IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 21 October 2024, accepted 16 November 2024, date of publication 2 December 2024, date of current version 16 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3510536

RESEARCH ARTICLE

Continual Learning for Object Classification: Integrating AutoML for Binary Classification Tasks Within a Modular Dynamic Architecture

DANIEL TURNER[1], PEDRO J. S. CARDOSO[2], AND JOÃO M. F. RODRIGUES[2]
[1]NOVA LINCS, FCT, ISE, Universidade do Algarve, 8005-139 Faro, Portugal
[2]NOVA LINCS, ISE, Universidade do Algarve, 8005-139 Faro, Portugal

Corresponding author: Daniel Turner (a49125@ualg.pt)

This work was supported by NOVA LINCS funded by Fundação para a Ciência e Tecnologia (FCT.IP) Grant UIDB/04516/2020 (https://doi.org/10.54499/UIDB/04516/2020) and Grant UIDP/04516/2020 (https://doi.org/10.54499/UIDP/04516/2020).

ABSTRACT For humans it is quite easy to identify a new object after learning to identify existing ones, but not for a machine. Deep neural networks (DNN) are the foundation of the current state-of-the-art methods for training machines to recognize sets of objects. The issue is that any modification to the DNN weights that were trained to classify an initial set of objects has the potential to seriously impair the network's ability to make those initial classifications; this behaviour is referred to as catastrophic forgetting (CF). This paper presents a continual learning (CL) architecture that can deal with CF. The architecture is composed of two primary parts: (i) The feature extraction component, which is based on the ResNet50 backbone and (ii) the modular dynamic classification (MDC) component. The latter is made up of multiple sub-networks that gradually assemble themselves into a tree-like structure that reorganizes itself as it learns over time, so that each sub-network can operate independently. The MDC relies heavily on binary classification, and here the application of automated machine learning (AutoML) was introduced, where each binary classifier is tailored on-the-fly, and is/can be different from object to object. The strategy involves a calculated selection from a predefined list of model types and parameters, optimizing them for their respective tasks. Results demonstrate that we advanced the adaptability and performance of the network, emphasizing the transformative potential of AutoML in modular CL approaches. Tests on the CORe50 dataset showed accuracy results of 81.1%, which are above the state of the art for CL architectures.

INDEX TERMS AutoML, binary classification, catastrophic forgetting, computer vision, continual learning, neural networks, object recognition.

I. INTRODUCTION

Continual learning (CL), particularly in the realm of object classification, has emerged as a vital area of research in the field of artificial intelligence (AI). Despite significant advancements, the community continues to struggle with the challenge of developing models that can adapt to new data on-the-fly while retaining previously acquired knowledge. This difficulty is largely due to the fact that standard neural networks (NN) have a tendency to "forget" their knowledge of previous tasks when trained on new data, a phenomenon known as catastrophic forgetting (CF), which poses a substantial obstacle in the pursuit of AI systems capable of mimicking a human-like learning processes [1].

This paper builds upon the foundation laid in our previous work on a "Modular Dynamic Neural Network" (MDNN) for object classification [2], a novel approach designed to address the issues of adaptability and scalability in CL. The MDNN consists of two main components: (a) the ResNet50-based feature extraction component as the backbone and (b) the modular dynamic classification (MDC) component. The latter consists of multiple sub-networks

The associate editor coordinating the review of this manuscript and approving it for publication was Antonio J. R. Neves.

and progressively builds itself up in a tree-like structure that rearranges itself as it learns over time, in such a way that each sub-network can function independently. The architecture, while robust, employed a static approach to binary classification tasks within its modular framework (complementary information can be seen in Section III).

In this study, we pivot from this static model to a dynamic and informed selection process through the integration of automated machine learning (AutoML) [3]. The primary objective is to harness the potential of AutoML to enhance the adaptability and performance of binary classification tasks within the MDC component. Our approach involves the use of a diverse range of classifiers, including support vector machines (SVMs), logistic regression (LR), and neural networks [2], [4], each specifically tailored to address distinct binary classification challenges within the MDNN. The strategy entails a systematic selection from a predefined set of models and parameters, which are then trained and optimized for their respective tasks.

This new method marks a significant departure from the original practice of employing a fixed model structure, promising improvements not only in the performance of individual modules but also in the adaptability of the overall architecture, creating the MDNN2.0.

To rigorously evaluate the effectiveness of the approach, the MDNN2.0 will be trained multiple times from scratch on the CORe50 dataset [5]. Each training session will utilize different model types and parameters available, with an emphasis on monitoring the network's learning progression and its ability to mitigate CF.

The uniqueness/the main contributions of this study lie in: (i) a new architecture for the MDC, that achieves better results than other CL methods, and (ii) its methodology, where for each binary classifier that is trained, a variety of model types and parameters are explored. The network's F1 score[1] is subsequently calculated for each variant of each classifier, and the one yielding the best performance is selected for integration into the MDNN2.0 (see Section III for more details).

Following this Introduction section, the contextualization, trends, and developments in continual learning are presented in Section II and the MDNN2.0 method is presented in Section III, including the detailed integration of AutoML in to MDNN. Section IV will lay the groundwork for the tests and results, dedicating itself to a set of tests and results from sub-models/classifiers (namely, SVMs, LRs, and NNs), followed by a combined approach using all of these sub-model types. Finally, in Section V, the conclusions and the future work are presented, including the analysis of the results and their implications, which will be discussed highlighting the impact of AutoML in modular CL systems.

[1]The F1-score is the harmonic mean of precision and recall, providing a balanced measure of clustering quality when ground truth labels are available. It is computed as $F1 = 2 \times (P \times R)/(P + R)$, where precision (P) is defined as $P = TP/(TP + FP)$ and recall (R) as $R = TP/(TP + FN)$. In this context, TP represents true positives, FP is false positives, and FN is false negatives. This metric is particularly valuable for imbalanced cluster distributions as it equally weights both false positives and false negatives. For further details, see [6].

II. CONTEXTUALIZATION, TRENDS AND DEVELOPMENTS IN CONTINUAL LEARNING

In this section, the AutoML concept is addressed and discussed, as well as a dataset that can best test CL models. The section ends with the presentation and small discussion of the trends and developments in continual learning methods.

Neural networks may gradually adapt to new tasks, data distributions, or surroundings thanks to ongoing learning, in contrast to classic machine learning (ML) paradigms where models are trained on fixed datasets. Catastrophic Forgetting is a phenomenon where neural networks lose performance on previously learned tasks when trained on-the-fly on fresh data [7]. In this context, continual learning is the capacity of a model to constantly pick up new information over time without losing track of previously learned information [8]. Continual learning makes lifelong learning situations possible by letting models build up knowledge over time, allowing them to adjust and perform better as they come across new tasks or information. For real-world applications where data is non-stationary and changes over time, this feature is essential.

To tackle the problem of CF, methods including rehearsal, parameter regularization, episodic memory, and task-based architectures are used. Focusing mainly on task-based architectures, Mirzadeh et al. [9] highlight the impact that different neural network architectures have on CL performance. Contrasting with the "traditional" approaches, their research demonstrates how different architecture choices can affect a model's ability to learn new tasks and retain previous knowledge. A thorough analysis of various architectural features and their influence on CL is presented, showing that in some cases, architectural modifications alone can enhance performance and give practical recommendations for optimizing CL architectures. In addition, most of the times CL task-based architectures are inspired by how researchers presume human brains work, particularly in terms of memory consolidation and reconsolidation [10].

A. AUTOML

Automated machine learning, AutoML, represents a more high level approach to ML [3]. At its core, AutoML focuses on automatically determining the most suitable models and parameters for a given task. This alleviates the need for extensive manual intervention in model selection and optimization, aiming to simplify the ML workflow. By navigating through a variety of options, AutoML identifies optimal solutions tailored to specific data and objectives, resulting in a step towards more autonomous and efficient machine learning processes.

The study presented in [11] focused on evaluating and comparing the capabilities of different AutoML tools in automating tasks in ML pipelines, including aspects such as data preprocessing, feature engineering, model selection, and

hyperparameter optimization. The study shows evaluations using various datasets to examine performance differences and highlight the strengths and weaknesses of each tool. The authors concluded that most tools were able to achieve reasonable results but that there was no ''perfect tool'' able to outperform all others on a multitude of tasks, indicating that, similarly to the situational and contextual applications of ML approaches, different AutoML approaches can also be situational and contextual. This general concept of automatically searching for the right method for a given task is something the present work aims to apply to all aspects of the architecture (MDNN2.0) when more than one potentially viable option is available.

Similarly, key areas such as data preparation, feature engineering, and hyperparameter optimization are considered in [3], but with a particular focus on neural architecture search. They emphasize this as a critical and active area of research within AutoML. The authors discuss the performance of representative neural architecture search algorithms, particularly on benchmark datasets like CIFAR-10 [12] and ImageNet [13]. Other studies provide a comprehensive review of the current advancements in AutoML, particularly focusing on the integral elements of the ML pipeline, see e.g. [14].

B. DATASETS

A major issue in CL research is the lack of suitable datasets [15]. While datasets like ImageNet [13] are very popular for object classification, they lack specific characteristics required for CL testing. Other datasets such as TinyImagenet [16], CIFAR [12], and iNaturalist [17] are sometimes used to test CL methods (see e.g. [18]), nevertheless they do not have the specifications to reliably validate CL methods.

The ideal dataset for CL should have a significant number of views for each class obtained throughout various sessions, in addition to numerous classes added in batches. Another important component is the existence of temporal coherent sessions, or films in which the objects move subtly in front of the camera. This is because temporal smoothness may be applied to handle unsupervised circumstances, enhance classification accuracy, and make object recognition easier.

CORe50 [5] has these crucial components, several (unrestricted) views of the same items captured during various sessions (with variable backdrops, lighting, occlusions, and poses), this being the reason why the present work uses CORe50 to test the proposed architecture. Despite the above, it is important to stress that others datasets for CL exist, for instance the CLOC dataset [19] for visual localization tasks.

C. CONTINUAL LEARNING METHODS

During the last decade, several methods were used to explore CL, such as the ones presented in [20], [21], [22], [23], [24], and [25].

Focusing on the last 5 years, Maltoni and Lomonaco [26] proposed AR1, an approach that combines architectural and regularization strategies to mitigate catastrophic forgetting; and Parisi et al. [7] explored three CL approaches: network retraining using regularization, training specific network sections, and expanding as needed, and methods modeling complementary learning systems for memory consolidation.

Pellegrini et al. [27] introduced latent replay for real-time CL, separating low-level and high-level feature extraction, and controlling their training rates. The ''Modular Dynamic Neural Network: A Continual Learning Architecture'' [28], was introduced by the present authors in 2021. It features a ResNet50-based feature extraction component and a modular dynamic classification component, with a tree structure, that rearranges and groups classification modules as it learns new classes.

Shaheen et al. [29] discussed the application of CL in autonomous systems, while Wang et al. [30] introduced the lifelong vision transformer (LVT), with features including an inter-task attention mechanism and a dual-classifier structure, which collectively reduce CF and improve overall performance. Arani et al. [31] introduce CLS-ER, a CL method inspired by the brain's complementary learning systems, employing dual memory for experience replay to mitigate CF in DNNs. The dual memory system maintains short-term and long-term memories, using what they refer to as a plastic model and a stable model respectively, enabling efficient learning and knowledge retention by updating the plastic model more frequently to adapt quickly to new information, and updating the stable model less frequently so that it retains more information from earlier tasks.

Smith et al. [32] investigation into rehearsal-free CL focuses on mitigating CF without relying on data rehearsal. Their approach, while distinct from rehearsal-based methods, contributes to the broader dialogue on optimizing CL strategies under constraints of memory efficiency and data privacy. Their findings, particularly around the effective use of knowledge distillation and parameter regularization, demonstrate the potential of alternative mechanisms to preserve knowledge across evolving tasks. In [33], Yang et al. presented a Bayesian generative model for continual learning that uses a fixed pre-trained feature extractor. Unlike typical class-incremental methods, this approach represents old class knowledge with statistical distributions, such as Gaussian mixture models, effectively preventing forgetting in CL scenarios. This method was tested on medical and natural image classification tasks, outperforming current state-of-the-art techniques that retain images of old classes. Its reliance on a fixed feature extractor aligns with our previously proposed MDNN method, emphasizing the efficacy and potential of building models on top of a stable, pre-trained foundation for knowledge retention and continual learning.

In addressing the challenges of on-device learning, particularly for resource-constrained devices, Vorabbi et al. [34] explored the integration of binary neural networks (BNNs)

with CL techniques, introducing a novel approach that leverages binary latent replay activations to minimize memory requirements while maintaining performance. This study demonstrates that by optimizing the quantization of both forward and backward passes, and continually adapting the convolutional layers and classification head, it is possible to significantly enhance the accuracy and efficiency of on-device training. The proposed method not only addresses the limitations of floating-point operations but also reduces memory use, making it a promising solution for continual learning in real-world scenarios where computational resources can be limited.

Having explored various approaches to mitigate CF and the dynamic architectures designed for CL, it is important to acknowledge the broader challenges that these methodologies have to face in real-world applications.

Mundt et al. [35] offer a critical perspective, highlighting the limitations of the noticeable common focus on benchmark performance under closed-world assumptions. They argue that for a more holistic approach to continual learning, lessons from both open set recognition and active learning should be incorporated. This perspective underscores the necessity of developing CL systems that not only efficiently manage memory and computational resources but also demonstrate adaptability and resilience in the face of unknown and evolving data distributions. Thus, integrating these insights, it becomes evident that future advancements in CL must start moving outside of traditional benchmarks, embracing more versatile forms of evaluation to measure the capabilities of navigating the complexities of real-world applications.

In [36] a real-time evaluation benchmark for online continual learning (OCL) is proposed. Using the CLOC dataset [19], a large-scale dataset with 39 million time-stamped images for visual localization with sequential images. Their findings reveal that a simple baseline method outperforms current state-of-the-art CL approaches under these realistic conditions. This suggests that most existing CL literature is not yet suited for practical, high-throughput data streams. In that context, the study advocates for the development of methods that consider computational efficiency alongside learning effectiveness.

Other recent surveys in the CL field can be accessed in [18], [37], and [38], with few-shot approaches being the main focus in [39].

In summary, the developments in CL research highlight the field's commitment to exploring diverse methodologies and, especially in practical applications, the need for continual advancements and new strategies to pursue adaptable and scalable solutions.

III. MODULAR DYNAMIC NEURAL NETWORK WITH AUTOML CLASSIFICATION TASKS

This section describes the MDNN2.0 architecture, beginning with a summary of the foundational architecture proposed by the authors, the MDNN [28], which forms the basis of the present approach, following with the innovative integration of AutoML within this framework.

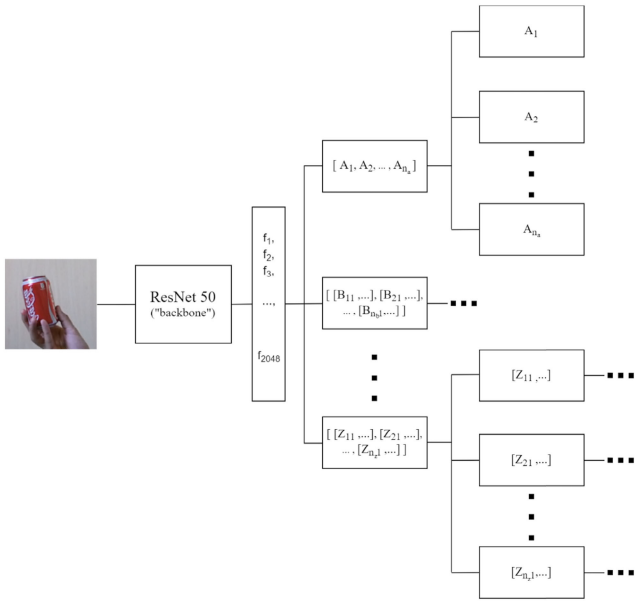

FIGURE 1. Generic representation of the modular dynamic neural network architecture, adapted from [28].

The MDNN consists of two primary components. (i) *Feature extraction* – This segment of the network uses a pre-trained ResNet50 model (ImageNet weights). Responsible for extracting generic features from input data, this component serves as the backbone of the MDNN and MDNN2.0. The stability and consistency of this component are crucial as it supports the functionality of the subsequent classification component. Therefore, it remains static and its weights are never altered, as this would invalidate the following component of the architecture which is dependent on its consistency.

(ii) *Modular Dynamic Classification* – The core of MDNN's innovation lies here. This component is an assembly of numerous binary classifiers (BCs). These classifiers are organized in a tree structure, enabling them to operate independently and dynamically. As the network encounters new classes, it adapts by modifying specific sub-networks and adding new ones, ensuring that the learning of new classes does not interfere with previously acquired knowledge. This second component will be explained in much more detail throughout this section. Figure 1 provides a generic representation of the MDNN architecture, illustrating the interconnectedness of the *feature extraction* (on the left) and *modular dynamic classification* components (on the right).

A. MODULAR DYNAMIC CLASSIFICATION

In the MDNN architecture, the classification of new data is a systematic and multi-layered process, initiated once an input is presented to the network. The classification of this input commences with the previously mentioned *feature extraction*

component, where a pre-trained ResNet50 model extracts low-level features from the data. This step is essential for preparing the data for the subsequent, more nuanced, stages of the process. The features extracted are represented in Fig. 1 by $f_1, \ldots, f_{2048}$ (more details in [28]).

Once the data passes through the feature extraction phase, it enters the *modular dynamic classification* component. This segment consists of an assembly of binary classifiers, each finely tuned to distinguish between specific object classes and groups of classes. These classifiers are not isolated units; they are part of a dynamic, hierarchical tree structure. This structure allows the binary classifiers to guide the input through a path of classification stages. Figure 1 demonstrates how the modules fit into the proposed architecture: the modules with information inside brackets contain their own sub-modules (e.g., $[X_1, X_2, \ldots, X_{n_x}]$) or groups of sub-modules in the case of nested brackets (e.g., $[[X_1, X_2, \ldots, X_{n_x}], \ldots]$), where X_1 to X_{nx} are modules with no children, and n_x represents the number of sub-modules belonging to their parent. The modules containing their own sub-modules are designated as *node* modules, and they have one BC per direct child module. The modules with no children are named *endpoint* modules and contain nothing but feature data obtained during the training process.

Following the above, the classification process within MDNN is akin to navigating a decision tree, where each node represents a BC. The composition of each BC for the MDNN architecture (in summary) consists of six fully connected layers with 128, 64, 64, 32, 32, and 2 units (''neurons''), respectively, with rectified linear activation functions (ReLU), with the exception of the last layer where the activation function is a Sigmoid function.

As the input traverses this tree, each BC evaluates it against its trained understanding of specific classes. The decision at each *node* dictates the direction the input takes —e either branching off into a more specialized classifier or progressing towards an *endpoint*. This ensures that the input is evaluated at various levels of specificity, from broader categories down to finer-grained distinctions.

One of the key advantages of this modular structure is its ability to scale efficiently as the number classes increases. In large-scale real-world deployments, this approach allows for the distribution modules across different GPUs or hardware platforms, prioritizing the most frequently used classifiers while offloading less common modules to conserve memory when needed. This distribution also makes it possible to parallelize the processing of sibling modules as they can be processed simultaneously and independently of one another. This flexibility makes the architecture highly adaptable to varying hardware constraints and resource limitations.

This modular structure enables dynamic adaptation to new classes without overwriting old knowledge, effectively mitigating catastrophic forgetting. While other modular approaches also tackle this issue, MDNN's ability to group similar classes and assign specialized classifiers allows it to efficiently manage increasing task complexity. This grouping mechanism ensures that, as the number of classes grows or class similarity increases, the network can scale without sacrificing performance. While direct numerical comparisons with other methods on the CORe50 dataset have so far only been with non-modular continual learning methods, future benchmarking on datasets with more tested methods will enable us to make broader comparisons, including with other modular-based approaches.

In essence, the classification process in MDNN is a combination of feature extraction and hierarchical decision-making, structured in such a way that it can handle complex and evolving datasets. For more details about how classes are organized in *nodes* and *endpoints*, as well as previous tests on the CORe50 dataset, demonstrating that the MDNN architecture can achieve state-of-the-art results in CL using this classification method, see [28].

It is important to stress that, in some instances, the performance of MDNN was comparable to networks trained on all classes simultaneously, highlighting its efficiency in learning incrementally without significant loss of accuracy on previously known classes, but nonetheless, still had room for improvement and optimization.

1) TRAINING: DYNAMIC GROWTH

The architecture's modularity allows for a dynamic growth pattern. When new classes are introduced, the network expands by calculating which of the known classes are the most similar to the new one and training new BCs specifically responsible for recognizing and distinguishing between them.

As the network progressively learns, the assignment of new classes to specific modules happens automatically by performing inference on the new data with the existing classifiers. Even though they are not yet familiar with the new class, the results from these classifiers that were trained on data from previously learned classes are used to estimate similarity and predict which ones will most likely be confused by samples from the new class being added, and it can then be decided, based on an empirical threshold value, which classes are similar enough to need to be grouped and have specialist classifiers trained to distinguish between them.

The flowchart in Fig. 2 shows a visual representation of the grouping logic. For a detailed explanation of the algorithm and the various types of module grouping please see [28].

Nevertheless, for a better comprehension of the algorithm and results, it is important to refer to the similarity score of ϵ (in Fig. 2 referred to as *threshold*), which is used to determine whether two classes bear enough resemblance to need to be grouped. In a previous study [2], ϵ was empirically set to 0.3, based on initial tests conducted across several datasets (CORe50, ImageNet, and CIFAR-10). During further experimentation in this work, $\epsilon = 0.4$ was found to yield similar, if not better, results in terms of intuitive class groupings. This intermediate value balances the network's behavior: lower values lead to over-grouping, while higher values make the model too selective

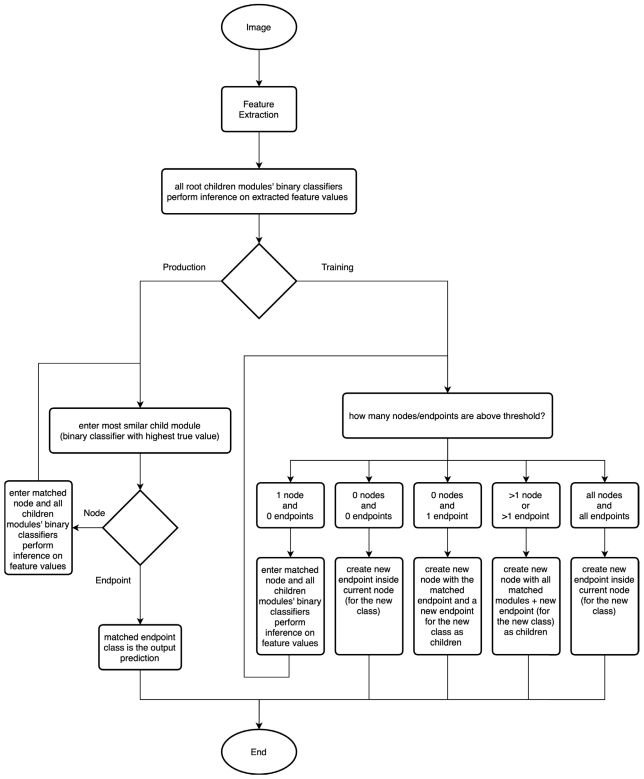

FIGURE 2. A global flowchart of the MDNN architecture for training and prediction, adapted from [28].

resulting in little to no class groupings. By using $\epsilon = 0.4$, the MDNN and MDNN2.0 maintain a sensible balance in grouping related classes, such as vehicles, plants, and animals, into intuitive categories and sub-categories. While this value has worked well in the current experiments, it remains flexible and could be adjusted depending on specific use cases to force more or less selective groupings. Future work will include further optimization of ϵ, potentially calculating a dynamic value based on factors like binary classifier depth or sibling modules. These ''intuitive'' groupings are further shown and discussed in the tests and results section (Section IV).

The resulting structure from this form of growth decentralizes the decision making of the entire classification process and helps ensure that the addition of new classes only impacts relevant parts of the network, significantly reducing the risk of catastrophic forgetting.

2) TRAINING: DATA SELECTION FOR BINARY CLASSIFIERS

A critical aspect of our methodology is the systematic selection of training data for the binary classifiers. In the MDNN architecture, each BC is responsible for distinguishing between a specific subset of classes and all other classes. This classification task is binary on the surface, but in effect it is not a simple binary division of two classes, but rather a differentiation between closely related subclasses.

For example, with a BC responsible for a category like ''mammals,'' the ''true'' data samples used for training would include those of various types of mammals. In contrast, the ''false'' data samples used for training this classifier would comprise all categories managed by the classifier's sibling modules in the network architecture. These ''false'' classes could be other subcategories of animals that are not mammals, like birds or fish, some of which may be terminal modules (endpoints) without further subdivisions and others may have large numbers of sub-categories. This approach ensures that each BC is finely tuned only to distinguish between its specific subset of classes and the other classes relevant to its particular section of the overall tree, enhancing the overall precision and efficiency of the MDNN.

B. MODULAR DYNAMIC CLASSIFICATION WITH AUTOML

This section represents the innovation relative to the authors previous work on MDNN [28]. Building upon the MDNN2.0 framework, our current study introduces an innovative integration of automated machine learning – AutoML – to enhance the performance of the tree's binary classifiers. All principles of training and prediction from MDNN are applied to MDNN2.0, except when mentioned otherwise.

The presented AutoML process is not static; it involves (automatically) experimenting with different classifiers (in the present study SVMs, LRs, and NNs) and hyperparameters for each BC, ensuring that each binary classifier is optimally tuned for its specific task within the tree. The unique aspect of this approach lies in the method used to construct the tree's BCs.

The introduction of multiple classifiers (SVM, LR, NN etc.) and hyperparameters into the tree brings with it a new challenge, which is the comparability of different BCs. In order to resolve this, we introduce a refined structure for our binary classifiers which now consist of a sequential arrangement of principal component analysis (PCA) [4], the classifier model itself, and an output normalization (ON) block. These blocks will be explained in detail in following subsections. Figure 3 top illustrates the new block for each individual BC.

As depicted in Fig. 3 bottom, the overall process consists of an exploration of various different classifiers (ranging from 1 to m) and parameter configurations (extending from 1 to p_m for each classifier type) for each BC. Whenever a BC is trained, multiple versions are trained sequentially using the varying classifiers and configurations, and the tree's overall F1 score is evaluated after each one (using a subset of training data dedicated to this task; more details later in this section). The F1 score was selected as it provides a more balanced evaluation of the classifier's performance, particularly in the context of imbalanced datasets.

The binary classifier variant yielding the best results is then selected and integrated into the MDNN2.0. An early stopping technique is also employed, where, if a BC variant achieves an F1 score higher than $F1_e$ (a predefined threshold for stopping further training), the training process is terminated and that

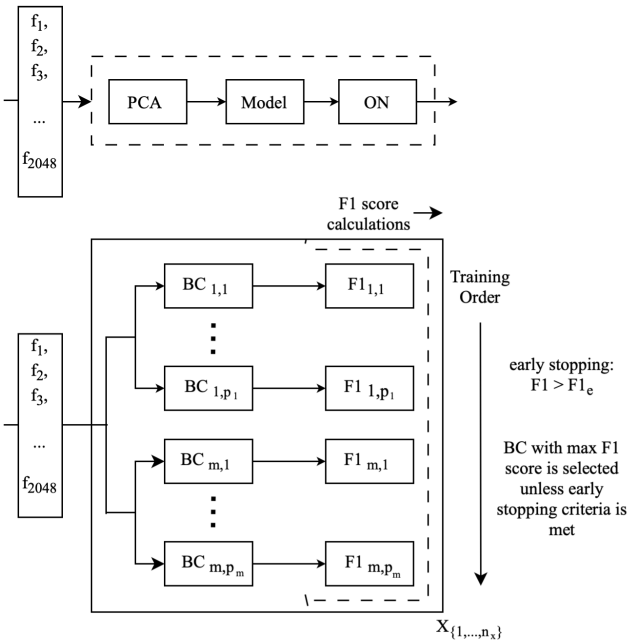

FIGURE 3. Each BC block model (in the top) and the AutoML process for BC (in the bottom), highlighting the exploration of model types (from 1 until *m*) and hyperparameter configurations for each model type (from 1 until p_m), and the BC's selection based on F1 score.

BC is selected and the remaining variants are not trained or evaluated (discussed in more detail later in this section).

By employing a dynamic selection process with AutoML, we aim to significantly enhance the framework's ability to handle binary classification tasks more effectively, which in turn is expected to have a strong positive effect on the adaptability and performance of the overall MDNN2.0. In the tests and results section, several variations of this implementation are shown using different combinations of classifiers (SVMs, LRs, and NNs) and hyperparameters. Additionally, as well as the accuracy results, we examine the different resulting trees showing the class groupings and model type/parameter selections obtained from each combination.

In summary, each sub-module and group of sub-modules in the case of nested brackets presented in Fig. 1 are now replaced with this new block, Fig. 3, but only with the BC with the highest F1 score (or the first BC to achieve an F1 score over the $F1_e$ threshold).

1) BC: PCA INTEGRATION AND TRAINING

The incorporation of a PCA module at the input of each BC in the architecture has two main points. Firstly, (i) it aims to improve the accuracy of the BC by providing them with more distilled and relevant features, effectively reducing the dimensionality of the input data. I.e., by transforming the data into a lower-dimensional space, PCA retains the most significant features while discarding less informative ones, also potentially enhancing the classifier's ability to discern between classes by focusing on the most relevant features. Secondly, (ii) it seeks to enhance computational efficiency by reducing the dimensionality of the data, thus speeding up both the training and inference processes.

Each PCA instance is trained on and applied to its respective training dataset (the same data used to train the BCs) and, as expected, is preserved for use during inference so that the same dimensionality reduction is performed during both the training and inference of the BCs. The number of components retained by the PCA is another parameter fed into the AutoML system, along with the classifier type and specific parameters. This adds a layer of flexibility, allowing us to tailor the complexity of the feature reduction to the needs of each specific task and model configuration.

Here it is hypothesized that by tuning the number of components used for each classifier, we can achieve a more precise balance between model complexity and performance. For simpler classification tasks, a lower number of principal components might suffice, reducing the processing burden without sacrificing accuracy. Conversely, for more complex tasks, increasing the number of components could capture the necessary variance in the data to maintain high classification performance.

In summary, the integration of PCA as a pre-processing step in the MDNN2.0 framework's binary classifiers contributes towards the development of a highly adaptive, efficient, and effective continual learning system.

2) BC: OUTPUT NORMALIZATION

Another important enhancement in our methodology is the implementation of output normalization for the BCs within the MDNN2.0 framework. This new step has become increasingly important with the integration of AutoML, as different classifier types and parameters are now employed in constructing these binary classifiers. Given the variability in the output ranges of these models, a normalization process is essential to ensure comparability and consistency between the BCs.

To achieve this, a subset of the training data dedicated to this task is used. This dataset, consisting of both ''true'' and ''false'' samples, is used to calibrate the output for each binary classifier. The process involves passing the dedicated subset of training data through the trained classifier in order to obtain a collection of output values that should resemble values obtained from new data, from which we calculate the mean values for the ''true'' and ''false'' samples. These mean values then define the upper and lower bounds of our normalization range respectively.

The normalization function employed is a linear mapping that scales outputs to a 0-1 range, where outputs below the lower bound are assigned a value of 0, and those exceeding the upper bound continue to scale linearly. This approach avoids the truncation of values beyond the upper bound, allowing for finer distinctions between highly confident outputs. Future output values that are lower than the lowest seen during training are trimmed to 0 because only the BCs with higher output values are taken in to consideration when

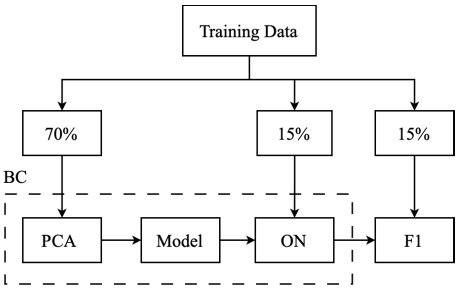

FIGURE 4. Structure of a binary classifier in the MDNN2.0 architecture, showing the PCA at the head, the classifier, and the ON block, along with the distribution of training data for each task.

making decisions for new class placement in the network and when making predictions.

This normalization mechanism is critical for two reasons. Firstly, it enables the integration of diverse classifier types and parameters within the MDNN2.0, ensuring that their outputs are comparable with each other. Secondly, it aids in maintaining the fidelity of the confidence or similarity scores during both the inference and training phases, contributing to the overall accuracy of the network.

In summary, our approach to BC output normalization is an important step that aligns with the goal of achieving an adaptable and effective continual learning system. It ensures that the diverse array of BCs, each potentially employing different classifier types and parameters, can operate in unison within the MDNN2.0 framework, thereby enhancing its overall performance in complex classification tasks.

3) TRAINING: DATA SPLITTING AND LIMITING

To manage the scale of data and ensure the MDNN2.0 remains agile and efficient, we have implemented a data splitting and limiting mechanism. This mechanism first involves dividing the training data into 3 subsets, each dedicated to a specific task: (i) training data for the classifier, (ii) data used for defining an output normalization function, and (iii) data for calculating the F1 score of each model type/parameter combination. Figure 4 illustrates BCs along with the distribution of training data across each component.

An upper limit is set on the amount of data used in each subset. This prevents the overwhelming of classifiers with excessive data, which could lead to increased training times and reduced responsiveness to new information. Especially for classifiers closer to the root of the tree, responsible for higher level categories and therefore larger amounts of sub-categories.

The split of the data is currently set to 70% for training, 15% for output normalization, and 15% for F1 score calculation. This distribution was chosen to balance the more extensive requirements of model training, which demands a larger dataset, with the more focused needs of output normalization and performance evaluation. Allocating a larger share for training ensures that the model has a comprehensive understanding of the dataset, thereby improving its predictive accuracy. Conversely, limiting the data for normalization and F1 score calculation is based on the principle that these processes require less data (in comparison) to achieve statistically significant results. This strategy not only enhances the training efficiency of the BCs but also contributes to the overall adaptability and responsiveness of the MDNN2.0 in a continual learning environment.

The limit for each subset is currently fixed at 2.400 samples for training, 600 samples for output normalization, and 600 for model F1 score calculation. In future work, these values will be dynamically adjusted based on the complexity of the classification task and the performance of the classifier. For simpler tasks, a smaller dataset might be sufficient, whereas more complex tasks may require a larger but still limited dataset to achieve the desired accuracy.

In summary, our approach to data selection, storage, splitting, and limiting is meticulously designed to optimize the training and performance of the MDNN2.0's binary classifiers. These strategies align with our objective of creating an adaptable, efficient, and effective CL system, capable of handling complex classification tasks with a high degree of precision and computational efficiency.

4) TRAINING: EARLY STOPPING IN AUTOML

Integral to the AutoML implementation is the early stopping criteria, which is applied during the training of the BCs. This mechanism is designed to cease the training process when a classifier configuration (type and parameters) reaches or surpasses an F1 score of 0.97 on its previously mentioned dedicated sub-set of the training data. The threshold of 0.97 was chosen based on early testing, where multiple models were achieving high performance with little variance in scores, making it unnecessary to continue training past this point. While this decision was made based on experience and efficiency, the value is flexible and can be adjusted depending on the desired trade-off between training time and accuracy.

The early stopping approach offers two primary advantages. Firstly, it expedites the training process by preventing further exploration of model configurations once an optimal solution is found. However, in cases where classifiers fail to reach this threshold, future iterations of this approach may consider dynamic adjustments based on a gradient-based optimization approach, stopping not only when a threshold is met but also when there are diminishing returns or a lack of progress over several iterations. In scenarios where BC tasks are less complex, and high F1 scores are readily achievable, this approach swiftly identifies and selects effective classifiers. This leads to significant time and computational resource savings, as it avoids unnecessary iterations over the remaining models in the AutoML selection pool. Secondly, this method helps in maintaining the balance between accuracy and computational efficiency.

In summary, by combining PCA with AutoML's dynamic model selection and early stopping criteria, we enhance the framework's capacity to optimize for both performance and

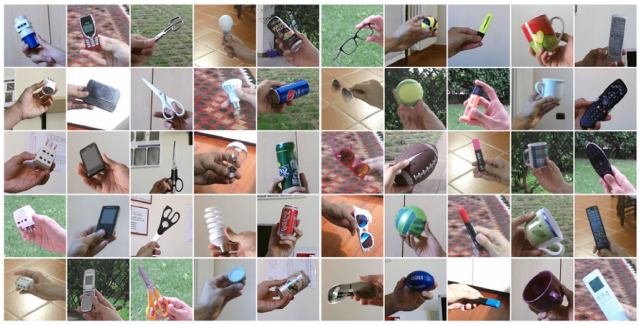

FIGURE 5. Example images of the 50 classes present in the CORe50 dataset, one image of each class.

computational efficiency in a continual learning context. By setting a high-performance threshold, we ensure the binary classifiers are both effective and efficient. This is crucial in CL environments where adaptability and swift learning capabilities are essential.

The next section explains the experimental setup and methodology used to evaluate the enhanced MDNN2.0 architecture integrated with AutoML.

IV. TESTS AND RESULTS

The experimentation involves a variety of tests on the CORe50 dataset's [5] new classes (NC) benchmark, specifically designed for testing continual learning models. The dataset consists of 50 classes (10 categories with 5 classes each). The 10 categories are as follows: plug adapters, mobile phones, scissors, light bulbs, cans, glasses, balls, markers, cups, and remote controls.

Figure 5 shows an example of each class and how they are divided into the mentioned categories. The data was collected over 11 distinct sessions (8 indoor and 3 outdoor) characterized by different backgrounds and lighting. For each session and each object, a 15-s video (at 20 fps) was recorded using a Kinect2.0 sensor producing (approx.) 300 RGB-D frames. The objects were handheld by the operator, and the camera's point-of-view is that of the operator's eyes. The full dataset consists of 164.866 RGB-D images, with a resolution of 128×128 pixels.

The NC benchmark evaluation is performed by initially introducing 10 classes and then incrementally adding one class at a time. This incremental nature allows for the monitoring of the network's learning progress and the assessment of its ability to mitigate catastrophic forgetting. The network's accuracy is calculated after the initial 10 classes and subsequently after every additional 5 classes, until all 50 have been added, resulting in 9 test steps that provide a comprehensive view of its performance over time. This experiment follows the same procedure presented in [5].

A distinctive aspect of these classes is their categorical grouping: every consecutive set of five classes belongs to a similar category (e.g., classes 1-5 are plugs, 6-10 are phones etc.). This characteristic is particularly relevant to our architecture because it also gives us insights in to our method's learning and grouping strategy. Throughout our experiments, the MDNN2.0 demonstrated an intuitive and efficient approach to grouping these classes. As the method developed its tree structure of modules, it grouped classes of similar types (such as plugs or phones) together. Within these groups, the network deployed specialist binary classifiers to discern between specific items, like different models of phones or plugs.

This capability was further used by the creation of more sub-groups within these categories, especially when the network identified particular items as closely related, necessitating even more specialized classifiers for accurate distinction. This intelligent and intuitive grouping by the MDNN2.0 shows its effectiveness in understanding and classifying objects in a way that somewhat aligns with human categorization.

Tests are organized into 4 distinct experiments, each using a different assortment of model configurations of 3 main classifier types: (#*I*) SVMs, (#*II*) LRs, (#*III*) NNs, and, finally, a (#*IV*) combined approach integrating all three classifier types. This structured approach allows for a nuanced understanding of each model's impact within the MDNN2.0 framework.

Each of these three classifiers configurations within the MDNN2.0 framework is further subdivided into four distinct variants, resulting in a total of 12. The variants are structured sequentially: (#1-4) Variants 1 to 4 are associated with SVMs, (#5-8) variants 5 to 8 with LRs, and (#9-12) variants 9 to 12 with NNs. This subdivision enables a thorough examination of the nuances of each model type and variant within the MDNN2.0 framework. Variants' details are provided below.

An intriguing aspect of our experiments is the dynamic nature of the tree structure that the MDNN2.0 autonomously constructs during training. Each experiment, corresponding to a different classifier type (or combination of them) and their variants, yields a unique tree structure. These structures vary not only in the number of layers but also in the density of groupings. This variation is reflective of how each classifier type and its variants interact with the dataset and the learning process.

Each experiment's unique tree structure gives insights into the grouping densities and learning patterns of the MDNN2.0. For instance, some experiments might show denser groupings in the early layers of the tree, indicating a strong initial categorization of objects, while others might exhibit more spread out groupings, suggesting a more detailed and fine-grained classification process in the earlier stages. As we discuss each experiment in detail, we will highlight the specific characteristics of the tree structure observed, the number of layers, and the grouping densities. These insights will provide a deeper understanding of how the MDNN2.0 adapts its learning strategy and structure to accommodate different types of models and their variants.

TABLE 1. SVM binary classifier parameters and PCA component number for each variant used in the MDNN2.0.

Param./variant	#1	#2	#3	#4
Kernel	RBF	RBF	RBF	Poly
C	5, 50, 100	550, 1100	1, 10, 100	120, 550
γ	Scale	0.02, 0.15	0.002, 0.01	*scale*
Degree	–	–	–	6, 7
PCA: n	150	210	100	210

All the experiments were conducted using a single NVIDIA A100 80GB GPU using the TensorFlow and CUML libraries to leverage GPU processing for machine learning tasks. Each experiment took approximately 6-8 hours for training and testing. This time frame includes the entire experimentation process, from initial data loading to intermediate and final model evaluations.

For inference, the models demonstrated impressive performance, processing between 500 and 900 images per second. This throughput makes the models suitable for real-time applications (using the mentioned hardware).

A. EXPERIMENT #1: SVM

In this experiment, we focused on the integration of support vector machines within the MDNN2.0 framework. The specific SVM binary classifier parameters and PCA parameters for each variant are provided in Table 1. In the table, the kernel RBF refers to ''radial basis function'' (RBF), Poly to ''polinomial'', C and γ to parameters of the SVM, with C being a regularization parameter that controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights to avoid overfitting, and γ defines how far the influence of a single training example reaches, where *scale* uses 1 / number of features. *Degree* is the degree of the polynomial kernel function (''Poly''). In the PCA, n is the number of components to keep. The hyperparameters ranges were decided based on the authors' experience and previous tests, meaning that the values were selected to cover a broad range of possible configurations, ensuring that the AutoML system could explore a wide variety of models and parameters, this applies to all following experiments. Nevertheless, the ranges were not exhaustive, and future work will explore more hyperparameters and configurations, potentially incorporating dynamic methods to adjust hyperparameter values during training, rather than relying on pre-defined ranges.

The tree structure of the MDNN2.0, as shown in Fig. 6, consisted of 5 layers, indicating the depth of the network. The first layer contained 30 modules, followed by 41 in the second layer, 46 in the third, 49 in the fourth, and the final layer, as expected, consisted of 50 modules, one for each class.

A notable observation was the limited number of groupings in this experiment compared to subsequent ones, suggesting a potential need for a lower grouping threshold for SVMs. The groupings that did occur, however, were consistently among similar classes. Regarding the use of SVM variants,

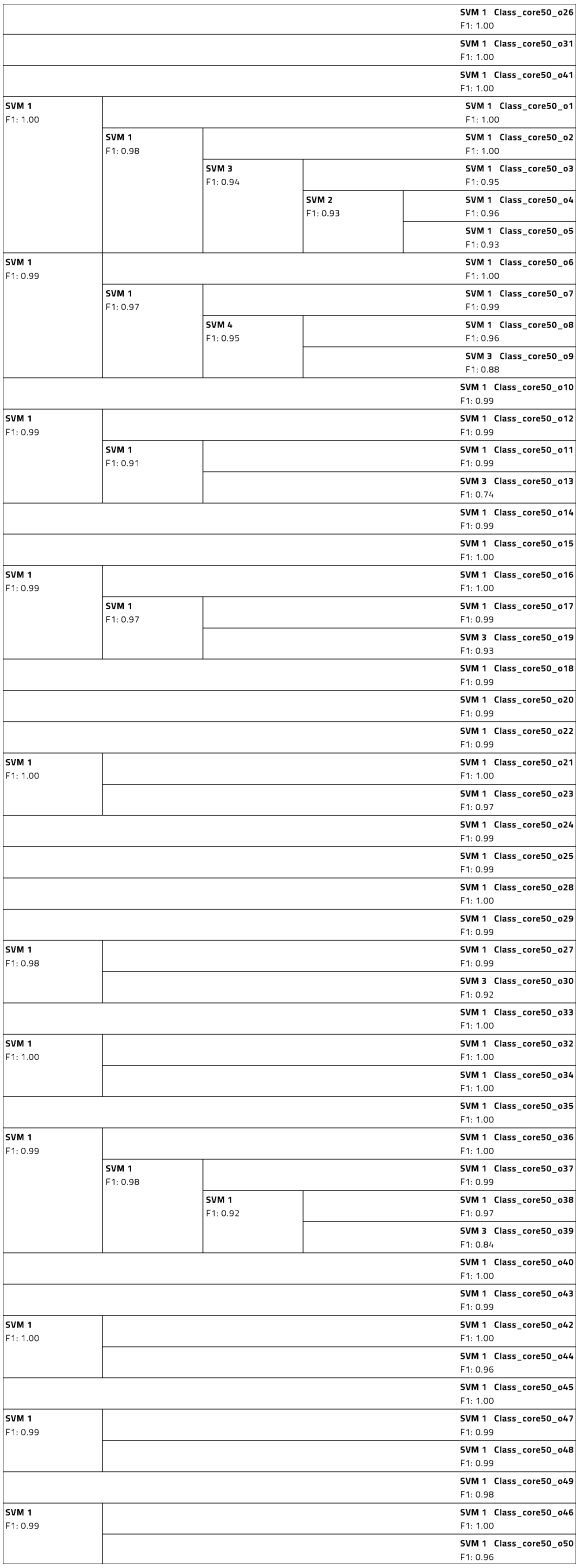

FIGURE 6. Graph showcasing the tree structure that was automatically constructed during training, and which SVM parameters were selected for each module.

with each variant represented by a different shade of grey, the overwhelming majority (62 out of 70 classifiers) employed

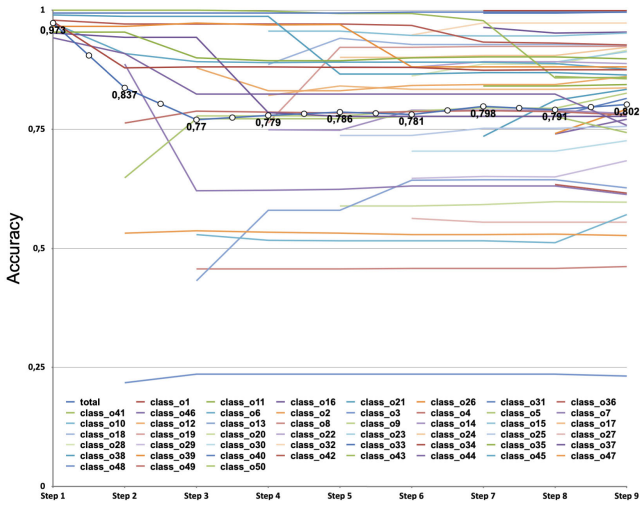

FIGURE 7. Graph showcasing the accuracy of MDNN2.0 over time on the CORe50 dataset using SVM binary classifiers.

TABLE 2. Example of the confusion matrix (only) for class o3.

Predicted class	Number of Predictions
o3 (Correct)	209
o5	402
o4	138
o2	56
o46	24
o6	15
o9	16
o7	13
o50	9
o26	7
8 classes with ≤ 2 predictions each	11

TABLE 3. Logistic Regression binary classifier parameters and PCA component number for each variant used in the MDNN2.0.

Param./variant	#5	#6	#7	#8
C	0.2, 2, 20	100, 500, 1000	50, 100	60, 120
max_iter	100, 500	10000	5000, 10000	5500, 11000
PCA: n	90	200	150	160

variant #1, with minimal utilization of the other variants: variant #2 (1 classifier), variant #3 (6 classifiers), and variant #4 (1 classifier).

The performance of the classifiers, measured by F1 scores, was consistently high across the board, with all scores above 0.9. However, the overall accuracy of the models did not mirror this high performance. This discrepancy may indicate a difference in the distribution of the training data subset used for F1 score calculation compared to the test data. The initial step of testing (shown in Fig. 7) which involved the first 10 classes, showed an average accuracy of 0.973, indicating a strong start for the network. Figure 7 illustrates the accuracy over time for each class. The blue line with white dots highlights the overall average accuracy, while the individual lines represent class-specific accuracies, allowing for a general view of the trends in how different classes were learned. As more classes were added (in groups of 5), we observed a decline in total accuracy, dropping to 0.837 in the second step and further to 0.77 in the third. However, this accuracy stabilized and improved to 0.802 by the final step.

An interesting case was that of class o3, a type of plug, which consistently underperformed, with accuracies just above 0.2. We studied the confusion matrix for this class, shown in Table 2, that showed a frequent misclassification of class o3 as class o5 and class o4, which represent two other similar-looking plugs. This highlights the challenges in distinguishing between highly similar objects. Despite the lower performance of some classes, the overall trend did not suggest catastrophic forgetting, as the accuracies remained stable within a range of 0.032 after the initial drop.

In summary, experiment #*I* with SVM classifiers in the MDNN framework demonstrates the network's ability to learn and classify objects incrementally. While there were challenges in maintaining high accuracy with the addition of new classes, particularly for similar-looking objects, the network's performance did not exhibit signs of catastrophic forgetting, maintaining a stable accuracy profile throughout the experiment.

B. EXPERIMENT #II: LR

In the second experiment the integration of logistic regression models within the MDNN2.0 framework is evaluated. The LR experiment revealed significant differences in the tree structure compared to the SVM experiment.

The MDNN2.0's tree structure in this experiment was more layered and uniformly distributed in terms of groupings. Specifically, the tree consisted of 7 layers, indicating an increased depth compared to the SVM experiment. The distribution of modules across these layers was as follows: 5 modules in the first layer, 10 in the second, 21 in the third, 31 in the fourth, 41 in the fifth, 48 in the sixth, and, finally, 50 in the last layer, consistent with the number of classes.

Table 3 shows the parameters used for each variant, with C representing the inverse of the regularization strength, like in SVM, smaller values specify stronger regularization, and *max_iter* is the maximum number of iterations taken for the solvers to converge.

In terms of variant usage, there was a more varied selection compared to the SVM experiment. Variant #5 was used 46 times, variant #6 appeared 28 times, variant #7 appeared 3 times, and variant #8 was used twice. The total number of BCs used in this experiment was 79. This distribution, particularly the frequent use of variant #5, is attributed to the AutoML's early stopping feature, as this is first variant to be trained from the list, and the F1 scores are frequently over the early stopping threshold ($F1_e = 0.97$).

Figure 8 illustrates the automatically constructed tree structure during training and the selected LR parameters for each module. Similar to the SVM experiment, the grouping patterns continued to exhibit logical categorizations of similar classes. Although the grouping was not as tidy as in the SVM

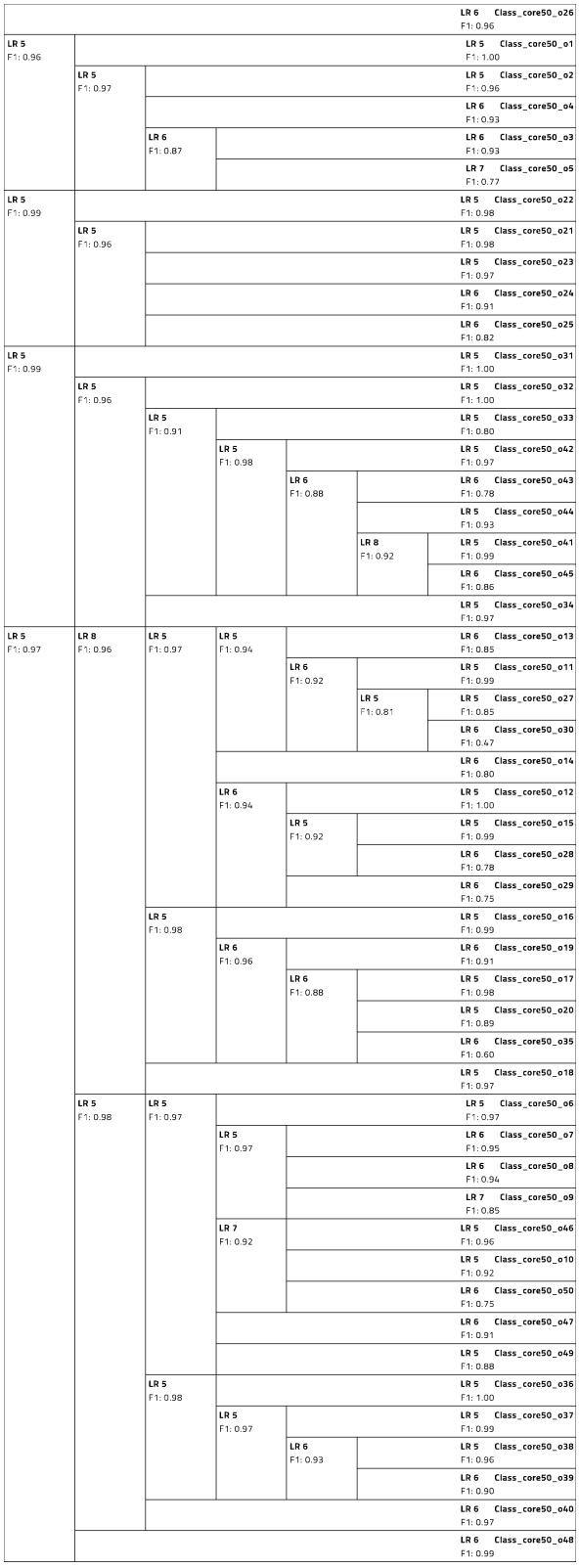

FIGURE 8. Graph illustrating the automatically constructed tree structure during training and the selected LR parameters for each module.

case, the pattern of clustering similar object types (like plugs, phones etc.) was evident.

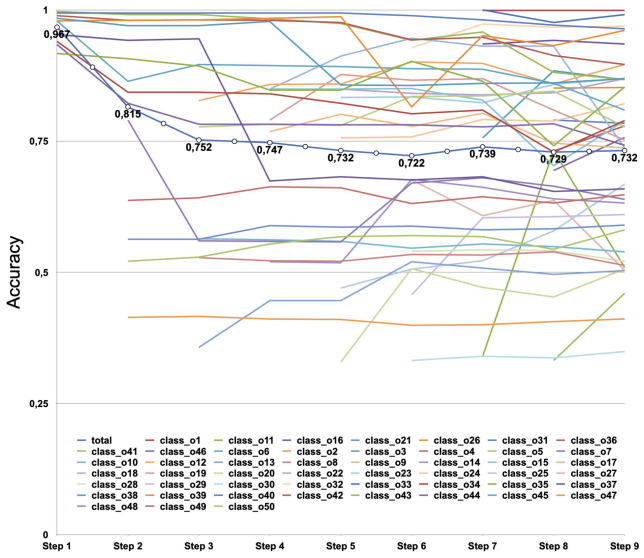

FIGURE 9. Graph depicting the accuracy of MDNN over time on the CORe50 dataset using LR binary classifiers.

TABLE 4. Neural Network binary classifier parameters and PCA component number for each variant used in the MDNN2.0.

Param./variant	#9	#10	#11	#12
Epochs	15	40	100	100
Batch Size	16	64	16	8
Layers	[100, 50, 2]	[100, 80, 40, 20, 2]	[210, 110, 110, 60, 60, 30, 30, 2]	[255, 210, 160, 110, 60, 60, 30, 30, 2]
PCA: n	100	100	210	255

When analyzing accuracy results (Fig. 9), the initial step with 10 classes showed high accuracy, similar to the previous experiment, with an average of 0.967. Figure 9 provides a visual representation of accuracy over time for the LR experiment, with each line representing the accuracy of one class and the blue line with white dots representing the overall average accuracy allowing us to observe both overall trends and class-specific variations. As more classes were added, the accuracy dropped to 0.815, then to 0.752, and finally stabilized at 0.732, towards the end of the experiment. Unlike the SVM experiment, there were no classes with extremely low accuracy, but there was a visibly higher variability in class performance. In addition, the overall accuracy was lower and more variable compared to the SVM experiment, catastrophic forgetting was mostly avoided. Most classes did not show drastic declines in accuracy over time, with the most significant drops being around -0.25 for a few classes.

In summary, experiment *#II* with LR classifiers in the MDNN2.0 framework demonstrated the tree's ability to adapt its learning and grouping strategy using a different type of model. The increased depth and uniform distribution of the tree structure, along with the more variable accuracy results, demonstrated the network's continual learning capabilities when employing LR models.

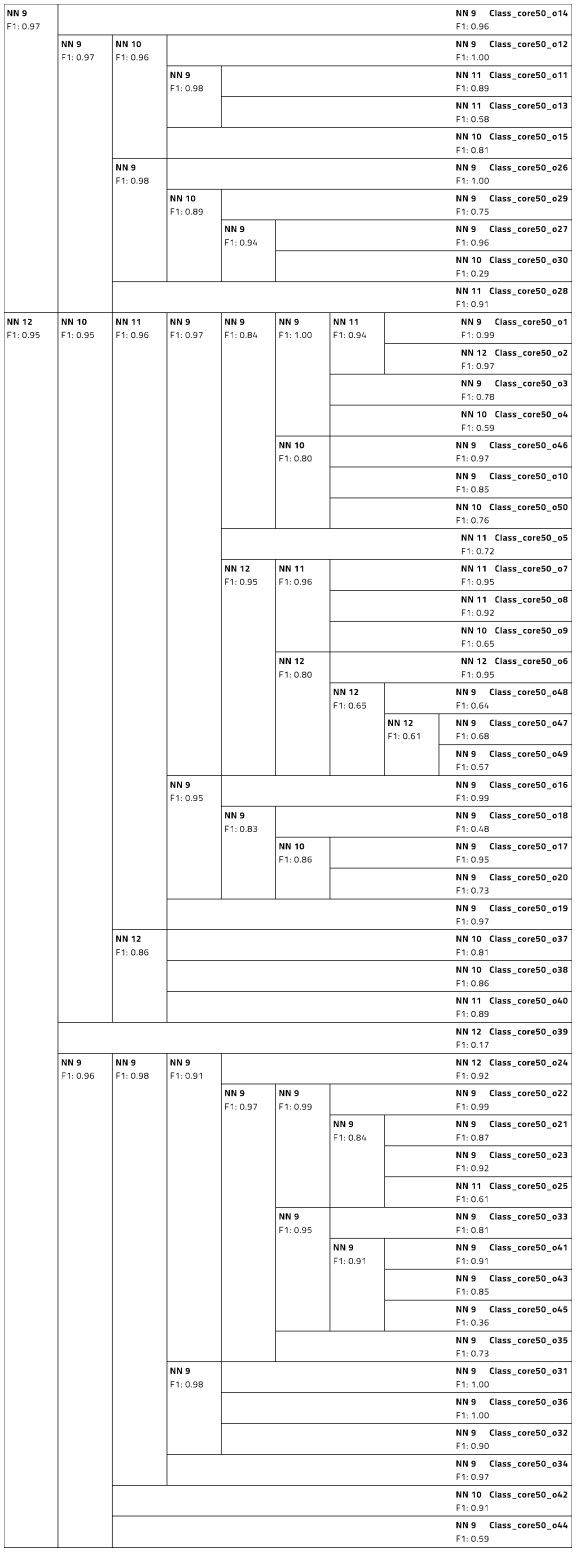

FIGURE 10. Graph showing the tree structure automatically constructed during training and the NN parameters selected for each module.

C. EXPERIMENT #III: NN

In the third experiment the use of neural networks within the MDNN2.0 architecture was investigated. This experiment

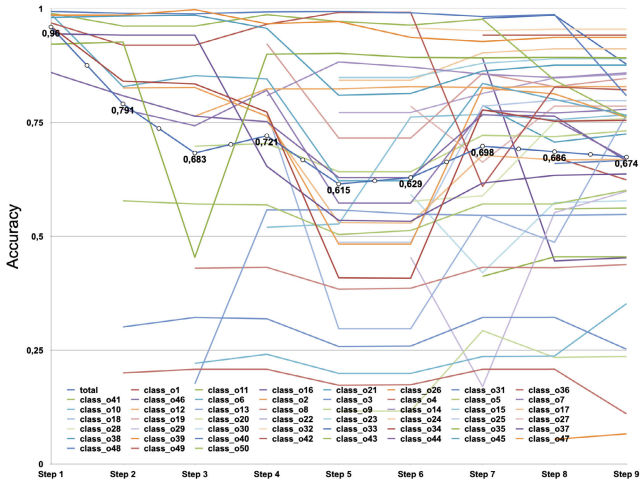

FIGURE 11. Graph displaying the accuracy of MDNN over time on the CORe50 dataset using NN binary classifiers.

presented a distinct tree structure and varied results, offering insights into how more complex classifiers, like NNs, interact within the MDNN2.0 framework.

An important note on the architecture is that the first layer always corresponds to the size of the PCA components, as this represents the input size for the NNs. The varying layer sizes in Table 4 reflect this structural detail.

The tree structure in the NN experiment (Fig. 10) exhibited the highest number of layers among all experiments, with a total of 9. This more layered approach indicates that a deeper and more nuanced classification process was needed by the MDNN2.0 when using NNs. The pattern of grouping the similar classes, as observed in previous experiments, was less pronounced here, though still evident.

The distribution of modules across the layers was as follows: 2 modules in the first layer, 5 in the second, 10 in the third, 19 in the fourth, 27 in the fifth, 33 in the sixth, 43 in the seventh, 49 in the eighth, and, finally, 50 in the last layer. The variety of model variants used in the NN experiment was the most evenly distributed among all experiments, possibly due to the rarity of achieving early stopping, as the models frequently failed to demonstrate sufficiently high performance. The counts for each variant were as follows: Variant #9 was used 48 times, variant #10 13 times, variant #11 11 times, and variant #12 10 times, totaling 82 binary classifiers.

The accuracy results for individual classes in this experiment (Fig. 11) were notably inconsistent. Figure 11 shows the accuracy for each class during the neural network (NN) experiment, with the blue line with white dots marking the average accuracy and each individual line representing a specific class. This highlights a wider range of accuracy fluctuations observed with NNs. Starting off with an average accuracy of 0.96, it then decreased to 0.791, followed by a further drop to 0.683. Subsequently, accuracy fluctuated between 0.7 and 0.6, eventually settling at 0.674.

This represented the lowest average accuracy observed in our experiments. The range of accuracies for each class was also the most varied, spanning from nearly 0 to 0.95. Notably, there was a significant drop in accuracy for most classes between the fourth and fifth steps, followed by an increase from the sixth to the seventh step.

This experiment's findings suggest that the more complex NN models did not perform as well as the simpler ones in this specific context. This could be attributed to the feature extraction and PCA preprocessing steps, which may be more compatible with less complex model types.

In summary, experiment #*III* with NN classifiers in the MDNN2.0 framework demonstrated that complexity does not necessarily equate to better performance in this continual learning setting. The deeper tree structure and the variability in class accuracies highlight the challenges and nuances of employing NNs for this type of task.

D. EXPERIMENT #IV: COMBINED APPROACH

In this final experiment, all model configurations were combined (SVMs, LRs, and NNs) to evaluate their collective impact on the MDNN2.0. The AutoML system was tasked with selecting the most suitable model from the entire range, starting with the first SVM variant and progressing through all SVMs, LRs, and then NNs. Combined model parameters are: (i) SVM parameters in Table 1, (ii) LR parameters in Table 3, and (iii) NN parameters in Table 4.

The resulting tree structure of the MDNN2.0 in this experiment (Fig. 12) was notably different from the others. With only 4 layers, it displayed a more streamlined approach to classification. The layers were distributed as follows: 30 modules in the first layer, 41 in the second, 46 in the third, and 50 in the fourth. Interestingly, this tree closely resembled the structure from Experiment #*I*, which solely used SVMs, but incorporated a few other model types for specific tasks.

The distribution of model variants used in this experiment was as follows:

- SVM: Variant #1 was used 58 times.,
- LR: Variant #5 was used twice and variant #6 was used 4 times.
- NN: Variant #9 was used once and variants #11 and #12 were each used twice.

The total number of binary classifiers used was 69, with the first SVM variant triggering early stopping frequently. When the first SVM variant was not the best fit, the system utilized a mixture of LR and NN variants.

The accuracy results in this experiment (Fig. 13) showed a pattern similar to experiment #*I*. Figure 13 compares the performance of the combined approach (SVM, LR, NN) in terms of accuracy over time. The individual lines represent each class's accuracy, while the blue line with white dots tracks the overall average accuracy. The initial average accuracy was 0.973 (identical to step 1 in experiment #*I*), which gradually decreased to around 0.8, with each step slightly higher than those in experiment #*I* by approximately 0.01 or 0.02. The final overall accuracy was 0.811.

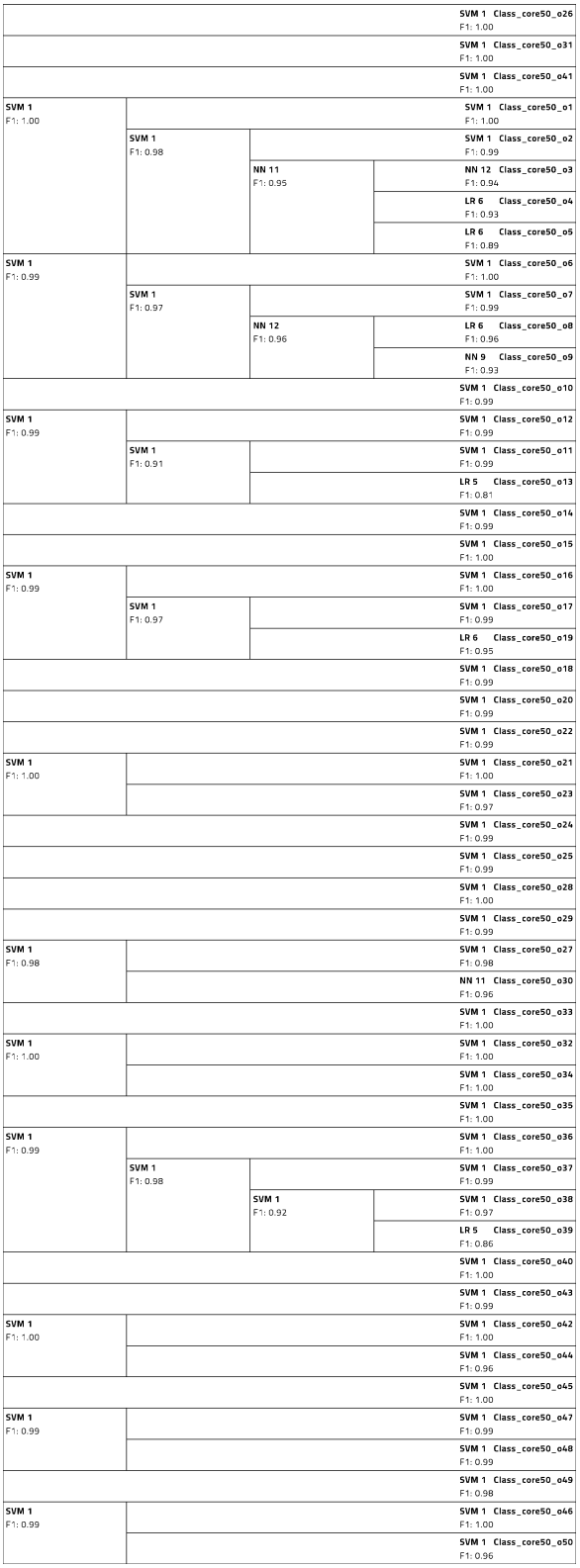

FIGURE 12. Graph showing the tree structure automatically constructed during training with all 12 (4 of each) model type-parameter combinations.

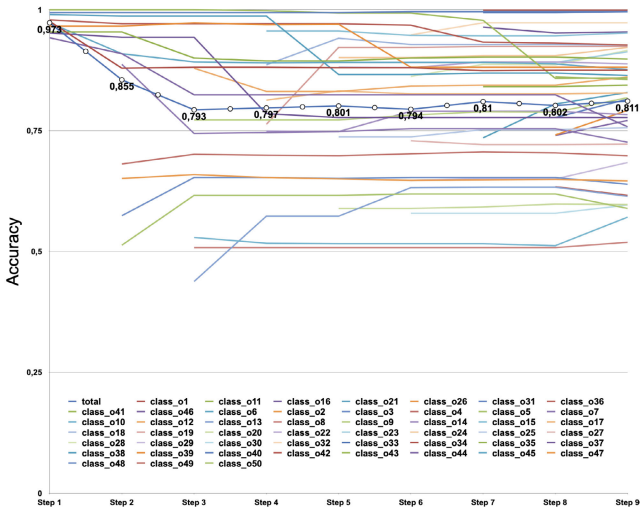

FIGURE 13. Graph illustrating the overall performance of MDNN2.0 using a combined approach of SVM, LR, and NN classifiers on the CORe50 dataset.

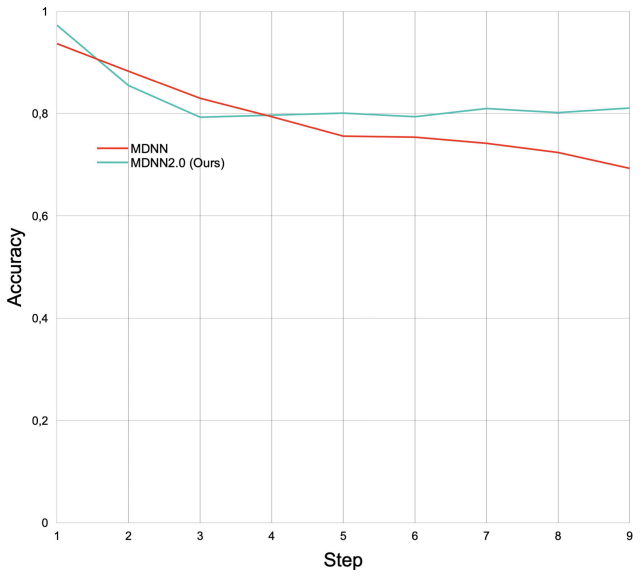

FIGURE 14. This graph represents the accuracy of MDNN and MDNN2.0, where 100% accuracy reflects perfect classification of the classes known at each stage, starting with 10 and adding 5 at each subsequent step. Unlike adapted representations used for comparability in other studies, this format shows the real-time learning capacity of the models as new classes are introduced.

This experiment demonstrated the best overall results among all four. While the improvement was slight compared to experiment #*I*, the incorporation of a wider selection of classifiers types did not hinder the tree's performance. Instead, it appears to have provided the MDNN2.0 with more options, allowing it to enhance its accuracy and stability. The consistent class accuracies and minimal decreases further suggest that catastrophic forgetting was effectively mitigated in this approach.

TABLE 5. Comparison of final accuracies on CORe50 Dataset between MDNN2.0 and other methods.

Method	Final Accuracy
BNN+CWR*-float [34]	0.59
BNN+LR+CWR*-float [34]	0.71
BNN+CWR*-32-bit [34]	0.60
BNN+LR+CWR*-32-bit [34]	0.72
BNN+CWR*-16-bit [34]	0.60
BNN+LR+CWR*-16-bit [34]	0.69
BNN+CWR*-8-bit [34]	0.55
BNN+LR+CWR*-8-bit [34]	0.57
MDNN [2]	0.69
MDNN2 (Ours)	**0.81**

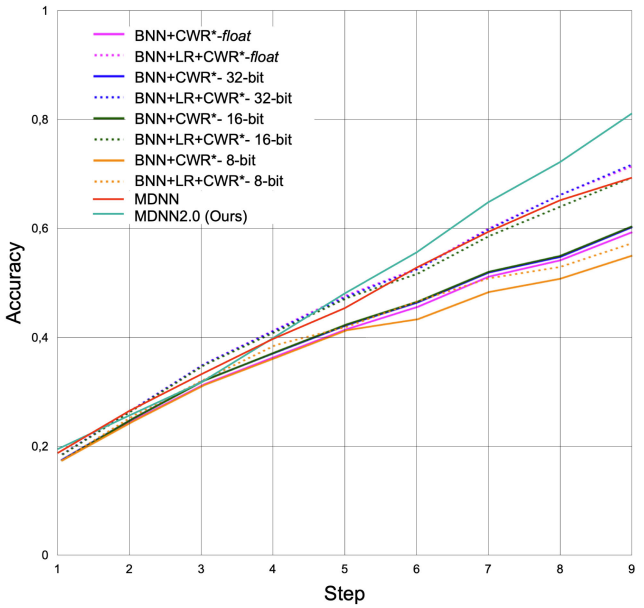

FIGURE 15. Comparison of MDNN2.0 performance: Previous architecture - MDNN (red line) versus enhanced AutoML (integrated approach) MDNN2.0 (teal line) on CORe50, compared with various BNN+CWR* configurations, adapted from [34]. The format used calculates accuracy as a percentage of the total classes regardless of how many have been taught. Hence, initially, with only 10 classes known (out of a total of 50), the maximum achievable accuracy is 20%, increasing as more classes are introduced.

E. COMPARATIVE ANALYSIS WITH OTHER AUTHORS

The previous results from the MDNN [2] paper achieved a final accuracy of around 70%, with a gradual decrease in accuracy over time. In contrast, the new results with the enhanced MDNN2.0 architecture integrated with AutoML show significant improvements, achieving a final accuracy of approximately 80%. Additionally, the accuracy stabilizes at around 80% after an initial drop (see Fig. 14), indicating more robust performance over time.

To further clarify the comparison between MDNN2.0 and other methods, we provide a detailed summary in Table 5. This table highlights the final accuracies of our method and various configurations of BNN+CWR* across different quantization settings, allowing for a more straightforward comparison of final performance metrics.

Figure 15 complements this table by visually depicting the progression of accuracy over time for MDNN (red line) and current results from our combined approach - MDNN2.0 (teal line). Note that this graph starts differently from the others, beginning with accuracies approaching 20% instead of 100%. This is because the original authors considered all 50 classes (including the unknown ones) throughout the training steps, making the initial maximum achievable accuracy for this benchmark 20% (10/50).

This comparison highlights the advancements made in the MDNN2.0 architecture, demonstrating improved accuracy and better stability in learning, which is crucial for continual learning models. These improvements suggest that the modifications and integration of AutoML with the MDNN2.0 have successfully enhanced its learning efficacy and ability to mitigate catastrophic forgetting. The graph also compares the previous and current MDNN architectures with various BNN+CWR* configurations, which are enhanced versions of Consolidated Weight Rehearsal that adapt the model by reloading and adjusting past weights, while also using different quantization values. For a detailed explanation of these methods, please see [34].

V. CONCLUSION

This research aims to contribute significantly to the field of CL, particularly in the context of object classification. By integrating dynamic and automated approaches within the established MDNN2.0 framework, the aim is to enhance the capabilities of continual learning, paving the way for more adaptable and efficient AI systems.

Reflecting on the key findings from the experiments made and the goals set out in the introduction, the research aimed to enhance object classification in CL by integrating AutoML with the Modular Dynamic Neural Network architecture. This integration targeted improvements in binary classification tasks, enhancing adaptability and performance.

The experiments revealed significant insights: Experiment #*I* (SVM): Showed the MDNN2.0's ability to learn and classify objects incrementally. Despite some challenges in maintaining high accuracy with the addition of new classes, catastrophic forgetting was mostly avoided. Experiment #*II* (LR): Demonstrated the MDNN2.0's adaptability in learning and grouping strategy with logistic regression models. The experiment showed a variable accuracy but overall stability, indicating the avoidance of catastrophic forgetting. Experiment #*III* (NN): Highlighted the challenges of employing more complex neural network models within the MDNN2.0, suggesting that simpler classifiers might be more compatible with the network's architecture and learning process, especially when considering its pre-processing blocks (feature extraction and PCA). Experiment #*IV* (combined approach): Yielded the best overall results, indicating that a diverse model approach does not hinder performance and may provide more options to enhance accuracy and stability.

One notable decision in our comparative analysis was the exclusion of the CLOC dataset [19]. The primary reason for this is the fundamental difference in the nature of the datasets and their respective applications. The CLOC dataset is designed for location estimation tasks, which is inherently different from object classification tasks that involve identifying and categorizing objects within an image, which is what our proposed method is designed to do.

It is also important to stress, that the MDNN2.0 architecture can efficiently handle the long-term accumulation of binary classifiers because of its hierarchical structure. During inference, only relevant subsets (and then subsets of subsets etc..) of classifiers are evaluated based on the hierarchical groupings of classes, ensuring that the entire network is not used for each classification. This selective evaluation greatly reduces processing overhead, making the system extremely scalable even as the number of classes grows. To prevent overfitting as new tasks are introduced, the system evaluates potential areas of confusion when adding new classes and dynamically places specialist classifiers to handle these challenging cases. This targeted strategy ensures that the model focuses on reducing misclassifications, minimizing the risk of overfitting as the classifier base expands.

In summary, the paper's findings contribute to the understanding of how different model types and their integration within a CL framework can affect learning and memory retention, paving the way for future research to further optimize these systems.

Additional future work was already mentioned in the text, which involves, but is not limited to, experimenting with additional classifier variants, optimizing current classifiers, and automatically and dynamically calculating the threshold for selecting each BC. In particular, we plan to explore a more dynamic approach to hyperparameter selection and early stopping allowing the system to more effectively search for optimal configurations. For the early stopping mechanism, as well as exploring alternative threshold values in different situations, in order to pair with the dynamic hyperparameter selection we plan to incoorporate additional stopping criteria, such as detecting stagnation in performance improvement after a certain number of iterations.

In terms of scalability, future work will explore the ImageNet dataset [13] to evaluate the performance of our models when faced with a much larger number of object classes, where we will also be able to witness the similar class groupings on a larger scale and measure their quality using clustering evaluation metrics. Furthermore, we plan to explore the use of alternative feature extraction architectures alongside ResNet50. Preliminary tests have shown minimal impact when swapping feature extractors or using them in combination with ResNet50, but more experiments across larger datasets will provide deeper insights into how these changes influence overall performance.

REFERENCES

[1] E. A. Duéñez-Guzmán, S. Sadedin, J. X. Wang, K. R. McKee, and J. Z. Leibo, "A social path to human-like artificial intelligence," *Nature Mach. Intell.*, vol. 5, no. 11, pp. 1181–1188, Nov. 2023.

[2] D. Turner, P. J. S. Cardoso, and J. M. F. Rodrigues, "Modular dynamic neural network: A continual learning architecture," *Appl. Sci.*, vol. 11, no. 24, p. 12078, Dec. 2021.

[3] I. Salehin, M. S. Islam, P. Saha, S. M. Noman, A. Tuni, M. M. Hasan, and M. A. Baten, "AutoML: A systematic review on automated machine learning with neural architecture search," *J. Inf. Intell.*, vol. 2, no. 1, pp. 52–81, Jan. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2949715923000604

[4] S. J. Prince, *Computer Vision: Models, Learning, and Inference*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[5] V. Lomonaco and D. Maltoni, "Core50: A new dataset and benchmark for continuous object recognition," in *Proc. Conf. Robot Learn.*, 2017, pp. 17–26.

[6] D. M. W. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," 2020, *arXiv:2010.16061*.

[7] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Netw.*, vol. 113, pp. 54–71, May 2019.

[8] S. Ebrahimi, F. Meier, R. Calandra, T. Darrell, and M. Rohrbach, "Adversarial continual learning," in *Proc. 16th Eur. Conf. Comput. Vis.* Glasgow, U.K.: Springer, 2020, pp. 386–402.

[9] S. I. Mirzadeh, A. Chaudhry, D. Yin, T. Nguyen, R. Pascanu, D. Gorur, and M. Farajtabar, "Architecture matters in continual learning," 2022, *arXiv:2202.00275*.

[10] D. Turner, P. J. S. Cardoso, and J. M. F. Rodrigues, "Continual learning for object classification: A modular approach," *Perception*, vol. 50, no. 1, p. 232, 2021.

[11] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss, and R. Farivar, "Towards automated machine learning: Evaluation and comparison of AutoML approaches and tools," in *Proc. IEEE 31st Int. Conf. Tools Artif. Intell. (ICTAI)*, Nov. 2019, pp. 1471–1479.

[12] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep. TR-2009, 2009.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2009, pp. 248–255.

[14] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, Jan. 2021, Art. no. 106622.

[15] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, "Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges," *Inf. Fusion*, vol. 58, pp. 52–68, Jun. 2020.

[16] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[17] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie, "The iNaturalist species classification and detection dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 8769–8778.

[18] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.

[19] Z. Cai, O. Sener, and V. Koltun, "Online continual learning with natural distribution shifts: An empirical study with visual data," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 8261–8270.

[20] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, "Overcoming catastrophic forgetting in neural networks," *Proc. Nat. Acad. Sci. USA*, vol. 114, no. 13, pp. 3521–3526, Mar. 2017.

[21] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2935–2947, Dec. 2018.

[22] F. Zenke, B. Poole, and S. Ganguli, "Continual learning through synaptic intelligence," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3987–3995.

[23] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "iCaRL: Incremental classifier and representation learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 5533–5542.

[24] R. Aljundi, M. Rohrbach, and T. Tuytelaars, "Selfless sequential learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17. [Online]. Available: https://openreview.net/forum?id=Bkxbrn0cYX

[25] A. Mallya and S. Lazebnik, "PackNet: Adding multiple tasks to a single network by iterative pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7765–7773.

[26] D. Maltoni and V. Lomonaco, "Continuous learning in single-incremental-task scenarios," *Neural Netw.*, vol. 116, pp. 56–73, Aug. 2019.

[27] L. Pellegrini, G. Graffieti, V. Lomonaco, and D. Maltoni, "Latent replay for real-time continual learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 10203–10209.

[28] D. Turner, P. J. S. Cardoso, and J. M. F. Rodrigues, "Continual learning for object classification: A modular approach," in *Proc. Int. Conf. Human-Comput. Interact.*, in LNCS, vol. 12769. Cham, Switzerland: Springer, 2021, pp. 531–547.

[29] K. Shaheen, M. A. Hanif, O. Hasan, and M. Shafique, "Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks," *J. Intell. Robotic Syst.*, vol. 105, no. 1, p. 9, May 2022.

[30] Z. Wang, L. Liu, Y. Duan, Y. Kong, and D. Tao, "Continual learning with lifelong vision transformer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 171–181.

[31] E. Arani, F. Sarfraz, and B. Zonooz, "Learning fast, learning slow: A general continual learning method based on complementary learning system," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–22. [Online]. Available: https://openreview.net/forum?id=uxxFrDwrE7Y

[32] J. S. Smith, J. Tian, S. Halbe, Y.-C. Hsu, and Z. Kira, "A closer look at rehearsal-free continual learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2023, pp. 2409–2419.

[33] Y. Yang, Z. Cui, J. Xu, C. Zhong, W.-S. Zheng, and R. Wang, "Continual learning with Bayesian model based on a fixed pre-trained feature extractor," *Visual Intell.*, vol. 1, no. 1, p. 5, 2023.

[34] L. Vorabbi, D. Maltoni, G. Borghi, and S. Santi, "Enabling on-device continual learning with binary neural networks and latent replay," *Proc. Copyright*, vol. 25, p. 36, Jan. 2024.

[35] M. Mundt, Y. Hong, I. Pliushch, and V. Ramesh, "A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning," *Neural Netw.*, vol. 160, pp. 306–336, Mar. 2023.

[36] Y. Ghunaim, A. Bibi, K. Alhamoud, M. Alfarra, H. A. A. K. Hammoud, A. Prabhu, P. H. S. Torr, and B. Ghanem, "Real-time evaluation in online continual learning: A new hope," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 11888–11897.

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

[38] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5362–5383, Aug. 2024.

[39] S. Tian, L. Li, W. Li, H. Ran, X. Ning, and P. Tiwari, "A survey on few-shot class-incremental learning," *Neural Netw.*, vol. 169, pp. 307–324, Jan. 2024.

DANIEL TURNER received the master's degree in electrical and electronic engineering, specializing in continual learning architectures for object classification. He is currently pursuing the Ph.D. degree in computer science and engineering with NOVA LINCS, Universidade do Algarve. He is also a Researcher with NOVA LINCS, Universidade do Algarve. He has authored several publications, including articles in journals and indexed conference proceedings. He has also been involved in various funded research projects in the areas of AI, HCI, HCAI, and implementing computer vision-based solutions for public safety and accessibility. His research focuses on machine learning and computer vision, with notable contributions to modular neural networks.

PEDRO J. S. CARDOSO received the B.Sc. degree in mathematics/computer science from the University of Coimbra, Portugal, the M.Sc. degree in computational mathematics from the University of Minho, Portugal, and the Ph.D. degree in mathematics/operations research from the University of Seville, Spain. He is currently a Full (Coordenador) Professor and a Researcher with the Universidade do Algarve, where he has been for over two decades. Additionally, he is an Integrated Member of the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) Research Center. He has broad knowledge in the fields of computational science, with particular emphasis in the field of machine learning and multiple objective meta-heuristics. Over the past years, he has been involved in over a dozen scientific and development projects, authored more than 80 publications, and an editor of over a dozen of books.

JOÃO M. F. RODRIGUES received the Ph.D. degree in electronic and computer engineering, in 2008, and the Habilitation degree in electrical and computer engineering, in 2021. He is currently the Vice-Rector of transfer, innovation, and digital university with the University of the Algarve (UAlg), Portugal, where he has also held positions such as the Pro-Rector of transfer and innovation and the Director of the Department of Electrical Engineering and the bachelor's degree in ICT and the master's degree in electrical and electronic engineering. Since 1994, he has been taught computer science and computer vision courses. He is a member of the Research Centre NOVA LINCS, Algarve, classified as Excellent by the Portuguese Foundation for Science and Technology. He has participated in over 30 nationally or internationally funded scientific projects, several of which he served as a coordinator. He is the co-author of more than 200 scientific publications and a member of the editorial board of several international journals. He has also organized or participated in the organization of special issues, tracks, workshops, and international scientific conferences in the fields of computer science and computer vision. His main areas of interests include computer vision, human–computer interaction, and human-centered artificial intelligence and affective computing.