University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

A Study on Prototype-Based Online CL methods

Supervisor
Prof. Nicolo Navarin
University of Padova

Co-supervisor
Giovanni Donghi
University of Padova

Master Candidate
Sejal Mansoori

Student ID
2106840

Academic Year
2024-2025

"We lose ourselves in what we read, only to return to ourselves, transformed and part of a more expansive world"
— Judith Butler

Abstract

Continual Learning (CL) seeks to overcome catastrophic forgetting, the tendency of machine learning models to lose previously acquired knowledge when trained on new data. Existing CL approaches include regularization, replay, and parameter isolation strategies. We focus on replay-based methods that rely on fixed-size memory exemplar buffers to capture evolving class representations alongside historical representations. Prototype-based methods enhance this approach by representing each class through mean embeddings (prototypes), which serve as compact exemplars for both classification and memory management.

In this thesis, we investigate the importance of replay selection and update in an online setting where the data stream is only processed once. We adopted Adaptive Prototype Feedback (APF), a sampling-based mix-up approach that predicts the likelihood of misclassification by measuring the distance between each pair of prototypes in the memory buffer, into replay-based continual learning frameworks.

We initially conducted an extensive survey of online continual learning techniques and identified replay methods with fixed-size memory buffers as the basis. We subsequently expanded on these baselines by implementing an APF-inspired technique to constantly enhance the replay samples and update class prototypes based on misclassification probabilities, moving the focus to reinforcing decision boundaries for ”confused” classes. Experiments with standard benchmarks show that replay with APF routinely improves performance by 3-5% compared to baseline approaches. These findings highlight the need to focus on the sampling technique for more robust and efficient continual learning in real-world applications.

Keywords: Continual Learning (CL), Prototype, Replay Sampling, Adaptive Prototype Feedback (APF).

Contents

Listing of figures

Listing of tables

Listing of acronyms

Symbol	Meaning
t	Current time step (or task) index
T	Total tasks
$\mathcal{D}_t$	Incoming streaming data (mini-batch) at time t
$\mathcal{M} = \lvert\mathcal{M}_r\rvert$	Replay memory buffer
$B = \lvert\mathcal{D}_t\rvert$	Mini-batch size from the stream
$m = \lvert\mathcal{B}_m\rvert$	Number of replayed samples drawn from $\mathcal{M}$
$\mathcal{B}_n = \lvert\mathcal{B}_M\rvert$	Incoming mini-batch of new data
(x_i, y_i)	Input–label pair, whether from stream or memory
p^c	Prototype for class c
N	Batch Size
$f_\theta(\cdot)$	Feature extractor / encoder network
$z_i = f_\theta(x_i)$	d-dimensional embedding
$\mathcal{Y}_t)$	Set of all classes seen up to (and including) time t
ρ_t	Dynamic Replay Ratio
$\varphi(z)$	Final classifier head (e.g. linear layer + softmax)
τ	Temperature parameter for contrastive-style losses
α	Momentum coefficient in prototype updates
β	Probability distribution
λ	Interpolation parameter

Table 1: Notations Table

Abbreviations and Acronyms	Meaning
ML	Machine Learning
CL	Continual Learning
NCM	Nearest Class Mean
NME	Nearest Mean Exemplar
KNN	k-Nearest Neighbor
ANN	Artificial Neural Network
iCarl	Incremental Classifier and Representation Learning
SupCon	Supervised Contrastive Loss
ER	Experience Replay
CoPE	Continual Evolution Prototype
PPP	Pseudo-Prototypical Proxy loss
SGD	Stochastic Gradient Descent
OnPro	Online Prototype Learning for Online Continual Learning
OPE	Online Prototype Equilibrium
APF	Adaptive Prototype Feedback
PDI	Prototype Distance Interpolation

Table 2: Abbreviations and Acronyms

1 Introduction

The expansion of the Internet has made available a wide range of data in different domains and access to a wealth of information. This abundance of data has opened opportunities for Machine Learning (ML) models to obtain remarkable results in various domains when trained efficiently. But, with so much data already available, along with new discoveries and facts being uncovered every day, what happens when we need the model to be up-to-date with the latest knowledge while also keeping previous data? It is especially challenging when the data arrive in a continuous stream, and the model has to be trained incrementally. To see its implications in real life, we can consider an instance of a recommendation system that must adapt itself according to user-specific preferences, which can constantly evolve.

As more and more information is fed into the model, it might struggle in juggling past and new information, and this might lead to outdated predictions if new data are not learned properly. Additionally, if sufficient measures are not taken during training, the parameters of old data might be overwritten by new data, thus leading to the model forgetting old information and the effect of catastrophic forgetting. Catastrophic forgetting is a phenomenon in which a model forgets the information that it previously learned when new data arrive. To address this, we need to employ a specific mechanism such that the equilibrium of maintaining old information (stability) and learning new knowledge (plasticity) can be established. Continual Learning (CL) comes as a savior that helps us to achieve the equilibrium of continuous learning of new data and retention of old data in models. In this work, CL is implemented in an online setting, such that the data arrive incrementally to the model and the model is only trained on it once, giving a realistic view of limitations in a real-life setting.

Continual learning is also called lifelong learning, as it enables ML algorithms to learn new information efficiently without losing old information. CL can come in various forms. In this work, we have focused particularly on replay-based CL, which involves storing a small subset of old data in a memory buffer. When new training is performed, we retrieve old data from the buffer and combine it with new data to facilitate efficient parameter learning. This reinforces old knowledge while learning new information. But there is also a limit to it; in today's world, data is getting generated at an exponential speed, and storing a subset of each of them is not scalable. This can make the replay-based method struggle to maintain such a high quality of replay samples in the buffer. Prototype-based learning comes as a promising solution as you represent each class with a compact prototype, denoted by the mean of each class. This learning method is especially helpful when we need to train a model on large datasets, without compromising resources such as time and memory.

In this work, we focus on the optimal selection of replay samples using replay-based methods such that the model concentrates more on data that have a higher chance of misclassification. The thesis tries to answer the shortcomings of the existing replay-based method, such as random selection of replay samples or giving equal importance to learning all features, by implementing a priority-based mechanism that can enhance the model's performance. We have taken prototype-based and raw replay methods, that is, CoPE and ER, which directly store raw samples in a buffer, and with them, we will explore possible enhancements catered to each of their specific needs. The idea of this priority-based mechanism has been inspired by the Adaptive Prototype Feedback (APF) mechanism introduced by the OnPro [1] method, which is a sampling mechanism that selects and emphasizes the learning of prototypes prone to misclassification. We tried to keep the spirit of the approach the same, but implemented it as per the architecture and needs of the individual method. In our proposed methods, we introduce the Prototype Distance Interpolation (PDI) method within the framework of CoPE's loss function. It creates mixed prototypes that emphasize confusing classes and feed them to the loss function to improve the model's adaptability so that it enforces sharp decision boundaries.

In general, this thesis explores the challenges of keeping ML models up to date with the latest information while not forgetting old data and combating catastrophic forgetting in an online setting. We have also given equal importance to ensure that the learning is done in a resource-efficient manner in terms of time and memory so that it aligns with the real-world setting.

2 Background

In this section, we will explain all the information necessary to understand the context of this research. We will begin with the fundamental idea and work our way up to more complex notions.

2.1 Machine Learning Fundamentals

Machine learning is a subdomain of AI in which we analyze how the performance of the learning model can be improved through interaction with experiences. Generally, an ML model is first trained on a set of seen data and then iteratively optimized to minimize the loss function and errors. Afterward, it is tested on unseen data for a better generalization on new inputs [2].

Machine learning can be divided into three main categories depending on data format and possible application. This includes supervised learning, where the model learns from labeled data, i.e., input-output pairs (e.g., classifying images for a given category label). In unsupervised learning, the model discovers patterns from the unlabeled data (e.g., clustering or classification). And finally, we have reinforcement learning, where the agent observes the environment and learns to make sequential decisions using a trial-and-error strategy.

Modern machine learning methods may rely on one or a combination of all of these approaches to solve real-world tasks. To solve these tasks, the ML model needs to be efficiently trained on the relevant available data. The input data can be defined as feature vectors that have been captured from the objects' characteristics, and the machine learning system analyzes these features to extract possible information. These feature vectors are represented by $x \in \mathbb{R}^n$, where the input is x_i denotes a single feature, and Y is the output label in the case of supervised learning.

During the training process, the model's weights are updated to minimize the loss function and improve its performance, usually accuracy, and this is done with the help of optimization algorithms that iteratively adjust the parameters. The most common example of optimization algorithms used is SGD (stochastic gradient descent), SGD updates parameters based on the gradient calculated from a mini-batch of training samples[2].

The scope of modern ML methods has been expanded far and wide, such that they can carry out activities in multiple domains, such as vision, language, robotics, etc., but typically with the assumption that the data source is a stationary setting. In other words, the model is primarily trained on a fixed stream of data and then deployed in a live setting. However, if we consider real life, the data arrive continuously and evolve. For example, in a recommendation system, continuous streams of user preferences data can change over time. The solution to this problem is addressed by continual learning methods, which can enable models to learn continuously from a stream of experiences without forgetting past knowledge.

2.2 Introduction to Continual Learning

Humans are gifted with the natural cognitive ability that allows them to learn new skills and knowledge with time without forgetting previous knowledge [3], but machine learning algorithms lack this ability, which limits their potential. Continual Learning (CL) is a methodology designed to overcome these shortcomings of learning from stationary data [4]. Neural networks are incrementally updated over several tasks and evolving data distributions as new data enter the network, the weights are adjusted, allowing them to obtain new information while not forgetting previously gained knowledge, so CL is also known as lifelong learning. Machine learning algorithms under incremental learning suffer greatly from forgetting due to their inability to adapt to new information. As a result, the risk of loss of previously learned information becomes more prevalent, making the model vulnerable and CL even more important. Adopting CL is necessary in many applications, such as autonomous driving, where new objects or environments may appear over time, robotics, where new skills need to be learned without forgetting old ones, and personalized systems, as user preferences change.

Additionally, it is also important to take into consideration that CL systems have to function under real-world restrictions such as finite memory and limited or no access to all of the previous data. So, we have to make CL systems operate more resource efficiently in terms of space, computational complexity, and controlled access to past data. Thus, it is better to take these restrictions into consideration at the initial stages, as the data is overgrowing due to the expansion of the internet, and we cannot simply retrain the model every time from scratch.

2.3 Catastrophic Forgetting

In incremental learning settings, once an ML algorithm has been trained and deployed, it must continue to learn new information to keep up with the outside world. As new data enters the algorithm, it loses track of old information; this is known as catastrophic forgetting. Catastrophic forgetting is a major issue in machine learning and is also the reason why we need CL [4]. In deep neural networks, parameters containing old task representations could be overwritten by new gradients if necessary precautions are not taken. For example, a standard training that minimizes loss on new data might cause the model's performance to degrade on old data, and as a result, the old parameters are replaced by new ones [5]. Continual learning attempts to mitigate catastrophic forgetting by using various strategies (such as replaying old data, constraining updates, or using specialized architectures).

Unlike natural cognitive systems such as the human brain, which can retain a wide variety of information and recall learned information when presented with favorable situations [3]. The neural network lacks this ability and, when trained sequentially on multiple tasks, succumbs to sudden and severe forgetting of previously learned knowledge.

2.3.1 Historical Identification of Catastrophic Forgetting

Michael McCloskey[5] and Roger Ratcliff[6] carried out the foundational research on the phenomenon of catastrophic forgetting in neural networks. These studies showed significant shortcomings within traditional neural network models. Ratcliff tested a series of error backpropagation models on a number of similar tasks, for vectors of different sizes, and for networks of various types, and also found that even well-learned information can be catastrophically forgotten because of learning new instances. These studies of Michael McCloskey and Ratcliff relied on *exactrecognition* to measure catastrophic forgetting. This method involves training a network on a set of binary patterns and then training on a second set of patterns. After the second training, its recognition ability on the first set was tested by presenting a sample of old input patterns. They analyzed how close the new identification was done by the model to the original associate. If the obtained output nodes were not within a range of 0.5 of the original associate, that means the network has forgotten the original pattern, i.e., it could not correctly generalize well to the original input.

2.3.2 Catastrophic Forgetting Persistence

Recent empirical studies by Goodfellow[7] further examined the severity of catastrophic forgetting with further experiments in modern neural networks when trained using various gradient-based methods. The work was particularly focused on comparing older gradient-based algorithms with newer gradient-based methods. With these experiments, he evaluated the impact of different activation functions and also analyzed the impact on the relationship between the first task and the second task due to catastrophic forgetting.

Goodfellow's findings also emphasized the problem of catastrophic forgetting. Modern neural network architectures and training techniques have somewhat reduced the problem, but it remains a substantial issue, especially when the tasks are different. To understand catastrophic forgetting[8] studied it while relating to the over-parameterization of linear regression models when trained sequentially on different input distributions. They analyzed how much the model forgets the true labels of previous tasks after being trained on subsequent new tasks to obtain exact expressions and bounds. They also quantified this measurement in terms of forgetting, i.e. how much the model forgets previously learned data, and intransigence, which is the inability of the model to update its knowledge with new data. Their findings showed that even simple linear models could experience deep forgetting when trained sequentially on tasks with different input distributions.

2.3.3 The Stability-Plasticity Dilemma

To avoid catastrophic forgetting, algorithms must focus on 2 things, first is the ability to acquire new knowledge and refine existing knowledge based on incoming input. And, second is to prevent new inputs from significantly interfering with existing knowledge[9]. The extent to which a system must be capable to learn new information and remain stable, while avoiding catastrophic forgetting and interference with learned knowledge, is known as the stability-plasticity dilemma. It has been widely studied in both biological systems and computational models[10]. Various studies have highlighted this trade-off of the stability-plasticity dilemma in CL. To effectively mitigate catastrophic forgetting, an ML algorithm must maintain a balance between plasticity and stability. Plasticity is the ability to learn and adapt to new information effectively. Stability is the ability to retain previously learned knowledge without any disruption from new learning. A key factor for machine learning systems that needs to be considered is resource expenditure when finding a balance between stability and plasticity, as the use of memory and computation must be minimized. Stability is the ability to preserve old knowledge, and storage must be optimized such that it does not take vast memory capacity, while plasticity is the ability to acquire new knowledge, and when the model is learning new data, the ML model should be computationally optimal.

2.4 Categorization of CL

Continual Learning settings can be categorized depending on how the tasks or data streams evolve. We can divide CL into 3 main paradigms based on whether the identity of the current context (task or distribution) is known at training and test time[11].

Task-Incremental Learning (Task-IL) , in which the model learns tasks sequentially, with each task distinguishable, and task labels are available during training and inference. In Task-IL, the model knows which task an input belongs to and can maintain separate outputs or parameters for each task. A separate classifier head is used per task, this simplifies CL, since the model can effectively solve each task separately at inference[12] [11].

Domain-Incremental Learning (Domain-IL), the tasks share the same output space (the same classes) but have different input distributions or contexts, and the task identity is not provided at the test time. For example, the model might see images of the same objects under different lighting or backgrounds as time passes, that is, the same task or classes are learned, but the data distribution changes over time[9].

In **Class-Incremental Learning (Class-IL)**, new classes are introduced over time, and the model does not have any label during inference to distinguish among all classes seen so far. This is often considered the hardest setting, since a single classifier must generalize over the output space as new classes arrive, and no task labels are given at test time[13].

In Task-IL, the context is known, so task-specific classifiers can be used; in Domain-IL, the context is unknown but not needed (same classes, different domains); in Class-IL, the context is unknown and must be inferred (ever-growing set of classes).

2.5 Evaluation Metrics

To evaluate the performance of the model, we utilized metrics such as average accuracy and average forgetting, which quantify how effectively the model is learning and forgetting, respectively, as usual practice in the literature. [14] [10]. Accuracy is defined as the ratio of correct predictions to total predictions. **Accuracy** for a task j after training on the task i is denoted as $a_{i,j}$ and computed as:

$$a_{i,j} = \frac{\text{\# correct predictions on task } j}{\text{total samples in task } j}$$

Average Accuracy is calculated in multiple experiences, taking accuracy from the data seen after learning all tasks. The average accuracy after training on all T tasks is defined as:

$$\text{Average Accuracy} = \frac{1}{T}\sum_{j=1}^{T} a_{T,j}$$

where $a_{T,j}$ denotes the accuracy in task j after the model has been trained sequentially on all tasks T. **Average Forgetting** is a measure of how much the performance of the model in previously learned tasks is influenced by forgetting after training in new tasks. It is determined by comparing the task's accuracy immediately after training with the accuracy after training all tasks. The average forgetting over all tasks (excluding the last) is defined as follows:

$$\text{Average Forgetting} = \frac{1}{T-1}\sum_{j=1}^{T-1} f_{T,j}$$

Where $f_{T,j}$ is the **forgetting** on task j after training on task i and is given by:

$$f_{i,j} = \max_{k\in\{1,\ldots,i-1\}} a_{k,j} - a_{i,j}$$

Where $a_{k,j}$ is the accuracy on task j after training on task k (before task i), and $a_{i,j}$ is the accuracy after training on task i. These metrics measure the performance of continual learning algorithms and ensure balanced between plasticity (learning new tasks) and stability (retaining old tasks).

2.6 Types of Continual Learning Methods

Continual learning methods are characterized into three main categories: regularization-based methods, replay-based methods, and parameter isolation methods[12] [9].

Apart from that, prototype-based methods have recently gained popularity as a potential implementation of resource-efficient CL, and can be leveraged with these approaches.

2.6.1 Regularization-Based Methods

Regularization-based approaches reduce forgetting by constraining model weight updates during new task learning by adding constraints in the loss function that penalize severe deviations from parameters crucial to identifying the prior tasks. The standard pipeline includes initializing the data, training a conventional neural network model such as ResNet, computing important parameters, and then applying a regularization term during new training, such as adjusting the loss function to charge a penalty for changing an important parameter. In this approach, we train a network to learn a new task while eliminating any significant changes in the essential weights. After each task, an evaluation of all taught tasks is performed to measure forgetting. [15]. This method is inspired by biological synaptic plasticity [16]. The most common regularization techniques are:

- **Elastic Weight Consolidation** (EWC)[17] calculates the importance of each parameter using the Fisher matrix to obtain covariance, providing information that an observable random variable x carries. As a regularization-based method, it imposes a penalty to the given anchor point, calculated using the current value of the parameters along with a weight by the Fisher information matrix, and multiplied by a scaling factor optimized for each task
- **Synaptic Intelligence** (SI)[18], tries to mimic a structure similar to the biological network of the ANNs, as biological networks naturally implement continual learning while efficiently mitigating catastrophic forgetting. In artificial neural networks, individual synapses (weights) are generally denoted by a singular scalar value, whereas biological synapses employ intricate chemical mechanisms within the brain that influence stability and plasticity across several spatial and temporal dimensions (Redondo and Morris, 2011). Synaptic intelligence studied the purpose of these internal synaptic dynamics to facilitate artificial neural networks in acquiring sequences of classification tasks. The researchers examined methods for reducing catastrophic forgetting by employing a more detailed three-dimensional state space, rather than relying solely on a scalar, one-dimensional state, which causes the network to forget prior information when learning a new task. In SI, the synaptic state of the model (junction where two neurons exchange information) monitors both historical and current parameter values and computes an "importance" metric by assessing the local contribution of each synapse to the variation in the global loss. When trained on new tasks, it prevents these important synapses from any change
- **The Learning without Forgetting** (LwF) method utilizes knowledge distillation by preserving the outputs of the previous task's model as soft targets during the training of new tasks. The model learns new information without access to the previous data by retaining its output responses.

2.6.2 Replay-Based Methods

Replay-based methods mitigates forgetting by combining samples of old data from previous tasks with the samples of new data during training. This is done by storing a small subset of data in a memory buffer and replaying it together with training on new tasks[19]. The pipeline of this method starts by initializing a model, such as a neural network, and a finite-space memory buffer. After that, we train the model on the first task as usual and store a few representative samples in memory. Then, when we train the model on a new task, we combine it for replay. For each mini-batch, mix a new sample with a reply buffer sample. This adjusts the model's weights while minimizing loss on both old and new samples. In each iteration, we update the memory buffer with new samples and then train and evaluate the model as usual by testing it on all previous data.

- **Experience Replay** (ER)[19] is a baseline replay method that maintains a memory buffer of old data samples from previous tasks and adds new task data during training. This is a simple yet effective approach to help the model retain on previous data.
- **Gradient Episodic Memory** (GEM)[14] uses a replay mechanism in a constraint-optimized manner to guide gradients towards a local task optimum.
- **Incremental Classifier and Representation Learning** (iCarl)[13] stores a limited set of old data in a memory buffer in the form of a prototype and trains the model on new data incrementally, combining with old exemplars.

2.6.3 Parameter Isolation Methods

Parameter isolation methods assign different subsets of the model parameters to different tasks while keeping the previous ones fixed to minimize interference between tasks[12].

PackNet[20] assigns different parameters to each task of the network using binary masks. It trains new tasks in two cycles- First, it trains the model on the new task without changing any parameters allocated to previous tasks, then, it removes the useless parameters. Then, in the second cycle, it retrains the remaining parameters. The mask helps protect important weights that the model has learned before by freezing the task-specific parameters for future use.

In the study of these existing methods, it has been observed that memory-replay approaches (like iCaRL, ER) tend to outperform purely regularization-based methods when a memory buffer is available.

2.7 Online Continual Learning

Continual learning can be optimized to implement with real-life constraints, where the data are ever-growing and very large. This can be done by following CL in an online setting so that ML algorithms are continuously trained on new classes from an online stream of data and the data samples are only seen once[21].

In this work, we followed data incremental learning, which is a more general paradigm of CL that allows us to learn from any data stream, with no restriction on the order in which the data are fed in an online setting. In this setting of incremental online CL, the model learns from a sequence of tasks $\mathcal{D} = \mathcal{D}_1, \ldots, \mathcal{D}_T$, where each task $\mathcal{D}_t = (x_i, y_i), i \in n$ labels, such that $y_i \in \mathcal{C}_t$, with classes $\mathcal{C}$. The model has to predict across all visible classes without any indication of task IDs during inference. Let t denote the current time step or task index, and $\mathcal{D}_t$ represent the incoming streaming of data at time t. The memory buffer is denoted by $\mathcal{M}$. B is a mini-batch from the stream, and m denotes the number of replayed samples drawn from $\mathcal{M}$. Each training sample is an input-label pair (x_i, y_i), either from the stream or the memory. The encoder network $f_\theta(\cdot)$ maps an input x_i to a normalized embedding of dimensions d such that $z_i = f_\theta(x_i)$.

2.8 Contrastive Learning

The main objective of contrastive learning is to bring similar samples together in the embedding space, and dissimilar samples should be kept further away from each other. [22].In case of absence of labels, similar samples are drawn closer by data augmentations of the target sample, while dissimilar samples are often obtained just randomly from the same batch of the target sample [23, 24, 25].In case labels are provided, i.e., self-supervised, similar samples are taken from the same class, and dissimilar samples are from different classes [26].

2.8.1 Supervised Contrastive Learning

Supervised contrastive loss (SupCon) is an enhancement of contrastive learning, which improves the efficacy of self-supervised contrastive learning by leveraging information from available labels. Unlike traditional self-supervised contrastive learning, it utilizes anchors that refers to an augmented view of an input sample, pair it with another augmented sample. Such that, a positive pair contains a sample as an anchor and another augmented anchor sample from the same class, and then SupCon contrasts each anchor with multiple positives drawn from samples belonging to the same class. This approach significantly improves the embedding space by creating more focused clusters of the same-class samples, which promotes intra-class similarity and hence is effective in separating different-class samples from each other. SupCon loss can be calculated by summing up the multiple positive pairs for each anchor outside the logarithmic function, which helps to obtain an effective gradient structure that promotes stability and accuracy during training.

$$\mathcal{L}_{\mathrm{SC}} = -\sum_{x \in \mathcal{X}} \frac{1}{|A(x)|} \sum_{x^+ \in A(x)} \log \frac{h(\varphi(x), \varphi(x^+))}{\sum_{x^- \in \mathcal{X}} h(\varphi(x), \varphi(x^-))} [26]$$

Where $h(a, b) = \exp(\mathrm{sim}(a, b)/\tau)$, $A(x)$ is the set of positive samples for x. In general, supervised contrastive loss optimizes the distinction between characteristics by encouraging positive pairs to cluster together and pushing them away from negative pairs. It has shown excellent empirical results in classification and has outperformed the standard cross-entropy loss on various benchmark datasets [23], including CIFAR-10 and CIFAR-100.

2.9 Prototype-Based Continual Learning

Prototypes can be defined as a summary of the data, mainly denoted as the mean feature vector [13]. The prototype-based method uses prototypes represented via embeddings, instead of raw images, making it efficient in terms of memory, as only the mean of the data is stored, and this property also makes it desirable in the online continual setting. This technique, when combined with the three existing CL methods, provides efficiency in terms of memory size, data representation, and overall resource saving.

2.9.1 Nearest Class Mean & Nearest Mean Exemplar

Ever-growing data has always been a major challenge in incremental learning, as it is difficult to represent evolving classes with the latest and accurate representation while not having access to complete training data. This problem was first observed in the case of finding a classifier architecture that can make space for new classes without having complete training data [13]. Mensink et al [27] investigated the use of K-nearest neighbor (KNN) and nearest class mean classifiers (NCM) to enable efficient learning of new classes. During his investigation, he observed that NCM has a unique property, that it can represent each class as a prototype vector using the average feature vector of the observed classes. But NCM had the drawback that it is difficult to extend to nonlinear data representation, as it prevents the mean vector of a class from being updated in an incremental way. Inspired by this, iCarl introduced the idea of Nearest Mean Exemplar (NME), which is a prototype-based classifier. iCarl[13] implemented a prototype representation of class samples and stored them in a fixed-size memory buffer, one per class, selected by herding for replay to prioritize examples. The main goal was to introduce a classifier robust against changing data representation, that is, the NME classifier, which represents each class by a prototype, i.e., the mean of the class, and stores a small set of representative samples of old classes in buffer storage. During training on new classes, iCaRL uses a distillation loss to retain the information from old classes and adds exemplars of new classes.

The earlier idea of prototypical networks was initially proposed for few-shot learning [28] to compute a prototype for each class and classify new samples depending on the proximity to these prototypes.

2.9.2 Prototype-Based Representation

In various state-of-the-art CL methods, prototypes are utilized for data representation and training in a space-efficient manner, as they replace raw images with compact class mean embedding. ICaRL[13], for example, adopted a prototype-based classification under the NME classifier, inspired by the nearest-class mean (NCM) classifier [27], and stored the examples with encoded prototypes (their mean) to avoid using raw data to train the classifier. As the data distribution changes, prototypes can also be updated using contrastive distance-based loss that emphasizes the separation between exemplars of dissimilar classes and brings the samples of the same class closer to each other. Thus, pulling similar sample features towards their class prototype while pushing different prototypes apart can enhance discriminability[1]. Recent studies have focused on exploiting prototype-based representation to improve computational efficiency and save memory in CL.

- **Prototype Augmentation and Self-Supervision** (PASS) does not store any old samples in memory but instead memorizes a class representation known as a prototype[28] in the feature space. Later, when the model learns a new task, each old prototype undergoes specific transformations and is then fed to the classifier head for classification. This reduces the disturbance or severe changes in parameters in the learned feature dimension and reduces the scope of classification bias due to either a new task or an old stored sample. In addition, it also reduces the chances of overfitting at the task level by adopting self-supervised learning to better generalize features on both previous and future tasks by using rotation-based label augmentation transformations[29].
- **Continual Prototype Evolution** (CoPE) [30] is an online continual prototype-based method, such that it uses a prototype vector that represents the mean of each class in memory, and when new data arrive, CoPE continuously updates these prototypes using a high momentum-based rule for each observed batch to stabilize sudden changes in the data stream. This allows the prototypes to evolve as the data distribution changes. CoPE introduces a pseudo-prototypical proximity loss (PPP) that encourages the prototypes of different classes to scatter and increases the variance between them. This keeps the samples of each class close to their prototype, reducing the variance between classes. Additionally, it also maintains a balanced replay to ensure that each class prototype receives equal representation in the memory buffer.
- **Supervised Contrastive Replay** (SCR)[26] is another prototypical online continual learning method using the same non-stationary data stream. It replaces the last layer of the softmax classifier with a NCM classifier to mitigate any recency bias and store new class representations without modifying the model architecture. SCR makes NCM more effective by explicitly shaping the embedding space using a supervised contrastive loss (Supcon)[22] to cluster embeddings from the same class together while pushing apart those of different classes. During training, the model processes each batch as a combination of current data and replay samples fed to the encoder and projection head, and optimized by contrastive loss. Then, at test time, the projection head is removed during classification, and NCM is used to classify class prototypes that are computed from memory.
- **Online Prototype for Online Continual Learning** (OnPro) [1] also prototype representation for the data and introduce online prototype equilibrium loss for better learning of representative and discriminative features. Furthermore, they also introduced a novel sampling strategy called adaptive prototypical feedback mechanism that utilizes the feedback from online prototypes to focus on classes that are prone to misclassifications and enhance their boundaries by giving them priority in the buffer.

Overall, prototype-based methods provide better class-centric representation to the data, and by combining the prototypes representation with existing CL methods, we can improve their resilience to forgetting.

3
Related Works

Following our discussion on continual learning in the background section, we saw many CL methods and their types, such as regularization-based, replay-based, and parameter isolation-based[10]. In this section, we will study replay-based CL methods in detail that have been the key focus of our research.

3.1 Replay-Based Methods

3.1.1 Experience Replay

Experience Replay (ER) [19] is one of the baseline replay-based strategies. It keeps a fixed-size memory replay buffer to store past exemplars and then uses the data from the buffer to continuously train the model with the samples of old data that get replayed along with the new data to mitigate catastrophic forgetting. The typical implementation of Experience Replay (ER) is described below.

In Algorithm 3.1, we can see the pseudocode for ER. It maintains a memory buffer $\mathcal{M}$ of fixed size *mem_sz* to maintain past examples. For the first batch of data, we train that model following the traditional ML pipeline, and the replay buffer is empty.

After training, we store a few samples of these data in the memory buffer $\mathcal{M}$. Then, as new data arrives, it takes the current mini-batch B_m along with a mini-batch $B_{\mathcal{M}}$ of the old data from the memory buffer. It merges the new data B_t and the old data M_t for training. The model then performs a standard optimization step using SGD in the combined batch and refreshes the buffer with items from B_m using reservoir sampling to maintain the unbiasedness of $\mathcal{M}$.

In this way, the model relearns previous classes alongside the new ones, obtaining optimized weights that are relevant to the current data distribution. If the buffer is full, it clears the buffer by removing some old samples to make space for new ones. To manage incoming and previous data in the memory buffer, ER uses reservoir

Algorithm 3.1 Experience Learning for Continual Learning

```
Procedure ER $D$, $mem\_sz$, $batch\_sz$, $lr$

$\mathcal{M} \leftarrow \{\}$ of size $mem\_sz$                    ▷ Allocate memory buffer

$m \leftarrow 0$                                 ▷ Number of training examples seen

for $t \in \{1, ..., T\}$

    for each mini-batch $B_m^K \subset D_t$

        Sample $B_{\mathcal{M}}^K \subset \mathcal{M}$

        $\theta_k \leftarrow SGD(B_m \cup B_{\mathcal{M}}, \theta, lr)$    ▷ Gradient step with current and memory batch

        Update memory $\mathcal{M}$

        $n \leftarrow n + |B_m^K|$                       ▷ Update counter

    end for
end for
return $\theta, \mathcal{M}$
```

sampling. With reservoir sampling, it decides which examples to select by giving each incoming example a chance to enter the memory, ensuring an approximately uniform sample distribution of the data stream in the long run.

This method is very simple, yet efficient. ER runs a forward-backward pass on a batch of size $B_{\mathcal{M}} + B_m$. Then, on stream N, the number of updates is $\approx N/B_m$. So, the computational complexity would be $O(N(1+B_{\mathcal{M}}/B_m)$, which is not very expensive. The performance of ER tends to improve as the memory size grows, but given a fixed memory, there is a trade-off in how exemplars can represent all past classes. ER does not inherently ensure class balance in the memory; the memory may become imbalanced, which can bias replay.

ER typically uses the standard model's softmax classifier for prediction and does not maintain prototypes or separate classification mechanisms for past classes. The fully connected final layer of the model can continually expand to accommodate new classes, and replay training allows it to discriminate among all classes seen so far. Because ER does not freeze any parameters, the model retains high plasticity and fast learning of new tasks, but without additional measures.

Experience Replays is a short and straightforward approach that overcomes forgetting by simply replaying previous data. It does not require task boundaries or ID labels, making it ideal for task-free continual learning. However, it has several limitations, such as the use of random sampling, which may not select the best or ideal sample to focus the model's ability where it is most needed. It does not use any unique method that prioritizes samples prone to misclassification and treats all exemplars equally.

3.1.2 Continual Prototype Evolution

Continual Prototype Evolution (CoPE) is an online class-incremental learning approach that implements class prototypes [30] in the replay process. CoPE does this by employing a set of prototypical representations for each class and updates them continuously as new data arrives. Storing prototypes eliminates the need to store raw samples in the memory, which shifts the catastrophic forgetting problem from the fully connected neural network parameter space to the lower-dimensional latent space. The prototypes are represented by the mean feature vectors of each class, which evolve incrementally as a new batch arrives for training. Furthermore, CoPE uses a high-momentum update rule per batch to update the prototype so that the old representation does not suffer major changes. It is used batch-by-batch and not the entire dataset to make sure the change is steady and the representation of previous knowledge is maintained.

In the CoPE training process, each new mini-batch B_n of incoming data is augmented by a replay batch B_m drawn from the memory buffer $\mathcal{M}$ (with $|B_M| = |B_n|$). The total batch $B = B_n \cup B_m$ is fed through the embedding network f_θ to obtain latent representations. The prototypes p^c are then updated using the features of batch B via a momentum update,

$$p^c \leftarrow \alpha p^c + (1 - \alpha)\bar{p}^c$$

where p^c denotes the prototype for class c, i.e. the mean of the features for class c in latent space, and $\bar{\mathbf{p}}^c$ denotes the center of mass for the current batch in the latent space. After the update, p_c it also has to be $l2-$normalized as it is no longer of unit length due to the triangle inequality.

A momentum factor $alpha \approx 0.99$ is used to stabilize the prototypes during update, so that they evolve slowly with each batch while addressing the non-stationary nature of the data stream and avoiding sudden large updates. If a class is not present in a particular batch, its prototype remains unchanged in that iteration, and only the classes that appear in B are updated. This continuous updating ensures that prototypes represent the current feature distribution of each class, where the model learns new classes and refines old ones.

For classification, CoPE uses a nearest-neighbor classifier in the latent space. Given an input x_i with a characteristic $f_\theta(x_i)$, CoPE predicts the class whose prototype p_c has the highest dot product using cosine similarity after $l2$ normalization with $f_\theta(x_i)$. This strategy is similar to the Nearest Class Mean classifier used in iCaRL[13], but CoPE continuously updates the prototypes.

When CoPE updates the prototype with a new approximation of the mean of the parent distribution, it is optimal with Bregman divergences, where the cluster mean minimizes the distance to its population. The Bregman divergence can be defined as,

$$d_\psi(f_i, f_j) = \psi(f_j) - \psi(f_i) - (f_j - f_i)^T \nabla\psi(f_j)[19], [30]$$

where ψ is a differentiable and strictly convex function. For $\psi(f) = ||f||^2$ (squared Euclidean norm), this can be simplified to the squared Euclidean distance:

$$d_\psi(f_i, f_j) = ||f_i - f_j||^2.$$

CoPE also mentions that for vectors normalized to unit length ($||f_i|| = ||f_j|| = 1$), the squared Euclidean distance is proportional to the cosine distance:

$$\frac{1}{2}\|f_i - f_j\|^2 = 1 - \cos\angle(f_i, f_j).$$

This equivalence highlights that, in a normalized space (unit length), the geometric relationship between vectors can be expressed with both the Euclidean distance and the cosine similarity. Cosine similarity is preferred when we need the orientation as the vectors are constrained to the unit hypersphere.

To train the embedding network f_θ, CoPE introduced a Pseudo-Prototypical Proxy loss (PPP-loss). This loss has two main goals:

- To pull the representation of similar samples towards its class prototype, therefore, reducing intra-class variance.

- To push the representation of different samples away from other class prototypes and increase the margin between classes.

CoPE uses PPP loss as an enhancement of contrastive loss, where each true prototype p^c denotes an anchor point for all samples in a class c. The loss encourages $f_\theta(x_i^c)$ being close to p^c and far from $p^{c'}$ ($c' \neq c$). This is done using the supervision signal y_i in a sample $(x_i, y_i) \in B$ to show which class x_i is in and to make a clear difference between positive and negative pairs in B.

All of this can be put as a one-to-one classification of subsets for an instance of class c, with positives from the same class in $B^c = \{(x_i, y_i) \in B\}$ and negatives in B^k. These sets are then used to make attractor and repeller sets for an instance x_i of class prototype p^c and the other instances of B.

Initially, the attractor set is defined by using other instances of the class c as pseudo-prototypes as:

$$P_i^c = \{p^c\} \cup \{\hat{p}_j^c = f_\theta(x_j) \mid x_j \in B^c, j \neq i\}.$$

Then, the samples of other classes, such as $x_j^k \in B^k$, avoiding both the example x_i in the latent space and its class representative p^c, which is encoded in the repeller set as

$$U_i^c = \{p^c, \hat{p}_i^c = f_\theta(x_i)\}.$$

The attractor P_i^c for x_i decreases the variance in the intraclass (with $1 \leq |P_i^c| \leq |B^c|$), while x_i is when the repeller U_i^c exploits the variance in the interclass between prototypes of other classes.

We can put attractors and relays set in one loss as a binary classification problem with a joint probability, such that x_i^c is predicted in class c and all $x_j^k \in B^k$ are *not* predicted in class c, such that:

$$P_i = P(c \mid x_i^c) \prod_{x_j^k \in B^k} \left[1 - P(c \mid x_j^k)\right]. \tag{3.1}$$

It is assumed that x_i^c and each x_j^k are independent of each other, and the expected posterior probabilities over attractor and repeller sets are defined as

$$P(c \mid x_i^c) = \mathbb{E}_{\tilde{p}^c \in P_i^c}\left[P(c \mid f_\theta^i, \tilde{p}^c)\right], \tag{3.2}$$

$$P_i(c \mid x_j^k) = \mathbb{E}_{\tilde{p}^c \in U_i^c}\left[P(c \mid f_\theta^k, \tilde{p}^c)\right], \tag{3.3}$$

with $\tilde{p}^c$ is a proxy for the latent mean of class c in the embedding space.

$$P(c \mid f, \tilde{p}^c) = \frac{\exp\left(f^\top \tilde{p}^c / \tau\right)}{\exp\left(f^\top \tilde{p}^c / \tau\right) + \sum_{k \neq c} \exp\left(f^\top p^k / \tau\right)}. \tag{3.4}$$

with temperature τ manages the concentration level of the distribution, presuming a cosine similarity metric ${f_\theta^i}^\top f_\theta^j$ with vectors normalized to unit length. Equation (3.1) is reformulated as the loss function $\mathcal{L}$ by utilizing the negative log-likelihood and subsequently summing it across all instances in B, resulting in the accurate joint probability under the condition of independence between pairs within the batch:

$$\mathcal{L} = -\frac{1}{|B|}\left[\sum_i \log P(c \mid x_i^c) \; + \; \sum_i \sum_{x_j^k \in B^k} \log\left(1 - P_i(c \mid x_j^k)\right)\right]. \tag{3.5}$$

Another important aspect of CoPE is its memory management and replay sampling strategy. Unlike ER, which used random reservoir sampling, CoPE adopts a balanced memory approach. The total memory M is divided equally among all the classes observed so far, and every time a new class is observed, an equal memory quota is allocated, and the quota allocated to other classes can also be reduced. This per-class allocation idea ensures that no class is completely erased from memory due to class imbalance in the stream. For each training step, CoPE runs a forward + backward pass of $O(B_M + B_{\mathcal{M}})$ and optimizes the model with PPP loss. The PPP loss calculates batch-batch cosine similarity for pseudo prototypes and updates the prototype with a momentum-based rule, along with performing L_2 Normalization. With all these operations, CoPE complexity becomes quadratic to batch size, as building a full batch similarity matrix in PPP loss is a heavy operation.

CoPE tries to create this optimal balance between the replay method and representation learning. The prototypes act as a connection between old and new data, and replay ensures that the embedding does not drift too far. Thus, older prototypes remain valid, and the prototypes provide a reference that guides learning new representations. However, there is an open gap here, by treating each class prototype as equal, it might struggle in dealing with classes that are prone to misclassification and thus need more attention. This is further addressed in OnPro.

3.1.3 OnPro

Online Prototype Learning (OnPro) is a state-of-the-art replay approach that introduces further development in the domain of prototype-based CL. It mitigates several shortcomings identified in previous CL approaches discussed in the preceding section. The primary objective of OnPro is to address ”shortcut learning” in online continual learning, wherein the model emphasizes learning oversimplified or biased features that work well for current data, but do not generalize well to new classes. And, it is necessary to target shortcut learning, otherwise, the assumption that the preserved features are valid, made by various techniques, will become invalid and will lead to error propagation to new classes, as the features are not discriminative. Therefore, OnPro emphasizes on learning features that are both representative of each class and also establish clear discrimination with other classes. To establish this using two key components, Online Prototype Equilibrium (OPE) and Adaptive Prototypical Feedback (APF) are introduced.

Online Prototype Equilibrium (OPE) is a component of the loss function, introduced to encourage equilibrium between all the seen classes and ensure they are well-separated in the feature space. It focuses on their discriminability; when new classes are learned, it keeps the prototype up to date, while pushing it toward the same class, thus avoiding accumulating any irrelevant information. Similar to CoPE, OnPro leverages the online prototype for each class, they are defined within each mini-batch. An online prototype p_i^c is defined as the mean of this batch's embeddings for the class c. For each time step iteration i of each class c, the current batch includes both the new data and the sampled replay data. Learning new information while addressing catastrophic forgetting is crucial. Hence, considering these, OnPro attempts to learn representative features of each class by pulling the prototypes P and their augmented views P' closer in the embedding space, and learn discriminative features between classes by pushing prototypes of different classes away, formally defined as a contrastive loss:

$$L_{\text{pro}}(P, P') = \frac{1}{2\,|P|} \sum_{i=1}^{|P|} \left(-\log \frac{\exp\left(\langle p_i, p_i' \rangle / \tau\right)}{\sum_{j=1}^{|P'|} \exp\left(\langle p_i, p_j' \rangle / \tau\right) + \sum_{\substack{j=1 \\ j \neq i}}^{|P|} \exp\left(\langle p_i, p_j \rangle / \tau\right)} \right) \tag{3.6}$$

By treating all classes equally in this equilibrium objective, OPE eliminates any bias toward new classes. As new classes are added, the equilibrium shifts to accommodate them, maintaining separation among all classes, both past and new. This reduces catastrophic forgetting because the representation of old classes is adjusted appropriately to remain discriminative as new classes appear. OPE functions as a zero-sum game between existing and new knowledge, aiming for an optimized equilibrium.

Adaptive Prototypical Feedback (APF) is introduced to balance the assumption made by OPE by treating all the classes uniformly, as not all classes are equally difficult to learn and prone to forgetting. Some classes become highly confused with others, like when they are similar or if the model hasn't seen them in a while. APF is a sampling strategy introduced to adaptively focus the replay on these 'hard to learn' classes by leveraging feedback from the prototypes. After each training iteration, APF looks at the current distribution of the prototypes for all classes and measures pairwise distances between them. If prototypes of any two classes are very close to each other, it means that the model can get confused between them, as their decision boundary is not very well separated. APF converts the prototype distance matrix into a probability distribution P over the classes and then applies a Gaussian kernel defined below to obtain a probability distribution. So, to identify the classes with higher probabilities of misclassification:

$$P_{i,j} \propto \exp(-\|p_i^b - p_j^b\|^2), \quad i,j \in \{1, \ldots, |P^b|\},\ i \neq j. \tag{3.7}$$

where $i,j \in \{1, \ldots, |P^b|\}$ and $i \neq j$. All probabilities are normalized to form a proper probability mass function that guides the sum to be 1. APF returns these $P_{i,j}$ values to M for guiding the next sampling process and sharpen the decision boundaries of easily misclassified classes.

The probability P_c for a class c guides us to see if the class c is in short distance with one or more pairs of prototypes, indicating that it can be easily misclassified. This distribution P thus counters the bias in the selection of the replay samples in the next iteration. Specifically, instead of randomly sampling memory M for replay, OnPro employs a two-stage sampling approach. A fraction of the replay batch is drawn according to P (get examples from confusing classes), and the remaining examples are drawn uniformly from the memory to maintain diversity. Moreover, APF also applies mixup augmentation to these selected samples by interpolating pairs of examples to create synthetic samples, thereby further smoothing decision boundaries between confused classes by providing intermediate samples.

At each Step, OnPro trains with batch B_m and replay $B_{\mathcal{M}}$, such that it calculates two augmented views for every sample, encodes with the encoder + projection head, and forms online prototypes per class. It then optimizes with three terms: OPE pulls the same-class online prototypes together and pushes different classes apart for both new and seen classes. It then applies supervised contrastive loss over all pairs for intra-class compactness and a cross-entropy term on replay for classification. After this, we apply APF by measuring the distance between the last step's class prototype and turning those distances into a probability matrix, and use it to sample classes that are prone to misclassification and mix them up with the next replay batch. Finally, update the memory uniformly and move to the next step. The computational complexity of OnPro can be concluded as quadratic overall, but it performs more operations in between than CoPE, thus needing more time.

Overall, OnPro offers an ideal solution to overcome shortcut learning and eliminate bias by enforcing a clear distinction between features with a strong decision boundary. This is done by employing refinements at different levels in the replay data as well as the feature representation. It highlights other factors that need to be considered, which are computational time and complexity of the algorithm. OnPro delivers an exceptional performance as state-of-the-art in online continual learning, but it also comes at a cost of large computational time and complexity. Therefore, thinking of the real-world environment of online learning, where learning has to be sufficiently fast and space-efficient, we tried taking inspiration from the innovative functions introduced by OnPro. In this work, we will try to devise an implementation with existing methods inspired by OnPro that can improve performance while balancing computation time and complexity.

4 Methodology

This chapter is dedicated to introducing the enhancements proposed in existing techniques. We implemented a priority-based mechanism into the existing CL methods to improve the model's focus on classes that are prone to misclassification while remaining resource efficient. To develop such an approach, we took inspiration from the OnPro adaptive prototypical feedback (APF) sampling technique and applied it to the Experience Replay (ER) baseline and the prototype-based CoPE method. The idea is to combine the resource efficiency of these techniques with APF-inspired prioritization mechanisms to improve boundary sharpening and target replay.

4.1 Enhanced ER

In this section, we will look at how the ER approach could be combined with a priority-based sampling strategy to improve its performance. In the related work section 3.2.1, we explored ER and found it to be a replay-based CL method that stores past instances in a fixed-size memory replay buffer. The model therefore observes previous instances as well as current examples while training on new data, preventing catastrophic forgetting.

In this approach, we follow the standard online continual learning environment, such that the model learns from a sequence of tasks $\mathcal{D} = \mathcal{D}_1, \ldots, \mathcal{D}_T$, where each task $\mathcal{D}_t = (x_i, y_i), i \in n$ s.t. n consists of the labeled samples $y_i \in \mathcal{C}_t$ class. The model has to predict across all visible classes without using task IDs during inference. As we saw in section 3.1.1, ER maintains a memory bank $\mathcal{M}$ of all past samples to avoid forgetting and then replaying them alongside new data during training. However, the standard ER randomly selects samples from memory $\mathcal{M}$, which is vague and does not prioritize classes prone to confusion, limiting its efficacy against forgetting and resulting in poor generalization of new features.

In Section 3.2.3, we discuss OnPro's innovative sampling technique, APF, which prioritizes classes that are prone to misclassification by computing the probability distribution. After studying both of these methods in depth, we propose a possible enhancement of ER by integrating an APF-inspired sampling strategy into it. The proposed method adheres to the same architecture as ER, making enhancements during sample selection and replay. To select confusing samples, the method employs a priority-based sampling technique that replaces random sampling; during replay, it adaptively prioritizes both the replayed and the new data.

During the construction of this method, it is also considered that the ER directly stores the replay samples, and thus the need for prototype computation is eliminated. The priority function is defined by computing forgetting scores based on class accuracies that highlight class confusion. The integrated selective playback of samples prioritizes the classes with a higher forgetting score. For each class c, we define a forgetting score s_c and set it to 0.90 at the beginning, and as new experiences are learned, it is updated based via:

$$s_c = \frac{1}{\mathrm{A}_c + \varepsilon}, \quad \varepsilon = 1 \times 10^{-6}, \tag{4.1}$$

Here, A_c is the accuracy of the class c on the test stream, and ε prevents the division to zero. After computation, scores are normalized to form a probability distribution on the memory bank samples. After obtaining the forgetting scores, the replay samples are selected via a two-stage process before each training iteration:

1. **Prioritized Sampling**: A fraction $\alpha = 0.25$ of the replay samples is sampled from confusing classes with higher forgetting scores, i.e., lower accuracy.

2. **Uniform Sampling**: The remaining samples are selected uniformly from $\mathcal{M}$ to ensure equilibrium and diversity to avoid excessive emphasis on confused classes, preserving the equilibrium of learned representations.

After training in a task $t \in \mathcal{D}_t$, a subset of samples is added to the memory buffer $\mathcal{M}$. The forgetting scores, s_c, are then used to select the replay samples that are more likely to be forgotten. The selected replay samples are then concatenated with the current mini-batch as usual and fed to the model for training. For this concatenation, we use a dynamic replay ratio to balance learning in new and old classes. The dynamic replay ratio ρ_t determines the number of replay samples m to be included with the current mini-batch for each task t. It is defined below as

$$\rho_t = \rho_{\text{init}} - \left((\rho_{\text{init}} - \rho_{\text{final}}) \cdot \frac{t-1}{T-1} \right), \quad \rho_t \geq \rho_{\text{final}}. \tag{4.2}$$

Where ρ_{init} stands for the initial replay ratio, it is the proportion given to the replay samples at the beginning of training from task $t = 1$. It is set to 0.8 to ensure that more importance is given to the replay samples in the initial stage of training, as few classes have been seen. ρ_{final} stands for the final replay ratio, which is the proportion of replay samples to be used at the end of training on the last task t. It is set to 0.6 to reduce the number of replay samples later, allowing the model to concentrate more on new classes as the number of classes seen ($|\mathcal{Y}_t|$) increases. ρ_{init} and ρ_{final} together create a dynamic replay ratio that focuses on the balance of learning between the new and old classes. A fixed replay ratio may either overemphasize new data, leading to forgetting, or overemphasize replay data, hindering adaptation to new tasks. The dynamic replay ratio ρ_t addresses this problem by adjusting the proportion of replay samples m relative to new samples $\mathcal{D}_t$ in each task t. It ensures sufficient passes of past samples in the training and allows more focus to be drawn on new classes as the number of tasks increases. With a fixed replay ratio,

- If low, the model might suffer catastrophic forgetting due to disproportional or less focus on old data.
- Similarly, if too high, the model's ability to adapt to new tasks is compromised as the replay is prioritized more.

Thus, the enhanced ER varies this ratio and ensures that early tasks prioritize the replay of previous knowledge. At the same time, in later tasks, more weight is allocated to new classes as the classification complexity increases with more seen classes.

4.2 Vanilla CoPE + APF

In this section, we will focus on the enhancement proposed on the prototype replay-based CoPE method in the data-incremental learning setting. As discussed in section 3.1.2, the existing CoPE method mitigates catastrophic forgetting by maintaining prototypes $\mathbf{p}^c$ for each class $c \in \mathcal{Y}_t$ in a latent space, and they represent the mean feature vector for each class. It utilizes pseudo-prototypical proxy (PPP) loss to reduce intraclass variance and enhance interclass variance through attractor and repeller sets. As a replay-based approach, it stores these prototypes in a memory buffer and selects them through uniform sampling, ensuring that every class is equally represented in the buffer. Although this works well in an ideal situation, in real life, not all classes are equally easy or difficult to learn. Some classes are more prone to misclassification than others, so the learner pays more attention to these classes.

We tried to overcome this limitation by integrating CoPE with the APF-inspired sampling mechanism so that memory samples with a higher risk of forgetting are prioritized. To implement such a method in CoPE, we first calculate the risk of forgetting for all samples in the memory buffer M following the formula.

$$\text{risk}_i = 1 - \cos(f_\theta(x_i), p_{y_i}) \tag{4.3}$$

where $f_\theta(x_i)$ is the normalized embedding of a memory sample x_i , p_{y_i} is the prototype of its class y_i, and cos() is the cosine similarity. Risk refers to the degree of alignment of a class prototype with the sample distribution of its class. A higher risk score indicates that the embedding of the sample y_i is poorly aligned with its class prototype and thus carries a higher risk of misclassification.

In this implementation, we used cosine similarity instead of distance as in OnPro because in the CoPE method, vectors are constrained to the unit hypersphere. In this case, the Euclidean distance and cosine similarity are directly proportional; this has been established by the Bergman divergences [30]. The cosine similarity is more aligned with the CoPE's training loss, which requires a similarity measure, and it is numerically stable while avoiding any squaring/root operation. We know that the prototypes $\mathbf{p}^c$ in CoPE are initialized for each new class $c \in \mathcal{C}_t$ and updated using a momentum-based rule:

$$\mathbf{p}^c \leftarrow \alpha \mathbf{p}^c + (1-\alpha)\overline{\mathbf{p}}^c, \quad \overline{\mathbf{p}}^c = \frac{1}{|B^c|}\sum_{x_i \in B^c} f_\theta(x_i) \tag{4.4}$$

where $\overline{\mathbf{p}}^c$ corresponds to the center of mass in the latent space for the current batch. Due to the triangle inequality, p_c is no longer unit length, and thus, after each update $\mathbf{p}^c$ is ℓ_2 normalized to uniform prototype evolution against stochastic batch sampling on the sample embeddings and class prototypes to make them lie on the unit hyperspace (i.e. $\|f(x_i)\|_2 = 1$, $\|p_c\|_2 = 1$)[30].

After calculating the risk through equation Equation 4.3, we obtained how well the prototype type aligns to the distribution, and now calculate weights based on the risk scores to find the sampling probability such that-

$$w_i = \frac{\exp(\text{risk}_i/\tau)}{\sum_{j=1}^{M} \exp(\text{risk}_j/\tau)}, \tag{4.5}$$

w_i represents the sampling probability for the sample i. Lowering the temperature τ sharpens the distribution, prioritizes samples with high risk, and allows for diversity. After computing the weights, we use them to calculate sampling probabilities and select memory samples by weight-based sampling. This ensures that samples with higher risk scores are more likely to be included in the training batch, while also focusing on replay samples that are crucial for distinguishing between prototypes.

The new framework follows the same processes as CoPE, so that.

1. For a batch $B = \{(x_i, y_i)\}_{i=1}^{|B|}$, extract the target labels $\{y_i\}$ and initialize the memory buffer M to store the replay samples.

2. We train the model as usual on the first task, and compute initial prototypes and store them as a tensor in the memory buffer uniformly.

3. For each sample in the buffer, we calculate a forgetting risk, $\text{risk}_i = 1 - \cos(f(x_i), p_{y_i})$, where p_{y_i} is the prototype for the sample class c. Then, compute the sampling weights using a softmax over the risk scores defined in Equation (3.5). It uses APF-inspired sampling to choose memory samples based on these weights, prioritizing high-risk samples that are likely to be forgotten, and then generates a memory data loader with these samples.

4. For each iteration i, samples are selected from the replay buffer and combined with the current data to further train the model.

5. The model is trained on the combined data to obtain normalized embeddings, and then calculates the PPP loss as usual, and uses an optimization algorithm to update model parameters and refine the decision boundary.

6. To update prototypes, use the momentum-based rule defined in Equation (3.4) for each class in the batch.

7. After training on the current task, evaluate the performance of the model in the test stream with no task identifiers.

We iterate this cycle for all tasks, with an APF-inspired replay that recomputes risk scores and sampling weights for each batch, ensuring that the memory replay focuses on high-risk samples throughout the training. In summary, this APF variant utilizes a risk-based sampling method that prioritizes memory samples based on the cosine similarity of their embeddings to the relevant class prototypes. Samples with lower similarity, indicating high risk, are given a larger sampling probability using a softmax weighting.

4.2.1 Enhanced CoPE + PDI

In Vanilla Cope with APF, we elaborated on the implementation of a priority-based sampling strategy to select the classes prone to misclassification in the baseline CoPE method. In this enhanced method, the memory buffer loading and sampling strategy remained the same as in the original CoPE, but now the focus is on updating the prototypes. We modify the part where prototypes are given to the loss function PPP to define the decision boundary. CoPE uses PPP loss, which is an enhancement of contrastive loss, and the main components of this loss function are the Attractor and Repellor sets (Section 2.2.2), which pull similar prototypes together and push others away. But the limitation here is that the repeller term pushes all embeddings that are not of the current class away equally, and this does not give special focus to confusing classes, leading to suboptimal boundaries for similar classes.

Thus, the proposed enhanced CoPE + PDI approach attempts to overcome this limitation by **interpolating class prototypes** by distance, before adding the embeddings to the loss function to force the decision boundary to a hard margin to separate confusing classes. **Prototype Distance Interpolation (PDI)** can be described as the process of modifying class prototypes by interpolating them with prototypes of other classes via a distance-based probability distribution to guide the selection of the secondary prototype. This mechanism is inspired by OnPro adaptive prototypical feedback (APF) and aims to strengthen decision boundaries in CoPE by making prototypes more robust to class confusion. We will be using a fixed size memory $\mathcal{M}$ to store prototypes $\mathbf{p}^c$ for each $c \in \mathcal{Y}_t$. After training in the task $\mathcal{D}_t$, a class-balanced subset of samples is added to $\mathcal{M}_r$. The prototypes $\mathbf{p}^c$ are initialized as usual for each new class $c \in \mathcal{C}_t$ and updated using a momentum-based rule defined in Equation (3.4).

PDI creates a temporary prototype for a given class by linearly combining the current prototype $\mathbf{p}^c$ with a secondary prototype $\mathbf{p}^s$ of another class, and this secondary class s is selected based on the Euclidean distance between the prototypes. To calculate these temporary prototypes $\mathbf{p}_{\text{mixed}}$, we use $\mathbf{p}_{\text{mixed}} = \text{normalize}\left((1-\lambda)\mathbf{p}^c + \lambda\mathbf{p}^s\right)$. Here, the interpolation is weighted by a scalar term λ sampled from the *Beta* distribution and normalized to map them back onto the unit hypersphere. The purpose of λ is to control the interpolation between the primary prototype ($\mathbf{p}^c$) and the secondary prototype ($\mathbf{p}^s$) of a confusing class. *Beta* The distribution is a continuous probability distribution defined in the interval $[0, 1]$, used for the interpolation parameter λ. λ It determines to what extent the confusing class influences the loss computation. When $\lambda \approx 0$, the prototype aligns more with the true class. When $\lambda \approx 1$, it aligns more with the confusing class **s**, it forces the model to distinguish c between the confusing class s and improves the decision boundary. Sampling λ randomly from $\text{Beta}(0.35, 0.35)$ introduces variability, ensuring that the model sees a range of interpolated prototypes, enhancing robustness. The selection probability prioritizes classes with prototypes that are closer to $\mathbf{p}^c$, as these are more likely to be confused. This process adjusts the prototypes to lie closer to the decision boundary between classes. This eventually makes the model focus more on distinguishing them during nearest-neighbor classification, hence improving decision boundaries without modifying prototypes directly. Then, $\mathbf{p}_{\text{mixed}}$ is fed to the loss function to make it focus on separating c from s by giving them a hard margin and thus sharpening the decision boundary.

In the original work of OnPro, APF uses prototype distances to prioritize replay sampling to focus on confusing classes, and we implemented this idea in Vanilla Cope+APF. In Enhanced CoPE+PDI, this time, instead of playing with the memory buffer, we tried to use PDI in the loss function, not sampling and aligning with the CoPE prototype-based classification. The general picture can be painted in Algorithm 4.1.

In Algorithm 4.1, the training loop iterates over T tasks, processing each task $\mathcal{D}_t$ in a single epoch as intended in an online setting. For each task $t \in T$:

1. **New Data Batch (B_n)**: From the current stream of data, we sample a mini-batch $B_n \subset \mathcal{D}_t$, containing pairs (x_i, y_i), where $y_i \in \mathcal{C}_t$.

2. **Replay Batch ($B_\mathcal{M}$)**: If $\mathcal{M}_r \neq \{\}$, that is, the memory buffer is not empty after task 1, and uniformly sample $|B_\mathcal{M}|$ samples from $\mathcal{M}_r$ to ensure balanced rehearsal just as CoPE.

3. **Forward Pass**: Train the model with combined data and compute embeddings for the batch B,

$$\mathbf{f}_i = f_\theta(x_i)/\|f_\theta(x_i)\|_2, \quad \forall (x_i, y_i) \in B,$$

where f_θ is the slim ResNet18, and $\mathbf{f}_i \in \mathbb{R}^{128}$ is the ℓ_2-normalized embedding.

4. **APF-inspired Prototype Update**: After the forward pass, we apply prototype interpolation to modify

Algorithm 4.1 Enhanced CoPE with PDI

```
1: **Input:** Tasks $T$, data $\mathcal{D}_t\ \forall t \in T$, memory $\mathcal{M}_r$, learning rate $\eta$, $\sigma = 0.35$
2: **Output:** Model parameters $\theta$, prototypes $\{\mathbf{p}_c \mid c \in \mathcal{Y}_t\}$
3: **Initialize:** Model $f_\theta$ (ResNet18), $\mathcal{M}_r \leftarrow \emptyset$, $\mathbf{p}_c\ \forall c \in \mathcal{Y}_t$, $\mathcal{Y}_t \leftarrow$ seen classes
4: **for** $t \in \mathcal{T}$
5:     **for** mini-batch $\mathcal{B}_n \subset \mathcal{D}_t$, pairs $(\mathbf{x}_i, y_i), y_i \in \mathcal{C}_t$
6:         $\mathcal{B}_M \leftarrow$ sample from $\mathcal{M}_r$ if $\mathcal{M}_r \neq \emptyset$
7:         $\mathcal{B} \leftarrow \mathcal{B}_n \cup \mathcal{B}_M$
8:         Compute embeddings: $\mathbf{f}_i = \frac{f_\theta(\mathbf{x}_i)}{\|\mathbf{f}_\theta(\mathbf{x}_i)\|_2}$, $\forall(\mathbf{x}_i, y_i) \in \mathcal{B}$
9:         **for** $y_i \in \mathcal{B}$
10:            Get $\mathbf{p}_c$ for $c = y_i$, nearest $s \neq c$ from $\mathcal{Y}_t$, and $\mathbf{p}_s$
11:            Sample $\lambda \sim \text{Beta}(\sigma, \sigma)$
12:            $\mathbf{p}_{\text{mixed}} = \text{normalize}((1-\lambda)\mathbf{p}_c + \lambda\mathbf{p}_s)$
13:        **end for**
14:        Compute PPP loss $\mathcal{L}$ with $\mathbf{p}_{\text{mixed}}$, update $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}$
15:        Update $\mathbf{p}_c$ for classes in $\mathcal{B}$ using the momentum-based rule, $\mathbf{p}^c \leftarrow \alpha\mathbf{p}^c + (1-\alpha)$
16:    **end for**
17:    Update $\mathcal{M}_r$ with class-balanced subset from $\mathcal{D}_t$ ($\lfloor|\mathcal{M}_r|/|\mathcal{Y}_t|\rfloor$ per class)
18:    Evaluate model on test stream
19: **end for**
20: **Return:** $\theta$, $\{\mathbf{p}_c \mid c \in \mathcal{Y}_t\}$
```

prototypes for the target classes of the batch. So, before the backward pass, for each target $y_i \in B$, we retrieve the prototype $\mathbf{p}^c$ and select the nearest other class $s \neq c$ from $\mathcal{Y}_t$ and its prototype $\mathbf{p}^s$. Then sample $\lambda \sim \mathrm{Beta}(\sigma, \sigma)$, with $\sigma = 0.35$ and compute the mixed prototype:

$$\mathbf{p}_{\text{mixed}} = \text{normalize}\left((1-\lambda)\mathbf{p}^c + \lambda\mathbf{p}^s\right),$$

Here, normalize denotes the normalization of ℓ_2 to map the temporary prototype to a unit hyperspace.

5. **Loss Computation**: The loss function PPP takes $\mathbf{p}_{\text{mixed}}$, pushes $\mathbf{f}_i$ away from the confusing prototype, and gives it a hard margin. We perform a gradient step using SGD to update the model parameters θ as usual,

$$\theta \leftarrow \theta - lr \cdot \nabla_\theta \mathcal{L}.$$

6. **Prototype Momentum Update**: After the backward pass, we update the prototypes for classes present in batch B via Equation (3.4).

7. **Update Memory Buffer**: After processing all mini-batches in $\mathcal{D}_t$, the memory buffer$\mathcal{M}_r$ is updated with a class-balanced subset of samples from $\mathcal{D}_t$. For $K = |\mathcal{Y}_t|$ seen classes, we select up to $\lfloor |\mathcal{M}_r|/K \rfloor$ samples per class in $\mathcal{C}_t$.

8. **Evaluation**: After each task t, we evaluate the model in the test stream and obtain evaluation metrics.

The proposed approach tries to enforce a hard margin on the decision boundary of confusing prototypes using prototype interpolation and improves the CoPE method. The core principles of CoPE methods are respected by maintaining accurate class prototypes and updating them using a momentum-based rule, while keeping uniform sampling for fair representation.

5 Experiments

In this section, we will describe the groundwork laid to ensure that all the methods work under the best conditions. This consists of the datasets used, hyperparameter tuning, model architecture, and all optimal settings.

5.1 Dataset and Benchmark

To run our experiments, we chose standard datasets that are publicly available and uniform in the works we studied in chapter 2. The benchmarks selected to run the experiments are the CIFAR-10 and CIFAR-100 datasets[31]. CIFAR-10 and CIFAR-100 are standard benchmarks consisting of 10 and 100 classes, respectively, and consist of approximately 60,000 colored images. For CIFAR-10, the dataset is split into five tasks, each with two labels, resulting in around 10,000 samples per task. To visualize the details of the classes under CIFAR-10, refer to Figure 5.1.

For CIFAR-100, the split is done into 10 tasks, ensuring each task carries 10 labels and a total of 5,000 samples. All classes under CIFAR-100 can be summarized in 20 subclasses, shown in Table 5.1 [31]. The data sets also went through a series of data augmentation operations for better classification.

5.2 Experimental Set-Up

For the implementation of Code, we relied on the Avalanche library[32], which is an open source end-to-end continual library written in PyTorch. Avalanche is a beginner-friendly and easy-to-use continual library that provides access to different CL methods to run experiments, such as benchmarks, models, and evaluation metrics, all under one umbrella. The library is also highly modifiable with the help of plugins and overriding methods, following the syntax provided in the documentation.

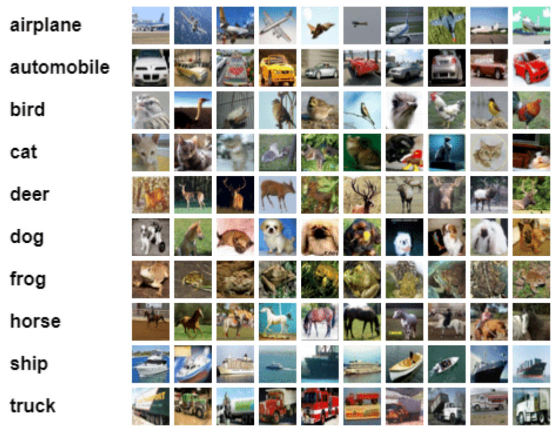

Figure 5.1: CIFAR-10 Benchmark

Table 5.1: CIFAR-100 Dataset Classes Categorization

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

The Vanilla CoPE and Vanilla ER methods were imported directly from the Avalanche. The improved methods, such as Vanilla CoPE + APF, Enhanced CoPE + PDI, and Enhanced ER, are written by creating a custom plugin and passing it to the respective continuous learning method. For OnPro, we had to customize the implementation from scratch in Avalanche, following the syntax defined in its documentation. The use of the same library across all the methods helps us to maintain uniformity and find a centralized space where everything necessary can be made available via a single resource.

All experiments were performed on an NVIDIA T4 GPU, which is renowned for its high performance and efficiency in a cluster environment. It provides mixed-precision training and inference, which helps it deliver improved performance in various modern AI applications.

5.3 Model Architecture

The model implemented for training is the same architecture used in the original work for fair experimentation and optimal performance. For ER and CoPE, we have used the SlimResNet18 model [33] as the backbone, imported via Avalanche, and the same applies to its enhanced version. The SlimResNet18 model [33] is a compact version of ResNet18 designed to give optimal performance in contrastive learning tasks in a resource-constrained environment. The network structure is the same in both ResNet-18 and SlimResNet18, but they differ in the number of layers, with SlimResNet18 being a lightweight variant that comprises 10-14 layers of the same residual network structure with reduced parameters and computational cost. To align with the CoPE architecture, we replaced the classification head to enable contrastive prototypical learning, and the output of the backbone is passed through a projection layer that maps the high-dimensional features into a lower-dimensional feature space. After projection, l_2-normalization is applied to map the embedding into unit hyperspace, necessary to obtain cosine similarity used in the PPP loss function.

The Backbone architecture of OnPro [1] is a ResNet-18 [33], which is suitable for both classification and contrastive learning. It consists of a total of 20 learnable layers with 2 layers for initial processing, 16 layers for residual blocks, and 2 final fully connected layers. The model begins with a 3x3 convolutional layer that processes the input images, followed by batch normalization and a ReLU activation function. The initial layers normalize the input data and introduce non-linearity to extract features efficiently and mitigate the vanishing gradient problem. The core architecture of the model uses four residual blocks with a standard ResNet design, each block contains a 3x3 convolutional layer, followed by batch normalization and the ReLU activation function. The number of filters also increases as we go to deeper layers from nf to 2nf, 4nf, and finally 8nf, with a downsampling via strides to adjust feature space. After the final residual block, the model applies global pooling to reduce the spatial dimension and prevent the risk of overfitting by minimizing the number of parameters. The classifier head is a fully connected layer that maps the feature vector to class scores, and the projection head is a linear layer that projects the feature vector to a 128-dimensional latent space.

5.4 Hyperparameter Tuning

The training of the model has been made only on 1 epoch to make sure the training is compliant with the standard online CL setting. Each method is trained for a fixed memory size according to the benchmark. For CIFAR-10, we used 3 different buffer sizes: 100, 200, and 500, and for CIFAR-100, they are 500, 1000, and 2000. All of the experiments were performed on 3 seeds to obtain the deviation and a margin of variance. For CIFAR-10, we set the last layer of the model to be mapped to 128 and 256 for CIFAR-100. To find the optimal hyperparameters, we performed an extensive grid search on a series of learning rates (*lr*), batch sizes, and temperature on the first few tasks of the stream, and the parameters that gave the best performance to the model were selected. We conducted experiments with a total of three learning rates: 0.005, 0.01, and 0.02. We can see the impact of the learning rate η in Figure 5.2. The rate at which the model gave optimal performance in most cases was 0.01.

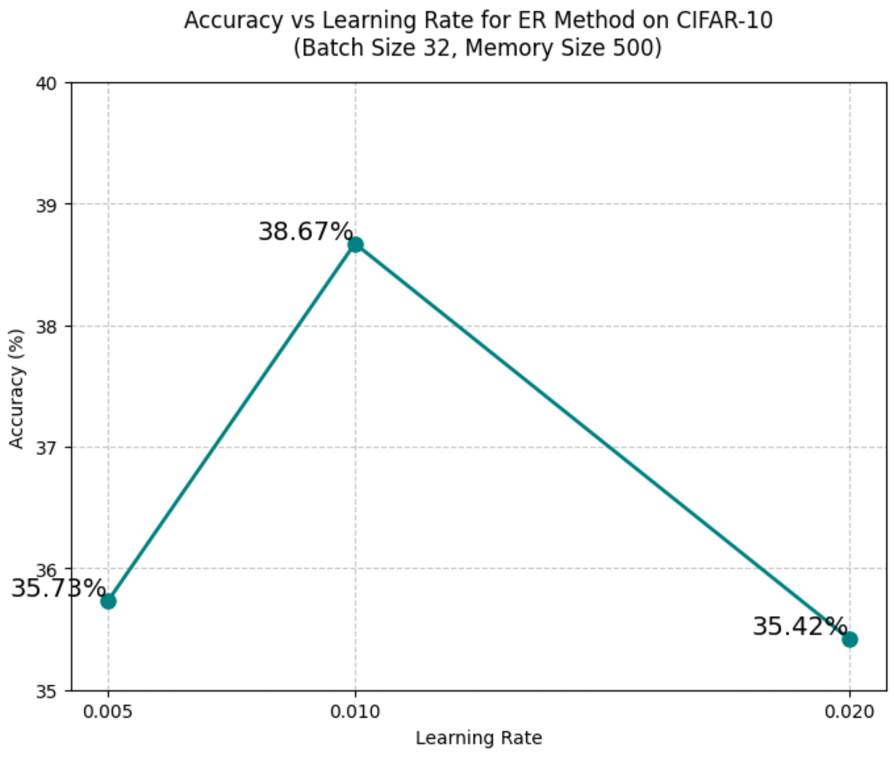

Figure 5.2: Impact of Learning Rate on ER Accuracy on CIFAR-10, memory size 500

For batch size, we used 32 for CIFAR-10 and 64 for CIFAR-100 in ER and its enhancement. As CoPE, Enhanced, and Vanilla CoPE + APF we used SGD with a limited processing batch size of 10[14, 16]. Temperature is an important parameter for modulating the concentration of the softmax layer. A lower temperature enhances the model's ability to focus on samples prone to forgetting, improving the stability of the model. We ran the model on different temperature settings, which are 0.1, 0.2, 0.25, and 0.3. In Figure 5.3, we can see that learning 0.25 gave optimal performance compared to others, and in the case of the CIFAR-10 dataset, for CIFAR-100, 0.1 was better.

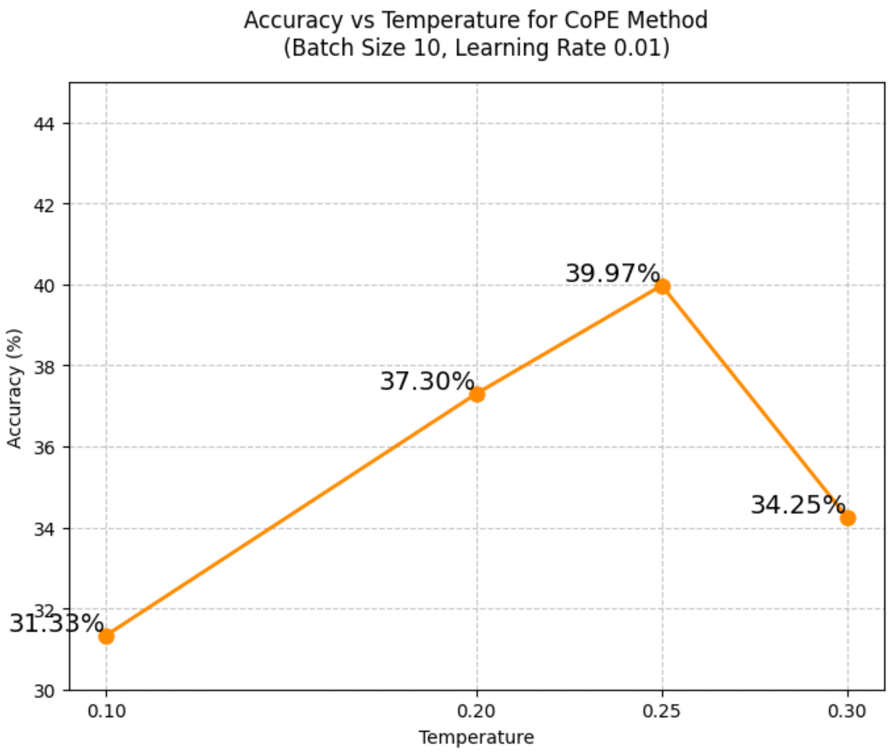

Figure 5.3: Impact of Temperature on CoPE Accuracy on CIFAR-10 memory size 500

6 Results

In this section, we present the results obtained from different baselines and their enhanced versions. The primary investigation is carried out on the original methods, followed by an evaluation of improvements by implementing priority-based sampling in it and proposing enhancements. Our experiments used both the CIFAR-10 and CIFAR-100 datasets, evaluating performance on evaluation metrics such as accuracy and forgetting. In the latter part of this section, we will see the performance of the proposed method. Now, let us see the preliminary results.

6.1 Performance of Vanilla ER and Enhanced ER

For performing experiments on ER, we have maintained the base assumption that it is a replay-based method that stores raw images rather than prototypes. Taking this assumption into account, we integrated a priority-based sampling method that selects samples based on their risk of being misclassified. We conducted all the experiments in the setting described in the Experimental Section.

In Table 6.1, you can see a side-by-side comparison of both Vanilla ER and Enhanced ER on the CIFAR-10 benchmark with evaluation metrics such as accuracy and forgetting. Similarly, in Table 6.2, a comparison is made between the two methods on the CIFAR-100 benchmark.

In Table 6.1 and Table 6.2, we can see that Enhanced ER with APF-inspired replay performs comparatively better than Vanilla ER. Both of these methods took approximately 1.5 minutes and 2 minutes on the CIFAR-10 and CIFAR-100 benchmarks, respectively.

We also ran the experiments for multiple iterations to observe the progression as the number of iterations increased to improve accuracy. It can be observed in Figure 6.1 that as the number of iterations increases, the accuracy improves initially, but for higher iterations it drops again, indicating overfitting.

These results clearly indicate that priority-based sampling offers better performance than random sampling and thus aligns with the assumption we made.

Method	Memory Size	Accuracy	Forgetting
Vanilla ER	100	26.4 ± 2.94	64.3 ± 3.6
	200	32.2 ± 2.7	53.9 ± 3.1
	500	44.3 ± 0.6	45.6 ± 2.9
Enhanced ER	100	33.5 ± 0.9	62.3 ± 1.3
	200	38.5 ± 0.2	51.9 ± 2.5
	500	48.7 ± 1.3	30.9 ± 1.5

Table 6.1: Comparison of Performance of Vanilla ER and Enhanced ER on CIFAR-10 across different memory sizes

Method	Memory Size	Accuracy	Forgetting
Vanilla ER	500	12.6 ± 0.7	38.0 ± 2.3
	1000	12.9 ± 0.6	37.8 ± 0.7
	2000	14.9 ± 0.4	34.7 ± 1.3
Enhanced ER	500	14.8 ± 1.3	33.8 ± 1.7
	1000	19.4 ± 0.5	23.0 ± 1.5
	2000	22.4 ± 0.14	18.7 ± 1.7

Table 6.2: Comparison of Performance of Vanilla ER and Enhanced ER on CIFAR-100 across different memory sizes

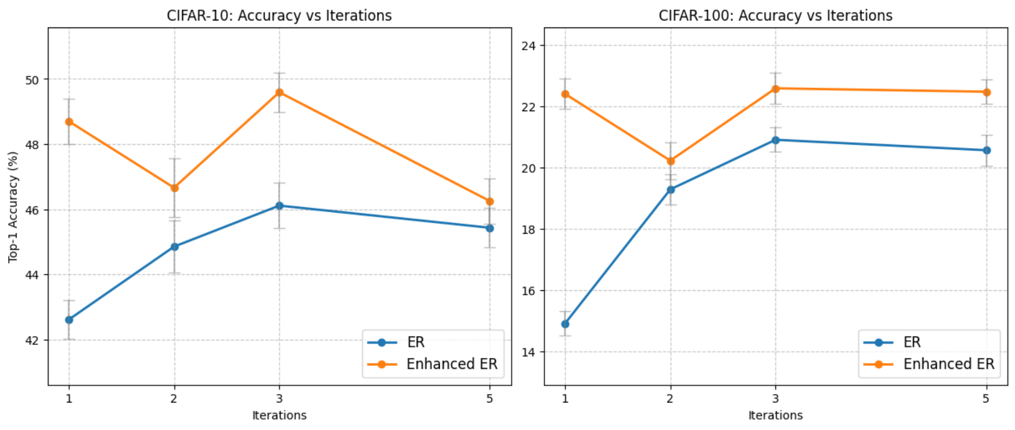

Figure 6.1: Impact of Iteration on Vanilla ER and Enhanced ER Accuracy on CIFAR-10 memory size 500

6.2 Performance of CoPE, Vanilla CoPE+APF, and Enhanced CoPE + PDI

In this section, we will compare the performance of the CoPE method, Vanilla CoPE + APF, with that of our proposed Enhanced CoPE + PDI method. In Vanilla CoPE with the APF method, we are implementing a priority-based sampling strategy, replacing the existing uniform sampling method in CoPE to observe the impact of giving more importance to classes prone to misclassification. But in Enhanced CoPE + PDI, we go one step further, as we recall from Section 2.2.2, that CoPE uses PPP function loss to optimize the network, and the PPP uses the Attractor and Repellor Set mechanism to do so. The attractor set pulls all the samples from the same classes together, and the repeller set pushes everything else. However, this uniform push to other classes leaves a gap in not giving more importance to the prototype of the closest class. In the proposed approach, we addressed this by feeding interpolated prototypes to the loss function, so it gives a hard margin to those classes and diverts more attention there. All experiments were conducted in standard settings for one iteration to respect the online continual learning assumption.

Method	**Memory Size**	**Accuracy**	**Forgetting**
CoPE	100	30.2 ± 1.2	49.7 ± 1.5
	200	34.7 ± 0.6	46.4 ± 0.8
	500	39.6 ± 1.4	40.2 ± 1.7
Vanilla CoPE + APF	100	34.0 ± 1.4	47.9 ± 1.1
	200	35.4 ± 1.2	44.7 ± 0.4
	500	39.5 ± 0.9	40.8 ± 1.2
Enhanced CoPE + PDI	100	35.1 ± 1.2	46.7 ± 1.8
	200	42.6 ± 0.9	42.6 ± 1.1
	500	45.9 ± 0.6	38.5 ± 0.7

Table 6.3: Comparison of Performance of CoPE, Vanilla CoPE + APF and Enhanced CoPE + PDI on CIFAR-10 across different memory sizes
‘

In Table 6.3 and Table 6.4, we can also see that there is not much variation in the results between CoPE and Vanilla CoPE+APF. A possible hypothesis to explain this is that the impact of priority sampling is being nullified by PPP loss, even if we perform priority sampling in the early stages. In later stages, the loss function is pushing all classes away uniformly, compromising the efficacy of the APF-inspired sampling strategy. This has been well addressed by the enhanced CoPE+PDI, as evidenced by the results. The proposed approach, Enhanced CoPE+PDI, performs better than the original CoPE and Vanilla CoPE with APF and attempts to counter the gap in PPP loss, better optimizing the network.

We also run the experiments for multiple iterations in these approaches to see what the results would be like when the model becomes more familiar with the data.

Method	Memory Size	Accuracy	Forgetting
CoPE	500	12.2 ± 0.5	39.6 ± 1.2
	1000	14.2 ± 1.1	20.4 ± 1.8
	2000	16.3 ± 0.4	18.9 ± 0.7
Vanilla CoPE + APF	500	11.9 ± 0.4	39.6 ± 1.2
	1000	13.8 ± 0.6	21.6 ± 1.7
	2000	15.1 ± 0.5	19.4 ± 0.6
Enhanced CoPE + PCI	500	13.1 ± 0.2	27.7 ± 0.3
	1000	16.9 ± 0.5	17.1 ± 0.2
	2000	18.1 ± 0.6	13.4 ± 0.7

Table 6.4: Comparison of Performance of CoPE, Vanilla CoPE + APF, and Enhanced CoPE + PDI on CIFAR-100 across different memory sizes

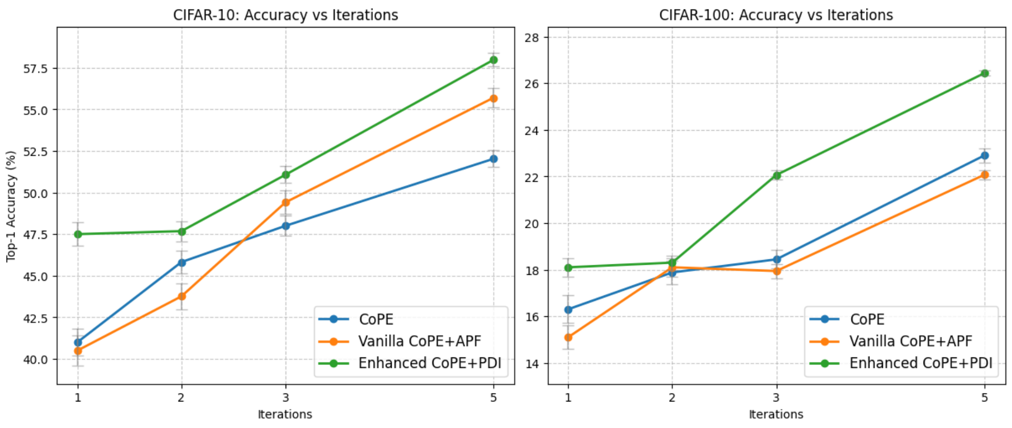

Figure 6.2: Impact of iterations on CoPE Accuracy on CIFAR-10 and CIFAR-100 for memory size 500 and 2000

The results show that as the number of iterations increases, the accuracy improves because the model becomes more familiar with the data, and this also reduces the likelihood of forgetting.

6.3 Comparison of Baselines and their Enhanced Version with OnPro

To ensure a fair comparison, we implemented all methods using the Avalanche library. This fair comparison also drove us to implement the original process from which we took inspiration, i.e., OnPro. We implemented OnPro, drawing inspiration from the original code in Avalanche, and the results match those of the original method code. This indicates that standardization is effective and that the methods retain their effectiveness. In Table 6.5, we can see the results of the OnPro [1] method on CIFAR-10 and CIFAR-100.

Method	Memory Size	Accuracy	Forgetting
OnPro CIFAR-10	100	55.4 ± 0.8	34.2 ± 1.2
	200	65.1 ± 0.8	17.9 ± 0.5
	500	70.5 ± 0.4	15.5 ± 0.7
OnPro CIFAR-100	500	20.7 ± 0.8	14.8 ± 0.5
	1000	28.9 ± 0.2	12.1 ± 0.2
	2000	35.3 ± 0.6	7.5 ± 0.6

Table 6.5: Performance of OnPro on CIFAR-10 & and CIFAR-100 across different memory sizes

OnPro provides high performance as an online CL method, and as expected of a state-of-the-art method. It introduced many novel concepts, such as OPE and APF. Now, let us try to put all the methods we have seen together and analyze how each method is doing.

The Table 6.6 draws a comparison of all methods together. As we can see, our primary assumption that implementing a priority-based mechanism in sampling favors the model to focus more on classes prone to misclassification. In Figure 6.3, we can also see a visual picture of how the methods perform and compare the accuracy of the methods on both benchmarks.

Sometimes we have to go a step further in analysis, as seen in the example of CoPE, where we had to intervene on how the loss function takes the prototype, and this indicates that the scope of improvements can often lie deeper. Now, let us take a look at how much time each method is taking to run for training and observe their efficiency in terms of running time.

ER takes the least time, approximately 2 minutes, due to its simple mechanism. CoPE takes around 8 minutes, along with the enhancements, which is not bad for prototype-based online CL methods. Finally, OnPro takes around 29 minutes, and as the method involves many complex operations. The objective of this work has been to find the scope of improvement in the existing method and implement it while maintaining cost efficiency.

In Figure 6.5, we can also see how accurate is changing over time. Enhanced ER improves the accuracy of the baseline ER without wasting much time. Similarly, if we consider Enhanced CoPE + PDI, it is also inexpensive in

Method	Memory Size	Accuracy	Forgetting
Vanilla ER	100	26.4 ± 2.94	64.3 ± 3.6
	200	32.2 ± 2.7	53.9 ± 3.1
	500	44.3 ± 0.6	45.6 ± 2.9
Enhanced ER	100	33.5 ± 0.9	62.3 ± 1.3
	200	38.5 ± 0.2	51.9 ± 2.5
	500	48.7 ± 1.3	30.9 ± 1.5
CoPE	100	30.2 ± 1.2	49.7 ± 1.5
	200	34.7 ± 0.6	46.4 ± 0.8
	500	39.6 ± 1.4	40.2 ± 1.7
Vanilla CoPE + APF	100	34.0 ± 1.4	47.9 ± 1.1
	200	35.4 ± 1.2	44.7 ± 0.4
	500	39.5 ± 0.9	40.8 ± 1.2
Enhanced CoPE + PDI	100	35.1 ± 1.2	46.7 ± 1.8
	200	42.6 ± 0.9	42.6 ± 1.1
	500	45.9 ± 0.6	38.5 ± 0.7
OnPro	100	55.4 ± 0.8	34.2 ± 1.2
	200	65.1 ± 0.8	17.9 ± 0.5
	500	70.5 ± 0.4	15.5 ± 0.7

Table 6.6: Comparison of all the Methods on CIFAR-10

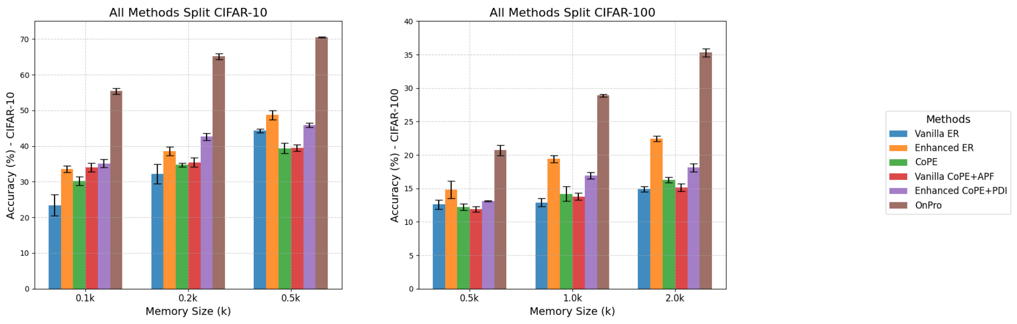

Figure 6.3: All the methods together on CIFAR-10 and CIFAR-100

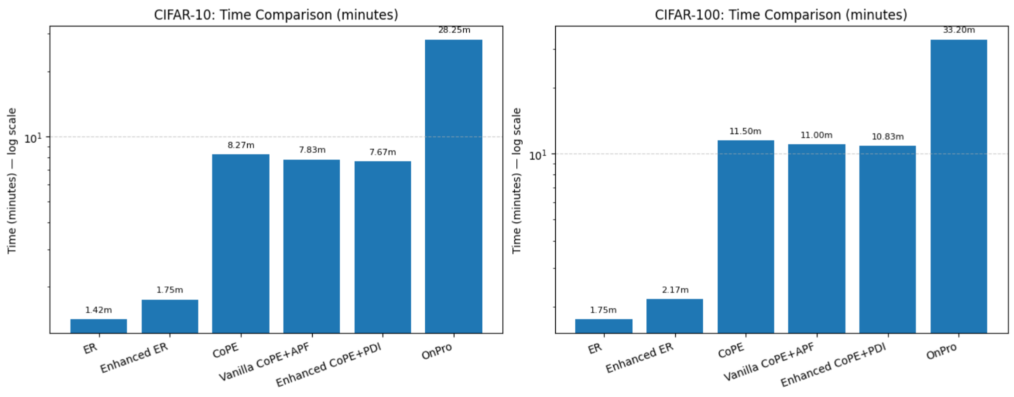

Figure 6.4: Time Comparison of All Methods on CIFAR-10 and CIFAR-100

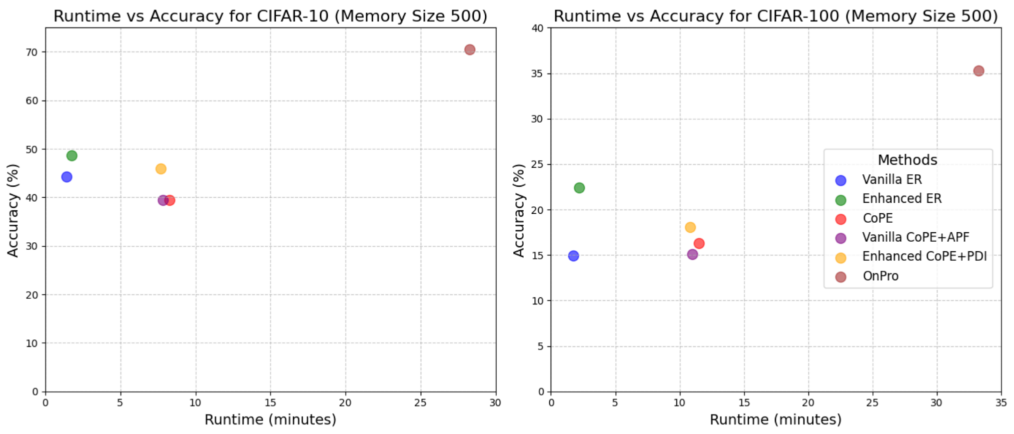

Figure 6.5: Accuracy vs Time Comparison of All Methods on CIFAR-10 with Memory Size 500 and CIFAR-100 with Memory Size 2000

terms of time and has almost the same running time as CoPE and Vanilla CoPE + APF with improved accuracy. If we compare our enhanced CoPE + PDI to OnPro, the methods perform less in terms of accuracy, but if we are using time as a factor, the picture changes somewhat. As our proposed method gives optimal performance and is time-efficient. After observing the results, we can see that our attempt has been successful with both the prototype-based CoPE method and the non-prototype-based method, i.e., storing raw samples and ER in an online CL replay setting. Both methods, Enhanced ER and enhanced CoPE+PDI, are cost-efficient and also provide a boost in performance of the original methods.

7 Conclusion

This thesis tried to address critical challenges faced by ML models in online continual learning by proposing enhancements that can help replay-based methods deal with catastrophic forgetting. The work successfully demonstrated that sampling data strategically and feeding data specific to forgetting to the loss function can improve the performance of existing methods while maintaining computational efficiency. We started by integrating the APF-inspired mechanism into ER, replacing random sampling with a more focused strategy that focused on examples prone to misclassification. The results demonstrated consistent improvements across the CIFAR-10 and CIFAR-100 datasets in increasing accuracy and reducing forgetting. The significant contribution of this thesis has been to enhance the CoPE method by integrating the Prototype Distance Interpolation (PDI) mechanism within its loss function. This addressed a fundamental limitation of PPP loss, where the Repellor set uniformly pushed away all the non-target classes without giving more importance to confusing classes. The proposed enhanced CoPE + PDI demonstrated improvements in accuracy gain and reduced forgetting, while optimizing resource utilization. Enhanced ER took approximately 2 minutes, while enhanced CoPE + PDI took approximately 8 minutes, which is like $1/3^{rd}$ of OnPro, which took almost 29 minutes, making our solution practically viable for real-world applications.

The comparative analysis revealed important information on the interaction between sampling strategy and loss function in CL. The observation that Vanilla CoPE+APF showed minimal improvement over baseline CoPE highlighted the importance of considering the entire CL pipeline. This led to our insight that priority sampling alone is not enough when the loss function counteracts the impact of the sampling strategy. Our proposed method successfully addressed this by feeding mixed prototypes to the loss function, such that the focus of the model is shifted toward refining the decision boundary between confusing classes and the target class. Further analysis of the iterations also revealed that there is a delicate balance that needs to be maintained in online CL. Increased iterations initially improve performance, but eventually lead to overfitting, and emphasize the practicality of single-pass learning.

Although our work demonstrated improvements to the current paradigm of CL, it focused only on evaluation on the CIFAR-10 and CIFAR-100 datasets and thus opens the scope for further improvements. The future of this work can be to extend this method to larger datasets across multiple domains, as we primarily considered image classification as the main task, but this can be extended to domains beyond that.

After examining the baselines and comparing them with the enhanced version, we can justify that our work has been successful in addressing limitations in existing replay-based CL methods through systematic enhancements that prioritize both performance and efficiency. The proposed enhanced CoPE + PDI method demonstrated that proactive integration of priority-based mechanisms can offer opportunities for improvement while maintaining computational tractability.

References

[1] Y. Wei, J. Ye, Z. Huang, J. Zhang, and H. Shan, "Online prototype learning for online continual learning," 2023. [Online]. Available: https://arxiv.org/abs/2308.00301

[2] T. Mitchell, *Machine Learning*, ser. McGraw-Hill International Editions. McGraw-Hill, 1997. [Online]. Available: https://books.google.it/books?id=EoYBngEACAAJ

[3] T. Flesch, J. Balaguer, R. Dekker, H. Nili, and C. Summerfield, "Comparing continual task learning in minds and machines," *Proceedings of the National Academy of Sciences*, vol. 115, no. 44, pp. E10 313–E10 322, 2018. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1800755115

[4] G. M. van de Ven, N. Soures, and D. Kudithipudi, "Continual learning and catastrophic forgetting," 2024. [Online]. Available: https://arxiv.org/abs/2403.05175

[5] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," ser. Psychology of Learning and Motivation, G. H. Bower, Ed. Academic Press, 1989, vol. 24, pp. 109–165. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0079742108605368

[6] R. Ratcliff, "Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions." *Psychological Review*, vol. 97, no. 2, p. 285–308, 1990. [Online]. Available: http://dx.doi.org/10.1037/0033-295X.97.2.285

[7] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1312.6211

[8] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," *CoRR*, vol. abs/1801.10112, 2018. [Online]. Available: http://arxiv.org/abs/1801.10112

[9] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *CoRR*, vol. abs/1802.07569, 2018. [Online]. Available: http://arxiv.org/abs/1802.07569

[10] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *Computational Intelligence Magazine, IEEE*, vol. 10, pp. 12–25, 11 2015.

[11] G. M. van de Ven and A. S. Tolias, "Three scenarios for continual learning," *CoRR*, vol. abs/1904.07734, 2019. [Online]. Available: http://arxiv.org/abs/1904.07734

[12] M. D. Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. G. Slabaugh, and T. Tuytelaars, "Continual learning: A comparative study on how to defy forgetting in classification tasks," *CoRR*, vol. abs/1909.08383, 2019. [Online]. Available: http://arxiv.org/abs/1909.08383

[13] S. Rebuffi, A. Kolesnikov, and C. H. Lampert, "icarl: Incremental classifier and representation learning," *CoRR*, vol. abs/1611.07725, 2016. [Online]. Available: http://arxiv.org/abs/1611.07725

[14] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continuum learning," *CoRR*, vol. abs/1706.08840, 2017. [Online]. Available: http://arxiv.org/abs/1706.08840

[15] H. Ritter, A. Botev, and D. Barber, "Online structured laplace approximations for overcoming catastrophic forgetting," 2018. [Online]. Available: https://arxiv.org/abs/1805.07810

[16] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," *CoRR*, vol. abs/1711.09601, 2017. [Online]. Available: http://arxiv.org/abs/1711.09601

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

[18] F. Zenke, B. Poole, and S. Ganguli, "Improved multitask learning through synaptic intelligence," *CoRR*, vol. abs/1703.04200, 2017. [Online]. Available: http://arxiv.org/abs/1703.04200

[19] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. S. Torr, and M. Ranzato, "Continual learning with tiny episodic memories," *CoRR*, vol. abs/1902.10486, 2019. [Online]. Available: http://arxiv.org/abs/1902.10486

[20] A. Mallya and S. Lazebnik, "Packnet: Adding multiple tasks to a single network by iterative pruning," *CoRR*, vol. abs/1711.05769, 2017. [Online]. Available: http://arxiv.org/abs/1711.05769

[21] Z. Mai, R. Li, H. Kim, and S. Sanner, "Supervised contrastive replay: Revisiting the nearest class mean classifier in online class-incremental continual learning," *CoRR*, vol. abs/2103.13885, 2021. [Online]. Available: https://arxiv.org/abs/2103.13885

[22] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning," *CoRR*, vol. abs/2011.00362, 2020. [Online]. Available: https://arxiv.org/abs/2011.00362

[23] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 1597–1607. [Online]. Available: https://proceedings.mlr.press/v119/chen20j.html

[24] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," *CoRR*, vol. abs/1911.05722, 2019. [Online]. Available: http://arxiv.org/abs/1911.05722

[25] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance-level discrimination," *CoRR*, vol. abs/1805.01978, 2018. [Online]. Available: http://arxiv.org/abs/1805.01978

[26] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," *CoRR*, vol. abs/2004.11362, 2020. [Online]. Available: https://arxiv.org/abs/2004.11362

[27] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka, "Metric learning for large scale image classification: Generalizing to new classes at near-zero cost," in *European Conference on Computer Vision (ECCV)*, ser. Lecture Notes in Computer Science, vol. 7573. Springer, 2012, pp. 488–501.

[28] J. Snell, K. Swersky, and R. S. Zemel, "Prototypical networks for few-shot learning," *CoRR*, vol. abs/1703.05175, 2017. [Online]. Available: http://arxiv.org/abs/1703.05175

[29] H. Lee, S. J. Hwang, and J. Shin, "Self-supervised label augmentation via input transformations," in *International Conference on Machine Learning*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:220249782

[30] M. De Lange and T. Tuytelaars, "Continual prototype evolution: Learning online from non-stationary data streams," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 8250–8259.

[31] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[32] A. Carta, L. Pellegrini, A. Cossu, H. Hemati, and V. Lomonaco, "Avalanche: A pytorch library for deep continual learning," *Journal of Machine Learning Research*, vol. 24, no. 363, pp. 1–6, 2023. [Online]. Available: http://jmlr.org/papers/v24/23-0130.html

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

Acknowledgments

I express my sincere gratitude to the individuals who made this thesis possible through their guidance and support.

First and foremost, I would like to extend my deepest gratitude to my supervisor, Prof. Nicolo Navarin. Your belief in this research from its early stages and your thoughtful guidance helped me a lot to shape the idea into a meaningful contribution. Thank you for the numerous meetings and patiently explaining complex concepts and for challenging me to deepen my analysis. Your expertise in continual learning has been instrumental in developing this work. I am also particularly grateful to Giovanni Donghi for his invaluable guidance throughout this research. Your help during challenging concepts, the willingness to explain them, and the consistent availability to help, despite having your own commitments, were truly appreciated. Your deep knowledge of CL and prototype-based methods significantly improved the quality of this work.

My sincere appreciation goes to my family, especially my mom, for their unwavering support throughout this academic endeavor. I am also grateful to that close friend who offered constant encouragement and reminded me to maintain perspective during the most intense periods of this research.

This thesis also deeply acknowledges the valuable contributions of the research community, particularly the authors of the OnPro, CoPE, and ER methods, and the developers of the Avalanche library, whose open-source implementations and well-documented approaches enabled this comparative study. Their commitment to reproducible research greatly facilitated this work.