of Heterogeneous Learning Modules". In: *Frontiers in Neurorobotics* 12 (Sept. 2018), p. 61. DOI: 10.3389/fnbot.2018.00061.

[28] Seyedali Mirjalili. "Evolutionary Feedforward Neural Networks". In: *Evolutionary Algorithms and Neural Networks: Theory and Applications*. Cham: Springer International Publishing, 2019, pp. 75–86. ISBN: 978-3-319-93025-1. DOI: 10.1007/978-3-319-93025-1_6. URL: https://doi.org/10.1007/978-3-319-93025-1_6.

[29] Tim Salimans et al. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *arXiv pre-print server* (2017). DOI: Nonearxiv:1703.03864. URL: https://arxiv.org/abs/1703.03864.

[30] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747 (2017). arXiv: 1708.07747. URL: http://arxiv.org/abs/1708.07747.

[32] Alex Krizhevsky. "Learning multiple layers of features from tiny images". In: (2009).

[33] Vijay Kotu and Bala Deshpande. "Chapter 8 - Model Evaluation". In: *Data Science (Second Edition)*. Ed. by Vijay Kotu and Bala Deshpande. Second Edition. Morgan Kaufmann, 2019, pp. 263–279. ISBN: 978-0-12-814761-0. DOI: https://doi.org/10.1016/B978-0-12-814761-0.00008-3. URL: https://www.sciencedirect.com/science/article/pii/B9780128147610000083.

[34] Margherita Grandini, Enrico Bagli, and Giorgio Visani. *Metrics for Multi-Class Classification: an Overview*. 2020. arXiv: 2008.05756 [stat.ML].

[35] David MW Powers. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation". In: *arXiv preprint arXiv:2010.16061* (2020).

[36] Natalia Dıaz-Rodrıguez et al. "Don't forget, there is more than forgetting: new metrics for Continual Learning". In: *arXiv preprint arXiv:1810.13166* (2018).

[37] E. Dral and E. Samuylova. *Machine Learning Monitoring, Part 5: Why You Should Care About Data and Concept Drift*. 2023. URL: https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift (visited on 02/11/2023).

APPENDICES

A - GITHUB REPOSITORY

All code and used in this document is included in the Github repository linked below. Further explanations are given in the readme-file.

Github repository link

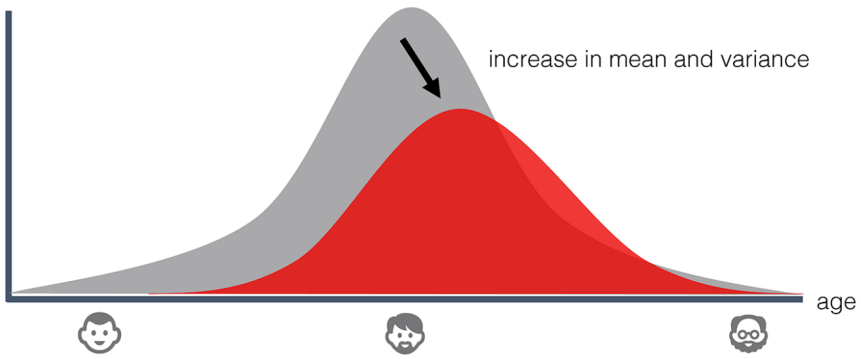
- `https://github.com/ata-bruna/thesis_es_continuous_learning`

B - MODEL DRIFT CONCEPTS

This section presents some concepts related to model drift. The performance of the model starts to decay in response to changes in data. This is called *model decay, drift* or *staleness*, and can be caused by a variety of reasons. The causes of model drift can be split into two main types, *data drift* and *concept drift* [8, 37]. Understanding these concepts is crucial as they form the basis for the development of continuous learning methods.

Data drift

Data drift occurs when there is a change in the statistical properties of the features used to develop a model. In other words, this means that the distribution of the features has been altered [8]. For example, considering that a model was trained using *age* as a feature. Initially, there is only data covering a wide part of the population, from young to older ages. However, as time passes, the distribution of the data shifts, becoming more skewed towards the right. This will impact the ability of the model to generalize well for the new data, as its distribution has changed. This is illustrated in Figure B.1.



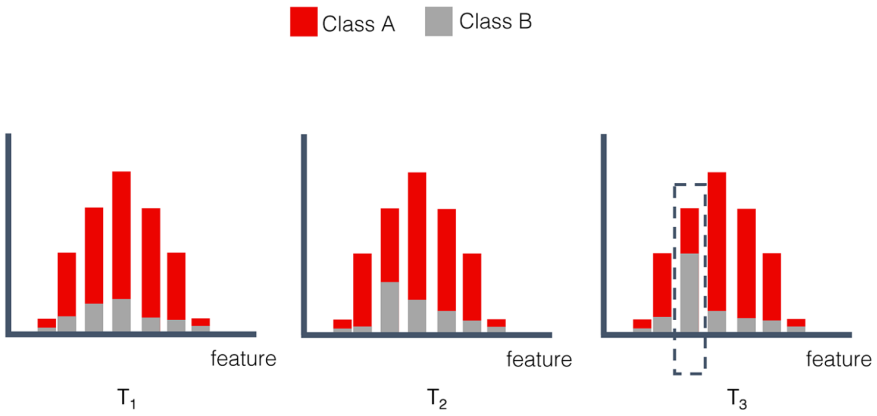
Source: `https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift`

Figure B.1: Data drift occurs when the distribution of the features changes over time. This can be seen as a change in the statistical properties, such as mean and variance

Concept drift

Concept drift happens when there is a change in the relationship between the features and outcomes [8]. In other words, concept drift occurs when the patterns the model learned previously are no longer useful to perform predictions. For example, consider a model where the relationship between input and output is linear. As time passes, more data is gathered and this relationship might change. Then it is said a concept drift has occurred. Furthermore, concept drift can be *gradual* or *drastic* [37].

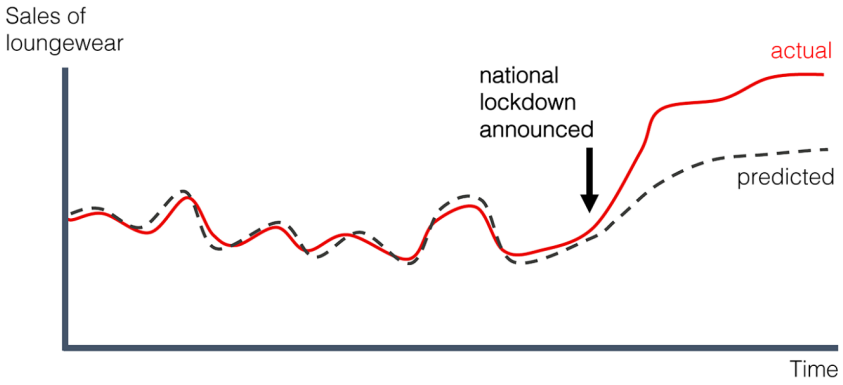
- **Gradual concept drift**: The gradual or incremental concept drift occurs when the performance of the model declines gradually as a response to subtle changes in data. These changes are not dramatic, rather they only affect a tiny segment of the data. Eventually, the changes add up impacting the overall performance of the model. The Figure B.2 illustrates this effect, for a binary classification task. Initially, the distribution of the features is stable. However, over time the share of one of the classes grows, resulting in the appearance of a new predictive pattern.



Source: https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift

Figure B.2: Gradual concept drift

- **Drastic concept drift**: The drastic or sudden concept drift occurs when a disruptive event happens. For example, when the stock market crashes or a pandemic happens. Unlike the gradual concept drift, a drastic concept drift is hard to miss, as the past data becomes irrelevant to describe the relationship between the model input and output and the models break. The Figure B.3 illustrates this phenomenon, using the forecast for sales of loungewear. During the COVID-19 pandemic, online sales of loungewear grew significantly, which was not expected given the observed past behaviour.

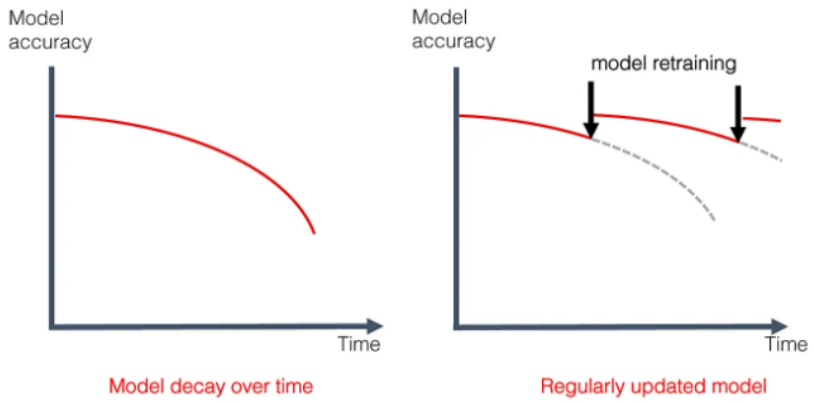


Source: https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift

Figure B.3: Drastic concept drift

The typical approach to mitigate concept drift is to retrain the model with more representative data. The strategy used can vary substantially depending on the problem. For instance, one can choose to retrain the model from scratch using all available data, before and after the change. Another approach is to assign weights to the most recent data so that the model

prioritizes the newest patterns. Additionally, incremental learning can be used to learn the model continuously. The latter will be explored later in this work. The effects of model decay and model retraining are presented in Figure B.4.



Source: https://www.evidentlyai.com/blog/machine-learning-monitoring-data-and-concept-drift

Figure B.4: Effect of concept drift and model retraining on accuracy

C - CLASS DISTRIBUTIONS IN DATASETS

A summary of class distributions in each dataset is provided in this section.

Table C.1: Class distribution in the MNIST dataset

		Class number									
Split	Subset	0	1	2	3	4	5	6	7	8	9
All classes (stratified)	Large subset	4738	5394	4766	4905	4674	4337	4734	5012	4681	4759
All classes (stratified)	Small subset	1185	1348	1192	1226	1168	1084	1184	1253	1170	1190
All classes	Large subset	4738	5394	4766	4905	4674	4337	4734	5012	4681	4759
All classes	Small subset	1185	1348	1192	1226	1168	1084	1184	1253	1170	1190
Missing class (stratified)	Large subset	4738	5394	4766	4905	4674	4337	NaN	5012	4681	4759
Missing class (stratified)	Small subset	1185	1348	1192	1226	1168	1084	5918	1253	1170	1190
Missing class	Large subset	4769	5457	4766	4887	4640	4347	NaN	5003	4617	4780
Missing class	Small subset	1154	1285	1192	1244	1202	1074	5918	1262	1234	1169

Table C.2: Class distribution in the Fashion-MNIST dataset

Split	Subset	Class number									
		0	1	2	3	4	5	6	7	8	9
All classes (stratified)	Large subset	4800	4800	4800	4800	4800	4800	4800	4800	4800	4800
All classes (stratified)	Small subset	1200	1200	1200	1200	1200	1200	1200	1200	1200	1200
All classes	Large subset	4784	4753	4761	4856	4781	4818	4855	4812	4747	4833
All classes	Small subset	1216	1247	1239	1144	1219	1182	1145	1188	1253	1167
Missing class (stratified)	Large subset	4800	4800	4800	4800	4800	4800	NaN	4800	4800	4800
Missing class (stratified)	Small subset	1200	1200	1200	1200	1200	1200	6000	1200	1200	1200
Missing class	Large subset	4784	4753	4761	4856	4781	4818	NaN	4812	4747	4833
Missing class	Small subset	1216	1247	1239	1144	1219	1182	6000	1188	1253	1167

Table C.3: Class distribution in the CIFAR-10 dataset

Split	Subset	Class number									
		0	1	2	3	4	5	6	7	8	9
All classes (stratified)	Large subset	4000	4000	4000	4000	4000	4000	4000	4000	4000	4000
All classes (stratified)	Small subset	1000	1000	1000	1000	1000	1000	1000	1000	1000	1000
All classes	Large subset	3972	3973	4034	3936	3947	4059	4004	4006	4040	4029
All classes	Small subset	1028	1027	966	1064	1053	941	996	994	960	971
Missing class (stratified)	Large subset	4000	4000	4000	4000	4000	4000	NaN	4000	4000	4000
Missing class (stratified)	Small subset	1000	1000	1000	1000	1000	1000	5000	1000	1000	1000
Missing class	Large subset	3972	3973	4034	3936	3947	4059	NaN	4006	4040	4029
Missing class	Small subset	1028	1027	966	1064	1053	941	5000	994	960	971

D - NEURAL NETWORK ARCHITECTURES

A summary of each network is described in the following sections.

MLP for MNIST model

The MLP based model initially demonstrated on the MNIST dataset is summarized in Table D.1.

Table D.1: MLP model architecture for MNIST

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
activation (Activation)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
activation_1 (Activation)	(None, 10)	0
Total params: 407,050 Trainable params: 407,050 Non-trainable params: 0		

CNN for MNIST and Fashion-MNIST models

The CNN based model demonstrated on the MNIST and Fashion-MNIST data set is summarized in Table D.2.

Table D.2: CNN model architecture for MNIST and Fashion-MNIST

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_1 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 128)	401536
dense_1 (Dense)	(None, 10)	1290

Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0

CNN for CIFAR-10 model

The CNN based model demonstrated on the CIFAR-10 data set is summarized in Table D.3.

Table D.3: CNN model architecture for CIFAR-10

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
conv2d_1 (Conv2D)	(None, 30, 30, 32)	9248
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 15, 15, 64)	18496
conv2d_3 (Conv2D)	(None, 13, 13, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 512)	1180160
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
Total params: 1,250,858		
Trainable params: 1,250,858		
Non-trainable params: 0		

E - LOSS VALUES

When training a machine learning model, an increase in loss values typically indicates that the model's predictions are becoming less accurate or less aligned with the true values of the target variable. There could be several reasons for an increase in loss values during training, such as underfitting, overfitting or poor hyperparameter settings. It is important to monitor the loss values during training to identify potential issues. If loss values consistently increase or fail to decrease over time, it may be necessary to adjust the model architecture, hyperparameters, or data preprocessing steps to improve the model's performance. In the following sections, losses from the experiments are presented. The lowest loss is highlighted in bold.

E1 - MNIST - MLP

Table E.1: Loss values for MNIST dataset (MLP)

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
All classes (stratified)	0.0629	0.0746	0.0829	0.0822	**0.0820**	0.0822	0.0822
All classes	0.0650	0.0663	0.0821	0.0824	**0.0812**	0.0817	0.0816
Missing class (stratified)	0.0659	1.8690	0.0793	0.0813	0.0792	0.0791	**0.0788**
Missing class	0.0693	1.5200	0.0863	0.0852	0.0845	0.0842	**0.0840**

All classes, stratified split

Figure E.1: Loss for MNIST - MLP considering all classes (stratified)

All classes, non-stratified split

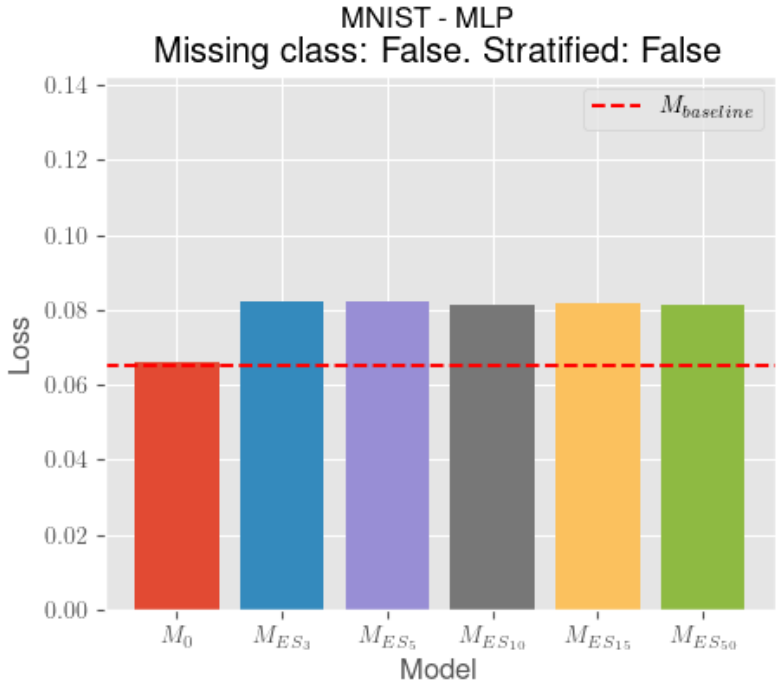


Figure E.2: Loss for MNIST - MLP considering all classes

Missing class, stratified split

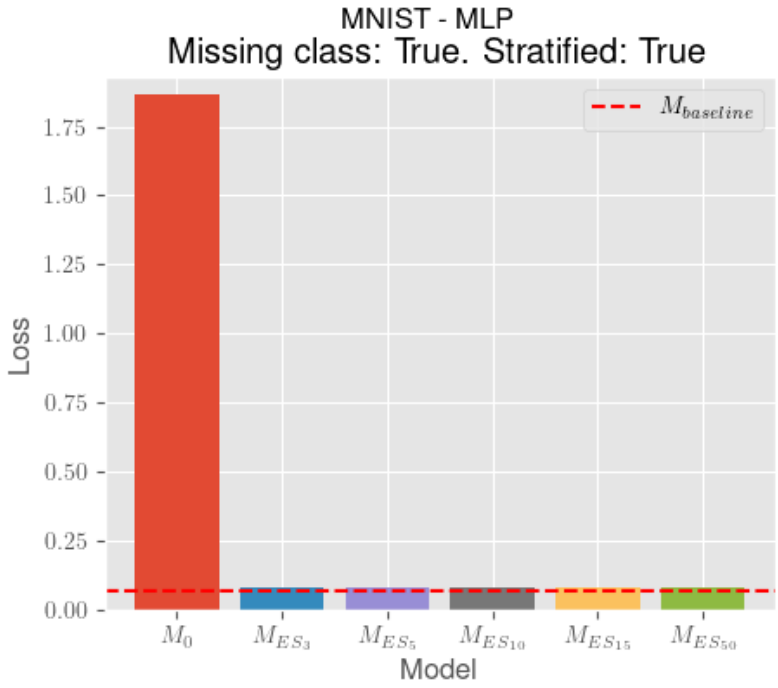


Figure E.3: Loss for MNIST - MLP missing class (stratified)

Missing class, non-stratified split

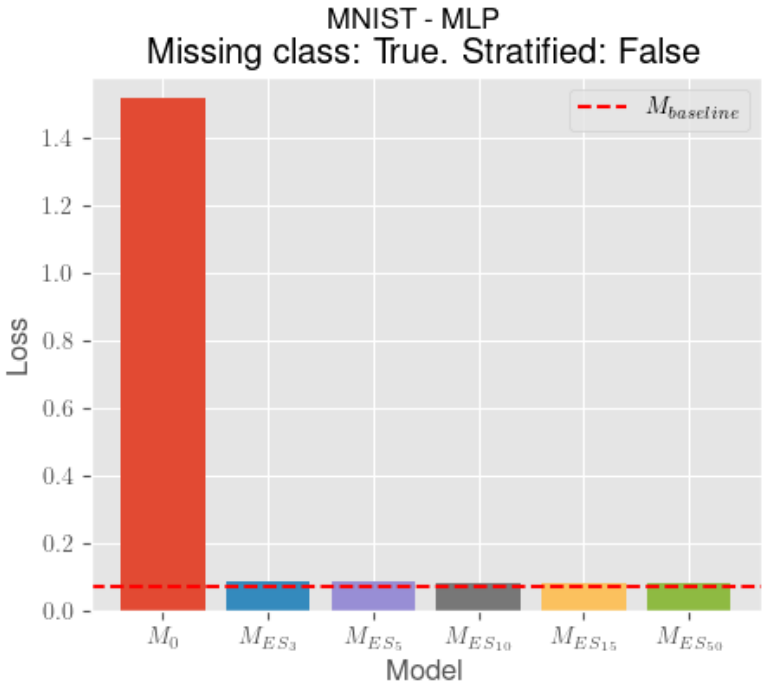


Figure E.4: Loss for MNIST - MLP missing class

E2 - MNIST - CNN

Table E.2: Loss values for MNIST dataset (CNN)

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
All classes (stratified)	0.0395	0.0369	0.0339	0.0350	**0.0314**	0.0328	0.0318
All classes	0.0299	0.0323	0.0331	0.0338	0.0323	0.0327	**0.0322**
Missing class (stratified)	0.0331	1.9315	0.0411	0.0382	0.0376	0.0373	**0.0367**
Missing class	0.0356	2.0838	0.0353	0.0361	0.0349	**0.0343**	0.0354

All classes, stratified split

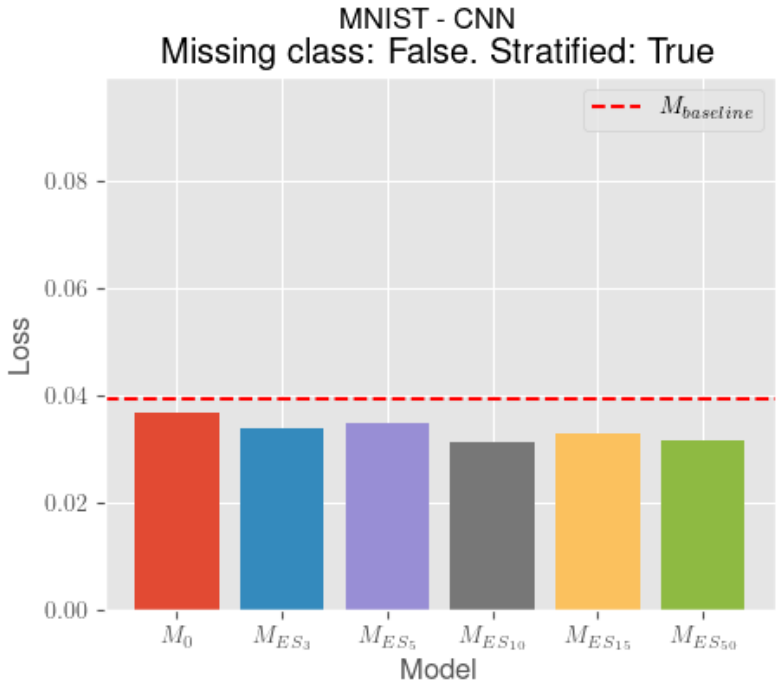


Figure E.5: Loss for MNIST - CNN considering all classes (stratified)

All classes, non-stratified split

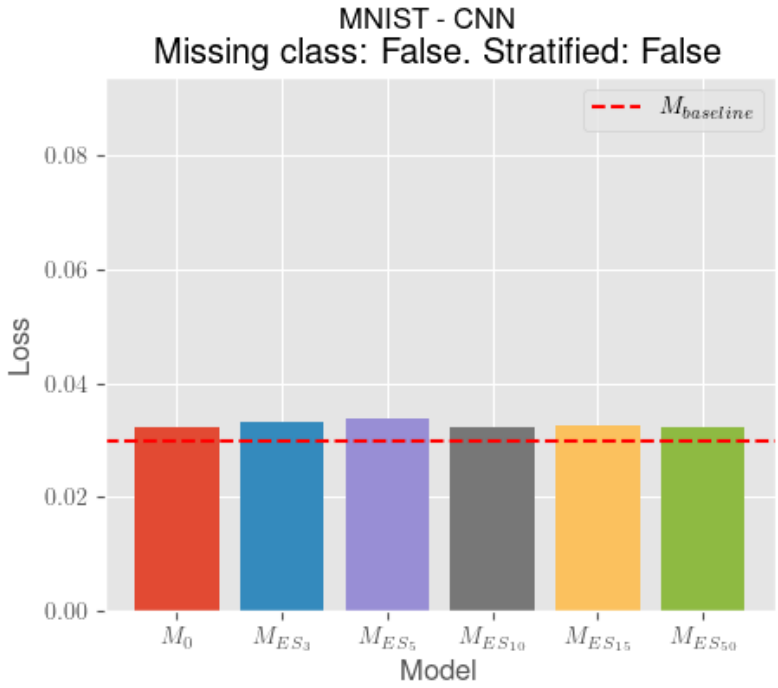


Figure E.6: Loss for MNIST - CNN considering all classes

Missing class, stratified split

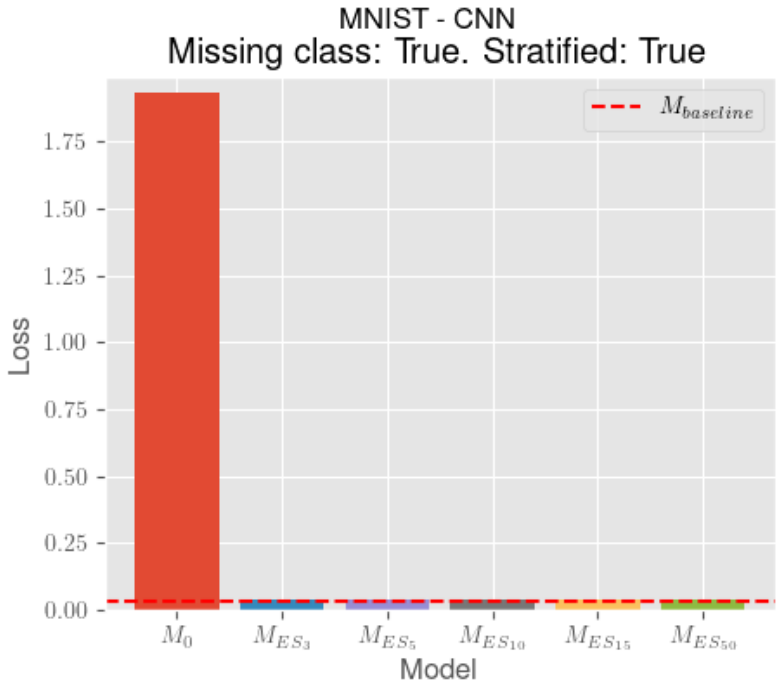


Figure E.7: Loss for MNIST - CNN missing class (stratified)

Missing class, non-stratified split

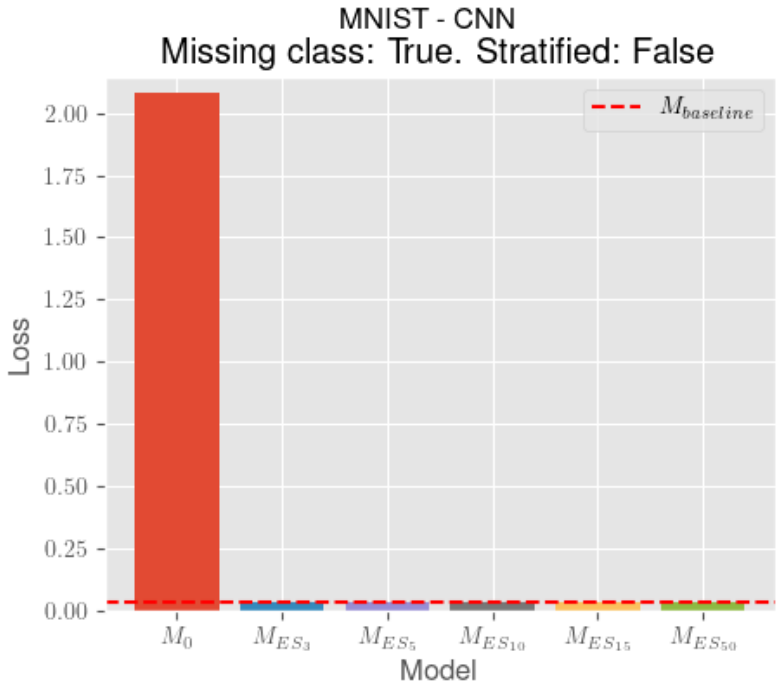


Figure E.8: Loss for MNIST - CNN missing class

E3 - Fashion-MNIST - CNN

Table E.3: Loss values for Fashion-MNIST dataset

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
All classes (stratified)	0.2653	0.2798	**0.2622**	0.2652	0.2661	0.2690	0.2628
All classes	0.2660	0.2805	0.2991	0.3241	**0.2852**	0.2969	0.2911
Missing class (stratified)	0.2911	1.8818	0.3200	0.3112	0.2979	**0.2941**	0.2962
Missing class	0.2774	1.9074	0.3020	**0.2876**	0.2925	0.2966	0.2890

All classes, stratified split

Figure E.9: Loss for Fashion-MNIST considering all classes (stratified)

All classes, non-stratified split

Figure E.10: Loss for Fashion-MNIST considering all classes

Missing class, stratified split

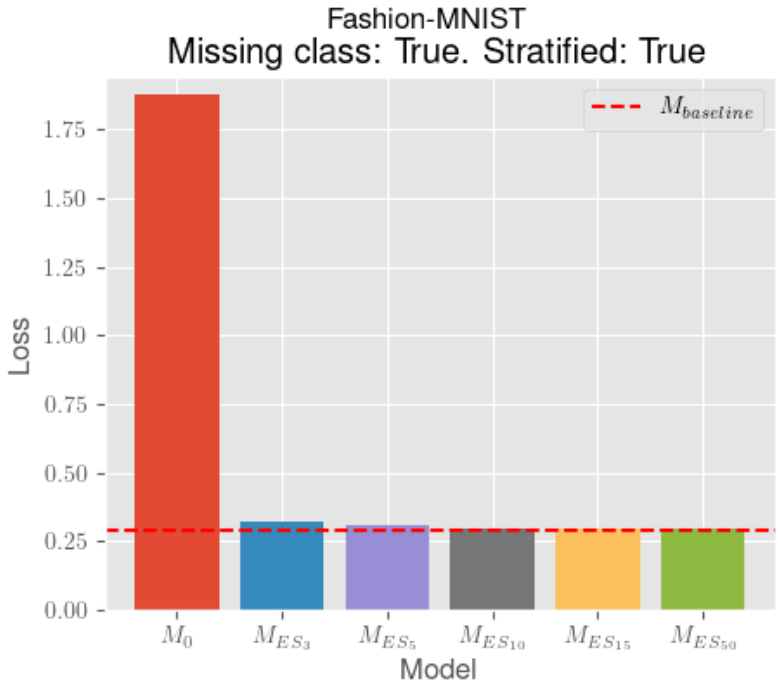


Figure E.11: Loss for Fashion-MNIST missing class (stratified)

Missing class, non-stratified split

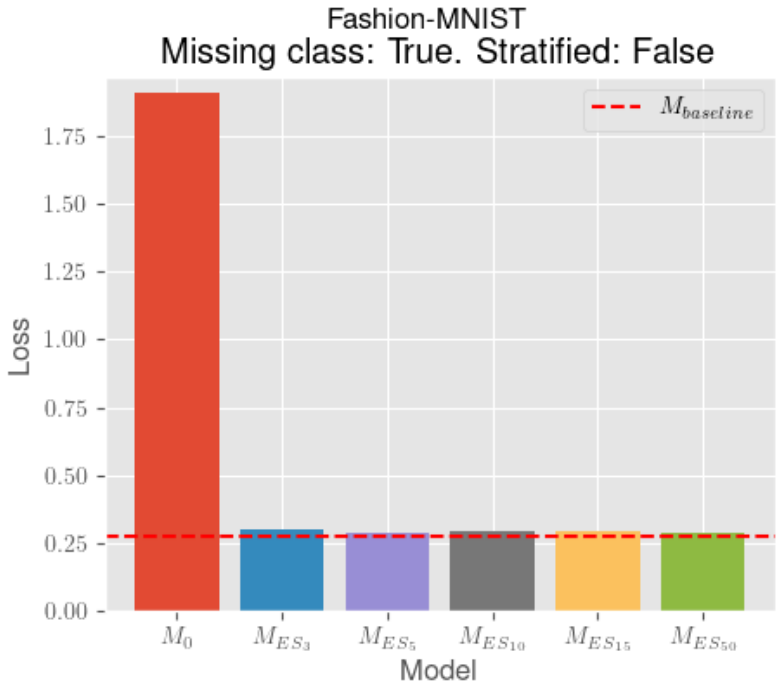


Figure E.12: Loss for Fashion-MNIST missing class

E4 - CIFAR-10 - CNN

Table E.4: Loss values for CIFAR-10 dataset

			ES models				
Model	$M_{baseline}$	M_0	M_{ES_3}	M_{ES_5}	$M_{ES_{10}}$	$M_{ES_{15}}$	$M_{ES_{50}}$
All classes (stratified)	0.6423	0.7122	0.7241	0.7066	**0.6795**	0.7079	0.6939
All classes	0.6330	0.6974	0.6898	0.6816	0.6706	0.6804	**0.6559**
Missing class (stratified)	0.6585	1.9993	0.8638	0.8055	0.8138	**0.7702**	0.8219
Missing class	0.6253	2.4061	0.8079	0.7578	0.7721	**0.7370**	0.7761

All classes, stratified split

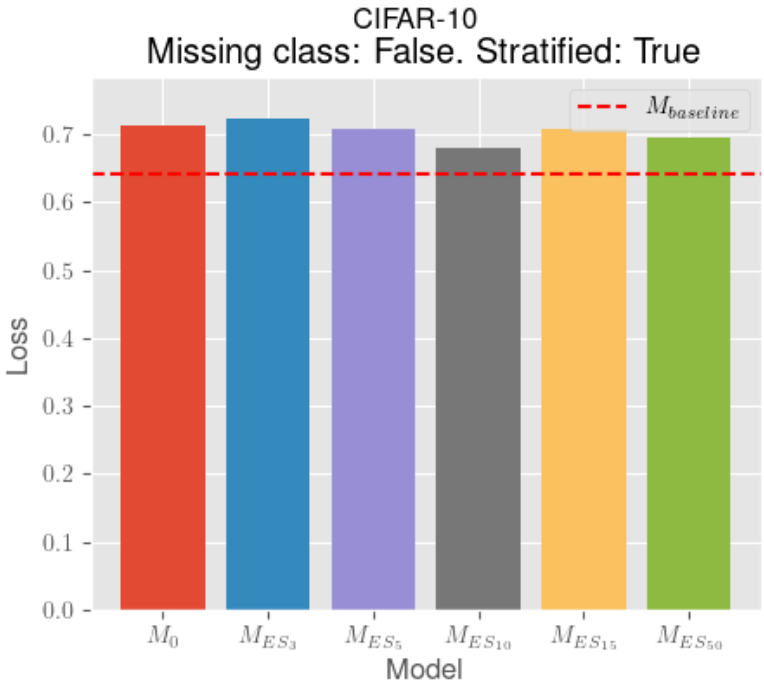


Figure E.13: Loss for CIFAR-10 considering all classes (stratified)

All classes, non-stratified split

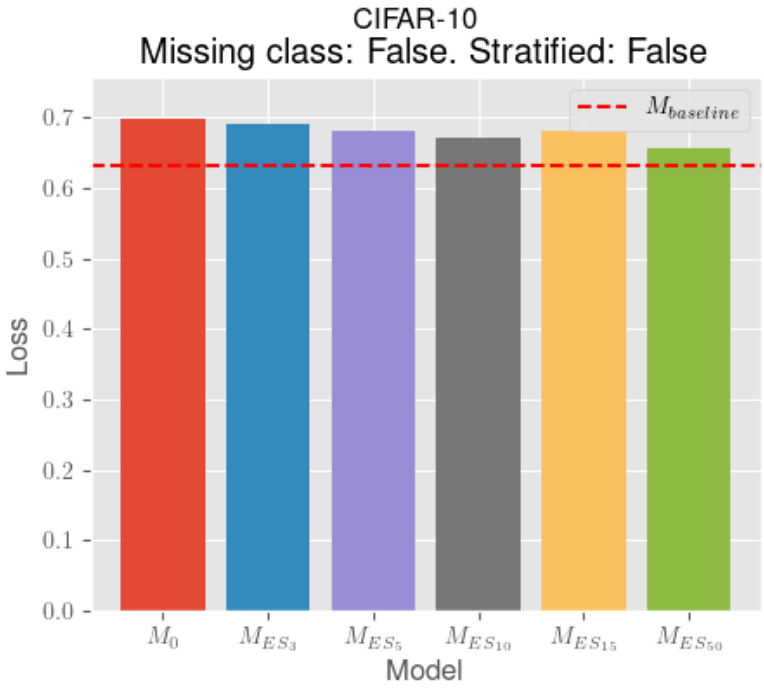


Figure E.14: Loss for CIFAR-10 considering all classes

Missing class, stratified split

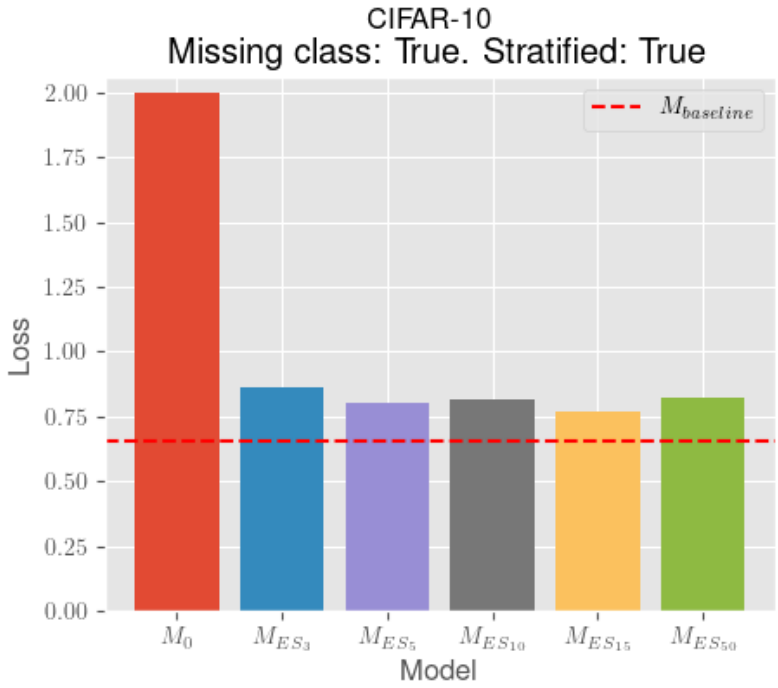


Figure E.15: Loss for CIFAR-10 missing class (stratified)

Missing class, non-stratified split

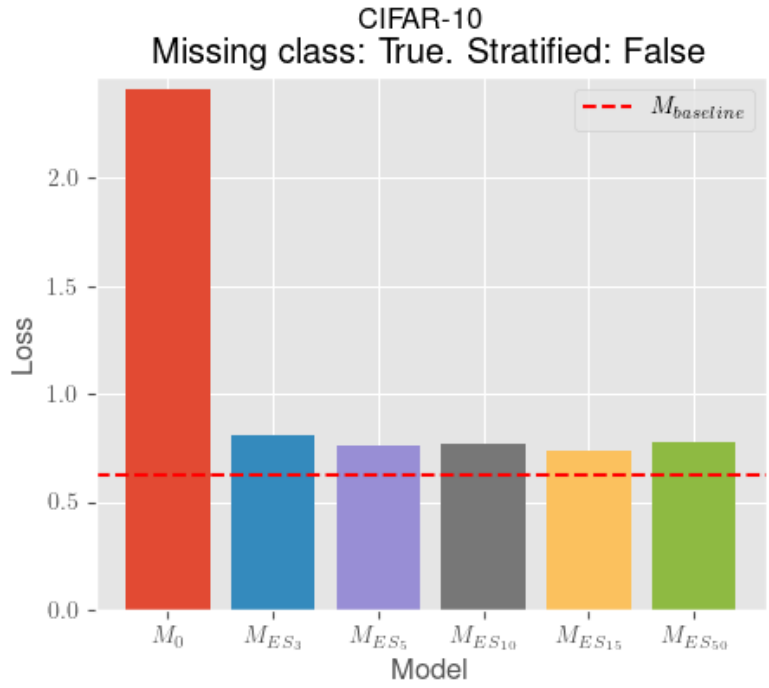


Figure E.16: Loss for CIFAR-10 missing class

Summary

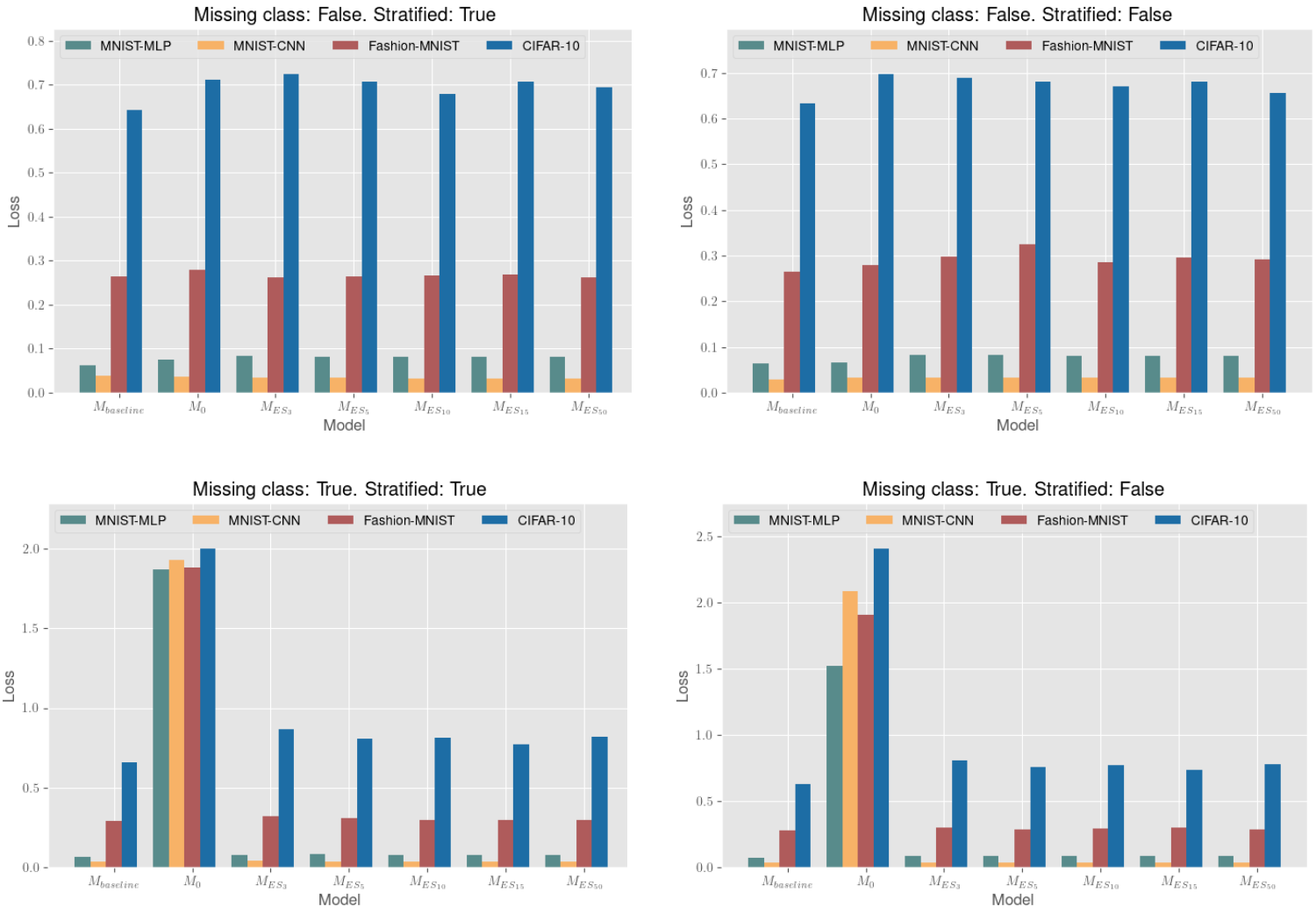


Figure E.17: Summary of losses for the models considering the different splits and datasets

The thesis findings reveal significant decreases in the model loss for scenarios where a class was missing, as depicted in Figure E.17. Throughout the continuous learning process, the models consistently experience a decline in loss when evolutionary strategies are employed in these situations. Comparatively, the M_0 model exhibits higher loss values due to the absence of a class during training. However, upon analyzing the training context, it becomes evident that the loss decreases with each epoch. For scenarios where all classes are considered, the loss remains relatively stable without significant variation. Notably, the loss for these cases starts at a lower level from the beginning of training. Importantly, the absence of any loss explosion indicates that there are no signs of overfitting in the models analyzed.

F - POSTER PRESENTATION

The poster is presented in the next page.

Evolving Deep Neural Networks for Continuous Learning: Addressing Challenges and Adapting to Changing Data Conditions without Catastrophic Forgetting

Bruna Atamanczuk, Kurt Arve Skipenes Karadas

University of Stavanger, Faculty of Science and Technology, Department of Electrical Engineering and Computer Science

Abstract

This thesis 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 16 scenarios investigated without catastrophic forgetting. The experiments were conducted on three benchmark datasets, MNIST, Fashion-MNIST and CIFAR-10. The data was split considering stratified and non-stratified sampling and with and without missing classes. The approach adapts to new classes 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.

Introduction

Continuous Learning (CL) is the practice of refining and improving machine learning models throughout their entire life cycle. In a traditional machine learning life cycle, a model is built with training data, deployed into production, and periodically improved over time. However, continuous learning takes this process a step further by acknowledging that models can continue to learn and improve even after deployment. This is achieved through ongoing refinement, retraining, and adaptation of the model to new data and changing environments. Continuous learning ensures that machine learning models remain relevant, effective, and up-to-date, providing businesses with a competitive edge in today's fast-paced technological landscape.

Traditionally, the approach used to update a model once more data is available was to retrain it from scratch. This can result in models becoming computationally expensive and time-consuming [1]. CL offers a more efficient solution by allowing a model to learn new classes incrementally, without the need to discard previous knowledge. Furthermore, CL also addresses problems such as **class imbalance** and **catastrophic forgetting**. The former is related to the difficulty of learning minority classes due to a lack of labelled data [2], and the latter is related to the tendency of deep neural networks to forget previously learnt tasks once new information is incorporated [3]. CL allows the model to learn new classes over time, and to retain the previous knowledge, which can help improve the overall performance of the model.

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 that introduces random perturbations to the current solutions, in order to explore the search space and potentially discover better solutions.

Methodology

The approach consists of introducing perturbations to the weights and biases of a neural network. Rather than replacing backpropagation as usually suggested, evolutionary strategies are used as a complementary step. The initial model is trained using backpropagation on a portion of the dataset to find the optimal parameters for the task. Then, the model is replicated, and random noise is applied to its weights and biases to simulate the mutation process. Each new model is fitted with new data and its accuracy is computed.

The process is repeated for several iterations, and the accuracy of each mutated model is used to compute a weighted average for the new weights and biases. This information is then used to create a new model that contains the average weights and biases of all the mutated models.

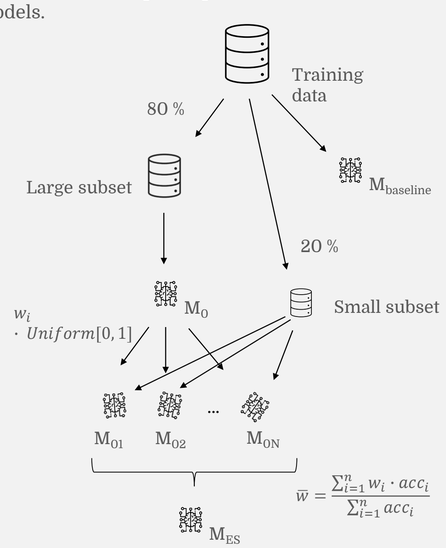


Figure 1: Continuous training procedure

The training data for each dataset was split into two subsets using four different strategies. The training data was split with and without stratification. Also, for half of the experiments, one randomly chosen single class was removed. When the class was removed, it was removed from the larger subset of training data and appended to the smaller one. The different strategies are shown in Table 1.

The idea behind removing one class, was to evaluate the evolutionary strategy when a new class becomes available, simulating a real-world application.

Table 1: Adopted training strategies

Model	Missing class	Stratified
All classes (stratified)	False	True
All classes	False	False
Missing class (stratified)	True	True
Missing class	True	False

Results

Totally, 16 experiments were conducted in our work for the three datasets mentioned. Below, results for a **Missing class (stratified)** model trained on Fashion-MNIST dataset is presented. In our experiments, **class number 6** was removed from the larger subset. Separate test data was used to evaluate the accuracy levels of the models.

The dashed red line shows the accuracy of the baseline model trained on the full training data. To make the findings easier to read, note that the y-axis has been trimmed. One can see that the accuracy increases with the number of mutations up until the final offspring model and outperform M_0 showcasing that the model learns new data while retaining old knowledge.

Figure 2: Accuracy of Fashion-MNIST. Missing class (stratified)

Confusion matrices were also produces to compare the performance of the baseline model and the best-performing ES model. As one can see below, the model obtained from evolutionary strategy performs well. At first glance, it may seem like the model struggles adopting to the new class, which it somewhat does compared to the baseline model. However, the reader should be aware that even the baseline model have some issues with correctly classifying this class.

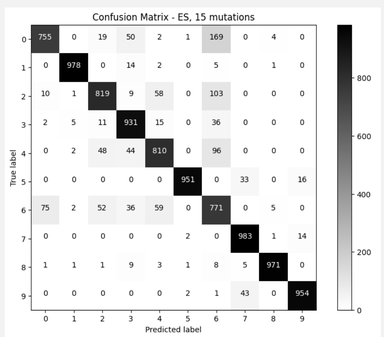


Figure 3: Confusion Matrix for Fashion-MNIST. Missing class (stratified)

In a real-world scenario, it may be not scalable to build a model that is trained on all available data such as the baseline model. For the purpose of this work, the achieved performance of the new approach provides valuable insights of how well the proposed methodology works.

Conclusion

This thesis has presented a novel approach to address the challenges of continuous learning. While traditional methods often require retraining the entire model (or making architectural changes) to prevent it from becoming obsolete over time, in this work, a new method inspired by evolutionary strategy is proposed. The main idea behind this approach is to introduce perturbations to the weights and biases of a neural network. The proposed approach was tested on three different datasets, and the results demonstrated that the accuracy remained stable or even increased over time, without exhibiting signs of catastrophic forgetting. The continuous training models, while falling slightly behind the baseline model in terms of overall performance, demonstrate promising results, suggesting their capability and potential in handling evolving data scenarios.

Acknowledgements

We would like to express our heartfelt gratitude to our supervisors, Antorweep Chakravorty at UiS and co-supervisor Bikash Agrawal at Simplifai.ai, for their invaluable guidance, support, and mentorship throughout the course of this research. Their expertise, dedication, and commitment to excellence have been instrumental in shaping the direction and success of this thesis.

References

[1] Yu Liu et al. "Model Behavior Preserving for Class-Incremental Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–12. ISSN: 2162-237X. DOI: 10.1109/tnnls.2022.3144183. url: https://dx.doi.org/10.1109/tnnls.2022.3144183.

[2] Eden Belouadah and Adrian Popescu. "IL2M: Class incremental learning with dual memory". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 583–592.

[3] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences 114.13* (2017), pp. 3521–3526. issn: 0027-8424. DOI: 10.1073/pnas.1611835114. URL: https://dx.doi.org/10.1073/pnas.1611835114.