THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : EEATS - Electronique, Electrotechnique, Automatique, Traitement du Signal
Spécialité : Signal Image Parole Télécoms
Unité de recherche : CEA /LIST - Laboratoire d'intégration de systèmes et de technologies

Stratégies de rejeu pour l'apprentissage incrémental dans les réseaux de neurones entièrement binarisés sous contraintes mémoire

Replay strategies for incremental learning in fully-binary neural networks under memory constraints

Présentée par :

Yanis BASSO-BERT

Direction de thèse :

Antoine DUPRET Chercheur CEA, CEA / DFSE	Directeur de thèse
Anca MOLNOS INGINIÉURE DOCTEURE, CEA List	Co-encadrante de thèse
William GUICQUERO INGENIEUR DOCTEUR, CEA Leti	Co-encadrant de thèse

Rapporteurs :

Vincent GRIPON
DIRECTEUR DE RECHERCHE, CNRS
Céline HUDELOT
PROFESSEURE DES UNIVERSITES, CentraleSupélec

Thèse soutenue publiquement le **17 juin 2025**, devant le jury composé de :

Antoine DUPRET, DOCTEUR EN SCIENCES HDR, CEA	Directeur de thèse
Vincent GRIPON, DIRECTEUR DE RECHERCHE, CNRS	Rapporteur
Céline HUDELOT, PROFESSEURE DES UNIVERSITES, CentraleSupélec	Rapporteure
Frederic PETROT, PROFESSEUR DES UNIVERSITES, Grenoble INP - UGA	Examinateur
Sylvie LE HÉGARAT-MASCLE, PROFESSEURE DES UNIVERSITES, Université Paris-Saclay	Examinatrice

Invités :

Romain LEMAIRE
INGENIEUR DE RECHERCHE, CEA LIST
Anca MOLNOS
DOCTEURE EN SCIENCES, CEA LIST

Résumé

Mots-clés : Réseau de Neurones Binaire, Apprentissage Profond, Apprentissage Incrémental, Mécanismes de Rejeu, Intelligence Artificielle Embarquée, Quantification

L'essor des capteurs intelligents s'accompagne d'un besoin croissant de déployer des modèles d'apprentissage profond au plus près des capteurs, permettant une inférence embarquée. Cette approche réduit la latence, la consommation de bande passante et d'énergie. Pour répondre à ces exigences, typiquement quelques mégabits de mémoire et milliwatts de puissance, les modèles doivent être conçus spécifiquement pour une exécution efficace. Les réseaux de neurones binaires (BNNs) apparaissent comme une solution prometteuse, en tirant parti d'une arithmétique en logique binaire, économe en mémoire et en énergie.

Cependant, l'inférence proche-capteur impose aussi aux modèles de s'adapter au fil du temps, à mesure que de nouveaux concepts émergent ou évoluent durant le déploiement. L'apprentissage incrémental permet cette adaptation, en autorisant le réentraînement du modèle sur de nouvelles données tout en préservant les connaissances acquises. Parmi les techniques disponibles, les méthodes de rejeu, qui consistent à conserver ou approximer les données passées via des exemples stockés ou générés, se révèlent particulièrement efficaces. Néanmoins, l'optimisation des BNNs reste difficile en raison de leur nature non différentiable, et cette complexité est amplifiée dans le cadre des entraînements successifs exigés par l'apprentissage incrémental. Les approches de rejeu classiques entraînent un surcoût mémoire souvent incompatible avec les contraintes embarquées.

Cette thèse explore dans quelle mesure les réseaux de neurones entièrement binarisés (FBNN) peuvent supporter l'apprentissage incrémental sous contraintes mémoire, et comment concevoir des mécanismes de rejeu sobres en mémoire pour rendre cela possible.

Nous adoptons une méthodologie d'étude progressive, en explorant des stratégies de rejeu de plus en plus intégrées au réseau binaire pour réduire l'empreinte mémoire, du rejeu natif au rejeu latent, jusqu'au pseudo-rejeu. Nous établissons d'abord de bonnes pratiques pour l'apprentissage incrémental avec des FBNN compacts, incluant la conception d'une architecture entièrement binaire et un protocole de pré-entraînement semi-supervisé favorisant la transférabilité des représentations. Une étude approfondie de la fonction de coût est menée, ainsi qu'une comparaison du rejeu natif et latent sous contrainte mémoire. Ces travaux sont validés sur le jeu de données CIFAR100, puis étendus au benchmark CORE50, où notre FBNN de 3Mb atteint une exactitude comparable à celle de réseaux à poids réels.

Sur cette base, nous proposons la méthode *Generative Binary Memory* (GBM), un pseudo-rejeu basé sur des modèles de mélange de Bernoulli permettant de générer des exemples binaires synthétiques à partir de prototypes compacts. Deux binariseurs étendent GBM aux réseaux non-binaires, et la méthode atteint des performances supérieures à l'état de l'art sur CIFAR100 ($+2.9\%$) et TinyImageNet ($+1.5\%$) avec une architecture ResNet18. Sur FBNN, GBM améliore la précision finale de $+3.1\%$ sur CORE50 tout en réduisant l'usage mémoire d'un facteur 4.7. Une variante en ligne et entièrement entière de GBM est également proposée pour permettre l'adaptation directement sur dispositif.

Ce travail démontre que les FBNN, lorsqu'ils sont conçus et entraînés avec des stratégies adaptées, peuvent prendre en charge l'apprentissage incrémental tout en respectant les fortes contraintes de l'inférence embarquée. Il ouvre des perspectives concrètes pour le déploiement de systèmes d'apprentissage adaptatifs et frugaux.

Abstract

Keywords: Binary Neural Network, Deep Learning, Incremental Learning, Replay Mechanisms, Edge AI, Quantization

The rise of smart sensors has led to a growing need to deploy deep learning models close to the sensor, enabling always-on on-device inference, where models run permanently to support uninterrupted local processing. This approach reduces latency, bandwidth, and energy consumption, especially important in resource-constrained environments. To meet these constraints, typically a few megabits of memory and a few milliwatts of power, models must be specifically designed for efficient execution. Binary Neural Networks (BNN) have emerged as a promising solution, replacing the costly multiply-accumulate operations of conventional deep neural networks with efficient logic-gate-based arithmetic.

However, near-sensor inference also requires models to adapt over time, as new concepts may appear or evolve in the deployment environment. Incremental learning strategies enable this adaptability by allowing models to be retrained on new data while preserving past knowledge. Among the various techniques developed for incremental learning, replay methods, where past knowledge is retained or approximated through stored or generated data are particularly effective. Yet, training BNNs remains difficult due to their non-differentiable operations and limited expressiveness, as they rely on only two discrete values. These challenges are further compounded in incremental settings, where retraining must occur repeatedly. Moreover, conventional replay methods introduce additional memory requirements that conflict with the tight constraints of edge deployment.

This thesis investigates whether Fully Binarized Neural Networks (FBNN) can support incremental learning under extreme memory and energy constraints, and how memory-efficient replay mechanisms can be designed to make this possible.

We adopt a progressive research methodology, exploring replay strategies increasingly embedded into the binary architecture to reduce memory usage, starting from native replay, through latent replay, and finally to in pseudo-replay. We first establish best practices for training compact FBNN under incremental learning constraints. This includes the design of a fully binary architecture and a semi-supervised pretraining protocol that enhances representation transferability. We also conduct a detailed study of loss balancing and compare native versus latent replay under tight memory budgets. These findings are validated on CIFAR100 and extended to the CORE50 benchmark, where our $3Mb$ FBNN achieves accuracy comparable to larger real-valued networks.

Building on this foundation, we introduce *Generative Binary Memory* (GBM), a pseudo-replay method based on Bernoulli Mixture Models that generates synthetic binary exemplars from compact prototype representations. Two binarization modules extend GBM to non-binary networks, and the method achieves state-of-the-art results on CIFAR100 ($+2.9\%$) and TinyImageNet ($+1.5\%$) with a ResNet18 backbone. On FBNN, GBM improves final accuracy by $+3.1\%$ on CORE50 while reducing memory usage by a factor of 4.7. An online, integer-only variant of GBM is also proposed to support on-device adaptation.

This work demonstrates that FBNN, when designed and trained with appropriate strategies, can support incremental learning while meeting the stringent requirements of embedded inference. It opens promising directions for deploying adaptive, low-power learning systems at the edge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Smart sensors are transforming the modern world by enabling real-time, local data processing to reduce latency, bandwidth usage, and energy consumption. In this context, deploying machine learning models on ultra-low-power devices presents significant challenges due to limited computational resources. Traditional Deep Neural Networks (DNNs) are too costly for near-sensor inference, requiring excessive memory and computation. Binary Neural Networks (BNNs) offer a compact, energy-efficient alternative by replacing operations with simple logic blocks, drastically reducing power consumption. This enables the design of circuits that can be much more optimized and efficient so that neural networks can be closely integrated next to sensors. This enables real-time, always-on processing in smart sensors.

Despite their advantages, BNNs struggle with incremental learning, a setting where models are updated sequentially on new data without access to all past samples, often leading to performance degradation on previously learned tasks.

This chapter first explores the evolution of smart sensors from centralized to on-device processing, highlighting the computational constraints they face. It then introduces BNNs as a disruptive solution for energy-efficient machine learning at the edge. Finally, it discusses incremental learning in BNNs as potential solutions for lifelong learning in resource-limited environments.

Beyond establishing the technical context, this chapter positions the scientific contributions of the thesis within the broader research landscape. It identifies a gap in the study of fully binarized neural networks (FBNNs) in incremental learning scenarios and motivates a structured exploration of replay mechanisms in this setting. Finally, the remainder of this chapter states the plan of the manuscript.

1.1 Context and background

Smart sensors are increasingly deployed in various applications, from environmental monitoring to industrial automation and wearable health devices. To maximize their utility, these sensors must process data locally, reducing reliance on cloud computing to mitigate bandwidth, latency, and privacy concerns. However, this shift toward on-device intelligence presents significant challenges, particularly for ultra-low-power devices constrained by energy, memory, and computational resources. While DNNs offer powerful solutions for edge computing, their high computational cost makes them impractical for extreme-edge deployment. BNNs emerged as a disruptive alternative, drastically reducing memory footprint and energy consumption through highly efficient binary operations. Yet, deploying BNNs in real-world scenarios introduces another challenge: adapting to non-stationary data without catastrophic forgetting, where new knowledge overwrites prior information.

Replay-based methods, inspired by biological memory mechanisms, offer a promising approach to mitigate this issue and enable lifelong learning in constrained settings. This section first examines the transition from centralized to on-device processing, then discusses the challenges faced by smart imagers, and finally introduces BNNs and replay techniques as key enablers of adaptive intelligence on resource-limited hardware.

1.1.1 Smart sensors: a need for adaptive machine learning on ultra-low power device

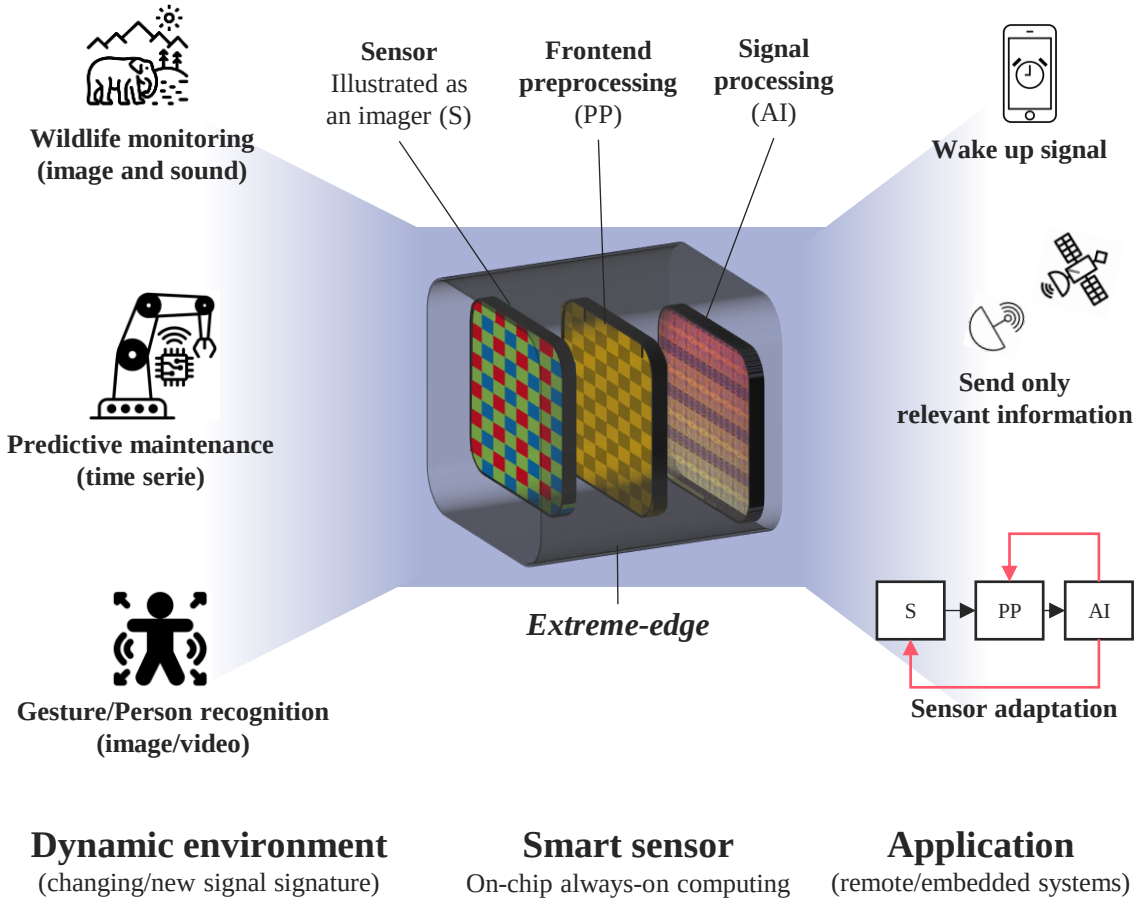


Figure 1.1: A graphical view of dynamic environement, smart sensor and their applications. Sensor is reprensented as an imager (pixel array).

Sensors: from centralized to on-device processing

Sensors have become indispensable in modern technology, seamlessly integrating into diverse applications such as computer vision, health monitoring devices, predictive maintenance systems, and environmental monitoring networks [48, 67]. From tracking vital signs in wearable health devices [159] to detecting anomalies in industrial machinery [116] or capturing environmental changes in IoT-based smart cities [13], sensors provide critical data that enable informed decision-making.

Traditionally, sensors function by collecting raw data and transmitting it to centralized servers or data centers, where all processing, analysis, and decision-making occur. This

centralized approach, while effective for limited datasets, is increasingly unsustainable due to several constraints:

- **Bandwidth Limitations**: The growing number of sensors generates vast amounts of raw data that strain network infrastructure [195].
- **Latency**: Remote processing introduces delays, making it unsuitable for applications requiring real-time responsiveness [97].
- **Energy Inefficiency**: Transmitting continuous streams of data drains the battery of resource-constrained devices [201].
- **Scalability Challenges**: Centralized systems face significant computational and resource bottlenecks as networks grow [97].
- **Privacy and Security Risks**: Transmitting sensitive data to external servers increases vulnerability to breaches [195].

To overcome these limitations, *smart sensors* have emerged, integrating sensing with localized, on-device processing. For example, in wearable devices, smart sensors are used in fitness trackers to measure heart rate variability and to issue alerts for abnormal patterns without needing constant data transmission [159]. In predictive maintenance, vibration sensors embedded with machine learning algorithms detect and classify faults in real time, enabling proactive servicing of industrial equipment [116]. Environmental monitoring systems employ smart air quality sensors that aggregate pollution data and transmit only critical updates to save bandwidth [176]. By analyzing data at the edge, smart sensors reduce bandwidth usage, improve response times, and enable energy-efficient operation, all while enhancing scalability and protecting data privacy [195].

The specific case of smart imagers

Smart imagers designed for always-on inference on-chip exemplify this shift, utilizing a low-power processing pipeline to perform uninterrupted processing in support of decision-making directly at the sensor edge. This enables maintaining minimal energy consumption, as illustrated in Figure 1.1. Such imagers can monitor their field of view continuously and trigger a wake-up signal when predefined events are detected, such as the presence of a human or motion in surveillance systems. This functionality is essential for applications requiring energy-efficient, real-time responsiveness, such as autonomous vehicles, remote wildlife monitoring, or home security systems (see Figure 1.1). In this context, smart imagers present unique computational challenges. The high dimensionality of image data necessitates advanced signal processing algorithms that can perform human-like tasks, such as image classification or object detection, with minimal computational overhead [216].

Current challenges faced by smart imagers

Three critical constraints define the design of smart imagers for edge computing, underscoring the necessity of co-designing neural network accelerators that can perform efficient inference near the sensing level while adapting to real, dynamic environments.

1. **Complexity of human-like vision tasks**: Smart imagers are required to perform tasks that mimic human vision, such as image classification, object detection, and scene understanding. These tasks involve extracting meaningful patterns from high-dimensional visual data, interpreting complex scenes, and recognizing objects or events of interest in real time. Achieving high algorithmic performance in these areas has become possible with advancements in machine learning, particularly through the development of deep neural networks (DNNs). DNNs have revolutionized computer vision by leveraging hierarchical structures to progressively extract low-level features (*e.g.*, edges or textures) and high-level abstractions (*e.g.*, object identities or scene semantics) [112][64]. Their ability to generalize across diverse datasets and adapt to complex, unstructured data makes them a natural fit for human-like vision tasks. However, the computational demands of DNNs, which often involve billions of operations, are currently unaccessible on resource-constrained edge devices such as smart imagers.

2. **Hardware constraints of edge devices**: Smart imagers operate in environments where hardware resources such as memory capacity, computational power, and energy availability are highly constrained. Unlike cloud servers or centralized data centers, edge devices rely on battery power and have limited physical space for advanced processors [12][40]. For instance, deploying a traditional DNN model on a sensor with only a few megabytes of memory and milliwatts of power budget demands innovative techniques, such as inference model compression to shrink network size and reduce computation.

3. **Adaptation to dynamic environments.** Unlike static settings, the environments in which smart imagers operate are dynamic. New signal patterns or previously unseen data classes can emerge due to environmental changes, lighting variations, or the appearance of novel objects or behaviors [160][49]. For example, in autonomous driving, a smart imager must adapt to new road signs or unexpected hazards; similarly, in surveillance, it may need to identify unfamiliar intruders. These scenarios require detecting changes in data distribution and incrementally incorporate new knowledge without forgetting previous ones. This demands the integration of adaptive learning algorithms and hardware capable of updating models on-the-fly, a capability that traditional inference-focused accelerators lack.

These combined challenges motivate the exploration and development of compact neural network design with adaptive learning mechanism specifically tailored for low energy inference in smart sensors.

1.1.2 From deep neural network to fully binary neural network: a network design for low-energy hardware mapping

Artificial neural networks

Artificial neural networks (ANNs), inspired by the structure and function of biological neurons (Figure 1.2a)), are computational models designed to learn patterns from data. Introduced by McCulloch and Pitts in 1943 [141], the early concept of a neuron modeled a

weighted summation of inputs followed by a threshold-based activation. This simple idea laid the groundwork for Artificial Neural Networks. A breakthrough came with the perceptron (1958) [181], a single-layer neural network capable of solving basic pattern recognition tasks (Figure 1.2b)). However, perceptrons were limited to linearly separable problems, as noted by Minsky and Papert (1969) [144], leading to a temporary decline in interest.

The introduction of the multi-layer perceptron (MLP) and the development of backpropagation [183] revitalized the field by enabling efficient training of deeper networks (Figure 1.2 c)). Backpropagation uses the chain rule to compute gradients and update weights, solving the optimization problems posed by multi-layer architectures. Neural networks evolved further with the advent of convolutional neural networks (CNNs) [114], which exploited local connectivity and weight sharing to process high-dimensional data like images efficiently (Figure 1.2 d)).

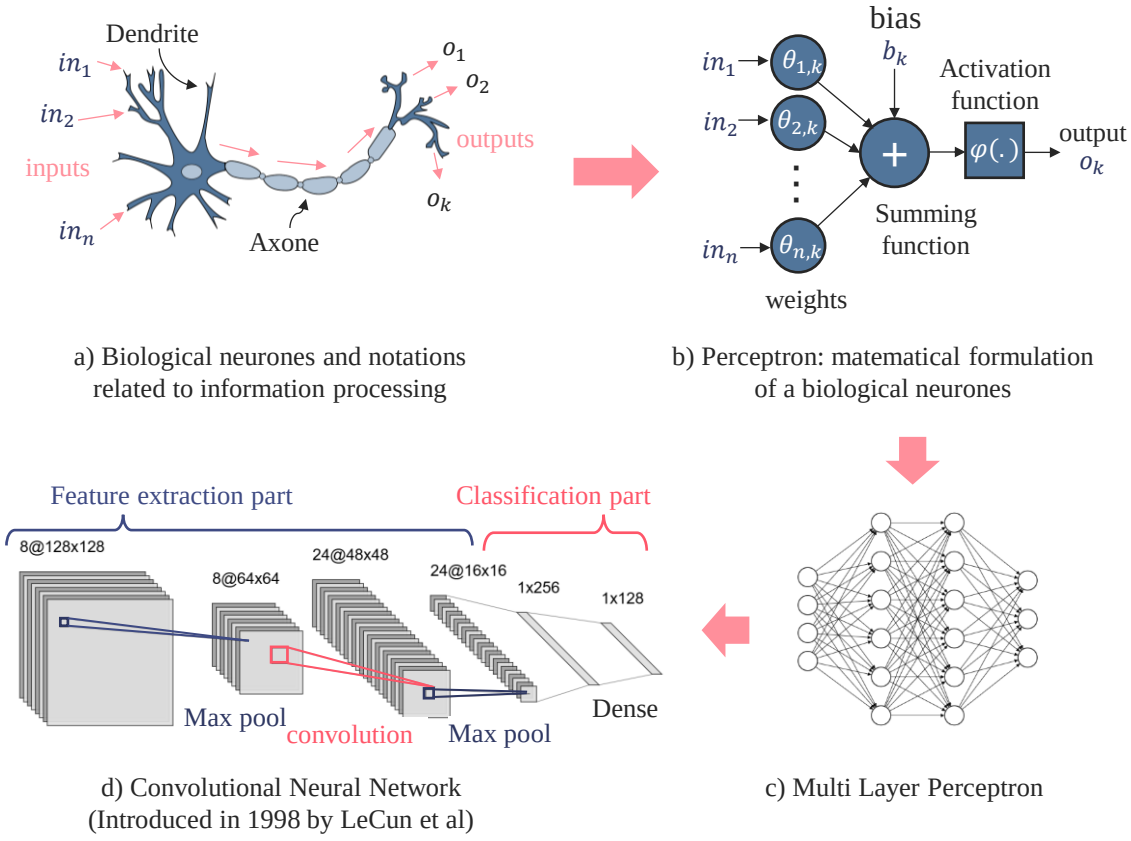
A CNN typically consists of two main components: the **feature extractor part** and the **classification part**. The feature extractor comprises multiple convolutional layers and pooling operations that progressively capture spatial hierarchies and invariant representations from input images. The fundamental operation in this stage is the **convolution**, where small learnable filters (kernels) slide across the input, computing feature maps by extracting edges, textures, and complex structures at deeper layers. To further reduce spatial dimensions and enhance translational invariance, CNNs use **max pooling**, a downsampling technique that retains the most significant feature responses in local regions while discarding less relevant information.

Once the feature extraction is complete, the output feature maps are flattened and passed into the **classification part**, which consists of one or more **dense layers** (fully connected layers). These layers perform high-level reasoning by learning complex relationships between extracted features and output classes. The final layer typically applies a softmax activation function for multi-class classification or a sigmoid activation for binary classification, producing the network's predictions. For more details on convolution, backprogation algorithm and mathematical foundation behind CNNs we refer to the work of Goodfellow et al [64].

The deep learning revolution

The renew interest in deep learning during the 2000s was fueled by a confluence of advancements in computational power, data availability, and algorithmic innovations. The emergence of graphics processing units (GPUs) revolutionized training processes by enabling efficient parallel computation, significantly accelerating the development of large-scale neural networks [45]. Simultaneously, the availability of large datasets such as ImageNet[50] provided millions of labeled examples, offering the diversity and volume necessary to train deep models effectively. Furthermore, innovations in neural network design and optimization helped closing the gap. Techniques like ReLU activation functions [151] mitigated the vanishing gradient problem, while batch normalization [88] stabilized training by normalizing intermediate activations. Additionally, dropout regularization [200] helped reduce overfitting, enabling neural networks to generalize better to unseen data. These collective advancements laid the foundation for the remarkable progress in deep learning observed over the past two decades.

The success of AlexNet[104] in the ImageNet competition demonstrated the potential of DNNs to achieve human-like performance on complex tasks, initiating a wave of innova-



a) Biological neurones and notations related to information processing

b) Perceptron: matematical formulation of a biological neurones

c) Multi Layer Perceptron

d) Convolutional Neural Network (Introduced in 1998 by LeCun et al)

Figure 1.2: From biological neurone to convolutional neural network.

tion in architectures like VGG[197], ResNet[74], and Vision Transformers (ViT) [52]. These architectures became the backbone of applications such as image classification, object detection, and speech recognition.

Compact neural network design

Deep neural networks (DNNs) achieve state-of-the-art performance in vision tasks, but deploying them on edge devices presents substantial challenges. Modern DNNs often contain millions of parameters, demanding extensive computational resources for inference. The dense layers within these networks require matrix multiplications between weights and activations, consuming significant memory and computational power when using full-precision floating-point arithmetic [230].

To overcome these challenges, researchers have developed model compression techniques that reduce both the storage footprint and computational complexity of DNNs while preserving accuracy. The three primary approaches include:

- **Pruning:** This technique removes redundant weights or entire connections in a network to reduce its size without significantly compromising performance. By removing unnecessary computations, pruning accelerates inference and decreases memory requirements, making DNNs more efficient for edge deployment [71].

- **Layer factorization:** This method decomposes weight matrices or convolutions into more efficient representations, reducing the number of operations required for inference. Factorization techniques also enable hardware-friendly designs that improve execution efficiency on low-power devices.

- **Quantization:** This process reduces the precision of model parameters and activations, enabling computations in lower bit-width formats such as 8-bit, 4-bit, or even 1-bit. In von Neumann architectures, lower precision reduces memory bandwidth and accelerates operations by decreasing the amount of data transferred between the processor and memory, optimizing execution on traditional computing units such as CPUs, GPUs, and TPUs [221]. In non-von Neumann architectures, such as compute-in-memory (CIM) accelerators, lower precision enables a fundamental rethinking of the multiply-and-accumulate (MAC) unit, allowing for significantly more energy-efficient computation by leveraging in-memory processing and analog computing techniques [190, 231, 142]. Post-training quantization (PTQ) applies precision reduction after model training, while quantization-aware training (QAT) integrates quantization constraints during training to mitigate accuracy loss [90][102].

These three techniques—pruning, layer factorization, and quantization—are often combined to achieve optimal compression and efficiency. When applied together, they enable the deployment of high-performance DNNs on edge devices, striking a balance between computational efficiency and accuracy.

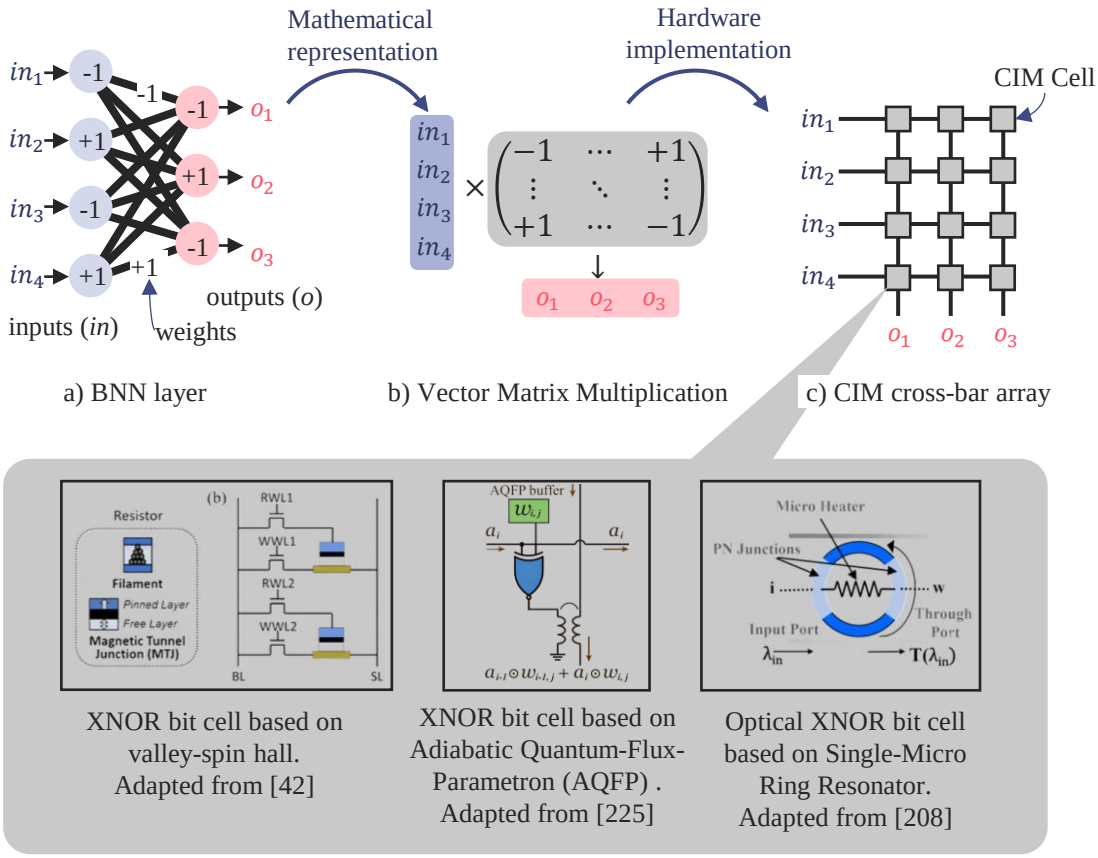
Binary neural networks

The most extreme form of quantization is found in Binary Neural Networks (BNNs), where weights and activations are constrained to just two possible values, typically +1 and -1 [85], as illustrated Figure 1.3 a). This drastic reduction not only minimizes the memory footprint but also fundamentally alters the operations types used during inference. The core operation at the heart of every ANNs layer is a vector matrix multiplication as represented Figure 1.3. Classically neural networks rely on energy-intensive multiply-accumulate (MAC) operators. BNNs simplify these computations by replacing multiplication with lightweight bitwise operations such as XNOR and bit-count. These binary inner products require fewer computational resources, reduce the surface area of the co-design accelerator and thus are significantly more efficient in terms of energy consumption, making BNNs highly suitable for energy-constrained systems [15], such as near-sensor inference in smart imagers.

Full binarization for hardware versatility

A key requirement for leveraging the potential of BNNs lies in full binarization, where both weights and activations are entirely binary throughout the network and all inference computations can be reduced to binary inner products. Such simplification enables hardware designs optimized exclusively for binary arithmetic, eliminating the need to support mixed-precision operations, which would increase complexity and energy usage.

Enforcing full binarization makes a wide range of hardware technologies viable, each tailored for binary inner product computations. Compute-in-Memory (CIM) architectures integrate computation directly into memory cells, reducing data movement and enabling



d) Examples of emerging CIM cells for cross-bar arrays enabled by BNNs

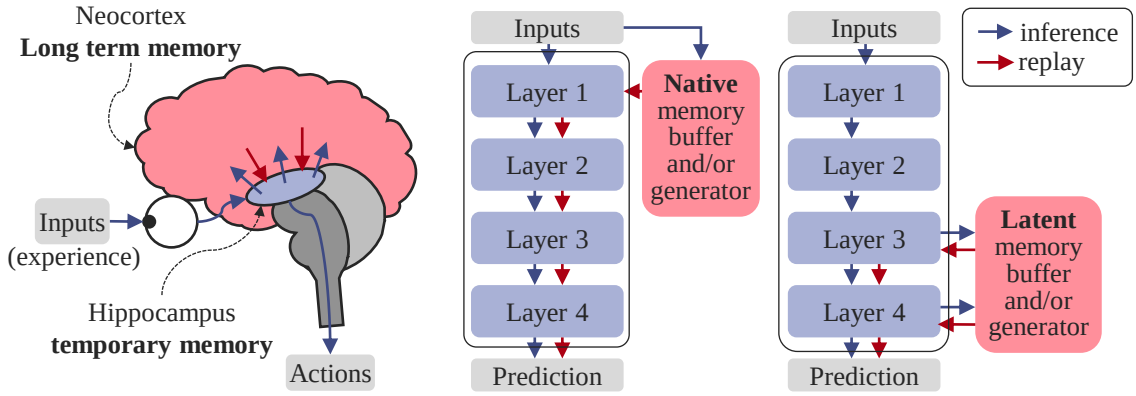
Figure 1.3: Advantages of binary neural networks.

highly efficient binary operations [236, 157, 5]. MRAM-based accelerators, such as STT-MRAM and SOT-MRAM, use binary-friendly non-volatile memory technologies that offer compact, energy-efficient designs ideal for binary arithmetic [32, 149, 42]. Additionally, emerging platforms like photonic computing [202, 208], phase-change memory systems [193], and superconducting quantum flux processors [225] show that binary-only inference can extend beyond CMOS, opening up opportunities for novel architectures. Figure 1.3 illustrates such implementations in CIM cells.

Limitations of BNN

Despite their advantages, BNNs face significant challenges. The reduced representational capacity of binary weights makes training difficult, often requiring specialized techniques such as gradient scaling, binarization-aware loss functions, and novel training algorithms to converge effectively [21]. Moreover, BNNs typically experience a drop in accuracy compared to their full-precision counterparts, especially when applied to complex datasets or tasks requiring fine-grained decisions [85][28]. Addressing these limitations is crucial to expanding the applicability of BNNs in real-world scenarios.

1.1.3 Replay for learning in dynamic environment



a) Replay in biological neural network

b) Native replay in deep neural network

c) Latent replay in deep neural network

Figure 1.4: The complementary learning system in biological networks as motivation for the replay mechanism in deep neural networks, illustrated with a four-layer structure.

Smart sensors, such as smart imagers, operate in dynamic environments where data distributions evolve over time. This non-stationary nature of real-world data requires artificial intelligence systems to not only perform efficient inference but also to learn and adapt incrementally as new patterns emerge [160]. For instance, a surveillance camera in a home security system might encounter novel objects, such as delivery drones or new household pets, that were not part of its original training set. In such scenarios, incremental learning is essential to ensure that AI models remain relevant and effective over time while maintaining efficiency in resource-constrained edge devices[72].

Unlike humans, DNNs struggle with incremental learning due to a phenomenon known as catastrophic forgetting[140]. When trained sequentially on new data, neural networks tend to overwrite previous knowledge, leading to the loss of earlier learned information. This occurs because learning involves updating network weights, and modifying weights crucial for past knowledge results in forgetting. In contrast, humans can accumulate knowledge throughout their lifetime, rarely suffering from such severe forgetting [60]. Research in the neuroscience community suggests that replay mechanisms in the brain play a fundamental role in enabling humans to retain and consolidate long-term memories. According to the complementary learning systems (CLS) theory, the brain relies on two distinct systems: the hippocampus, which enables rapid encoding of new experiences, and the neocortex, which gradually integrates knowledge over time[138]. The hippocampus, a critical brain region for memory consolidation, reactivates past neural activation patterns during sleep and wakefulness, facilitating the transfer of short-term memories to long-term storage in the neocortex [106][139]. Figure 1.4 a) illustrates this biological replay mechanism, which has inspired artificial methods for mitigating catastrophic forgetting.

In artificial neural networks, researchers have leveraged replay techniques to maintain past knowledge by revisiting previously encountered data during training [179][35][29][161]. One approach, known as native replay, stores a subset of past examples in a memory buffer

and interleaves them with new data during training. By directly revisiting stored experiences, the network preserves previous representations while learning new ones. However, maintaining even a small memory buffer introduces overhead that conflicts with the stringent storage and energy constraints of edge devices, making this approach challenging for resource-efficient systems [220] [118]. Figure 1.4 b) represents this method of native replay, which directly stores and replays past inputs.

To overcome memory limitations, latent replay has emerged as an alternative strategy inspired by the biological replay process observed in the hippocampus [205][209]. Instead of storing raw past data, latent replay replays internal, lower-dimensional, representations of previous experiences, extracted by the first layers of the networks [161].

Pseudo replay approaches proposed to encode and generate those latent representation using models such as Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs) or other statistical models [196][150][241]. By eliminating the need to store full datasets, latent replay significantly reduces memory overhead and better suits memory-constrained devices like smart sensors [89]. Figure 1.4 c) illustrates this approach, where processed representations are replayed within the network.

Replay-based learning remains one of the most effective strategies for overcoming catastrophic forgetting in neural networks [54][73]. Embedding long-term memory and leveraging pseudo-replay from latent space offer a promising path to more efficient incremental learning, particularly for edge devices where storage is limited. However, this requires addressing algorithmic and hardware constraints to ensure computational efficiency and learning performance.

1.2 Challenges and opportunities

Among the various compression methods, 1-bit quantization introduces unique challenges in the context of incremental learning. The extreme reduction in quantization, while advantageous for memory and computational efficiency, significantly reduces the expressiveness of the feature extractor, making it harder for the network to learn rich and discriminative representations. Additionally, BNNs are harder to train compared to DNN, even in standard offline training settings, due to the discrete nature of weights and activations. These factors compound the difficulty of applying BNNs in incremental learning, where the network must be retrained sevral times. At the beginning of this thesis, BNNs had not been systematically studied in the context of incremental learning, with only two articles dealing with quantization on incremental performance [83][107]. To the best of our knowledge, no work have been conducted on FBNNs and incremental learning.

1.2.1 Challenges: pseudo replay on binary neural networks

Among various incremental learning techniques, pseudo replay stands out as a promising approach for edge applications due to its ability to mitigate catastrophic forgetting without requiring access to original training data. However, adapting pseudo replay to BNNs faces two main challenges.

One key challenge is designing an effective compact binary feature extractor. Most successful pseudo replay methods rely on generating pseudo-examplars in a latent space defined by a fixed feature extractor. In the context of BNNs, feature extractors are constrained

by reduced representational ability and the quantization errors introduced by binarization, which limit the quality and diversity of generated examples. Achieving the necessary balance between compactness, efficiency, and accuracy in binary feature extraction remains a significant hurdle.

Another challenge lies in training a pseudo-examplar generator under edge constraints. Generative models like Generative Adversarial Networks (GANs) or Variational Auto Encoders (VAEs), used for pseudo replay [196][198], rely on high-precision operations for stable training and effective output generation, making their direct adaptation to BNNs difficult. Furthermore, the use of alternative, non-deep learning statistical methods for generating binary pseudo-examples has not been explored in incremental learning contexts. Investigating these approaches may provide a pathway to more efficient and robust pseudo replay for BNNs, particularly in resource-constrained edge environments.

1.2.2 Opportunities: incremental study of replay methods

Despite these challenges, replay methods offer a progressive framework to systematically explore incremental learning on FBNNs:

- **Native replay:** This baseline approach involves storing and replaying raw input data. Native replay allows for an initial understanding of how FBNNs manage incremental learning while maintaining accuracy and addressing catastrophic forgetting.
- **Latent replay:** Transitioning to storing latent representations instead of raw data significantly reduces memory usage. By replaying compact, encoded features rather than full inputs, latent replay provides a memory-efficient alternative while retaining the key characteristics of past examples. It allows to evaluate the capacity of the first BNN layers to extract meaningful and transferable features across tasks.
- **Pseudo replay:** Pseudo replay eliminates the need to store samples entirely by generating synthetic data. Leveraging the binary nature of weights and activations in FBNNs, pseudo replay offers the potential for ultra-efficient incremental learning systems. However, exploring this approach requires addressing both data generation quality and its compatibility with FBNN training.

This progressive methodology allows to incrementally tackle the challenges of replay in FBNNs, offering a clear path to evaluate and improve their compatibility with memory-efficient incremental learning. By moving from native replay to latent replay and finally to pseudo replay, this approach enables a step-by-step investigation of the adaptability and scalability of FBNNs with incremental learning.

This structured exploration not only addresses the challenges of pseudo replay but also opens the door to novel hardware-friendly learning systems. As FBNNs continue to evolve, replay methods, particularly pseudo replay, represent a promising avenue for enabling compact and efficient incremental learning tailored for edge applications.

1.3 Contribution overview

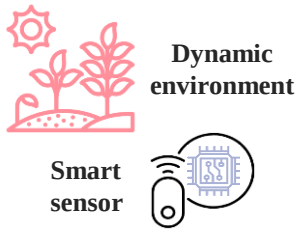
This thesis makes advances in BNN and incremental learning, addressing the mentioned challenges and introducing innovative methodologies. The main contributions, detailed in

Chapters 3 and 4, are summarized as follows:

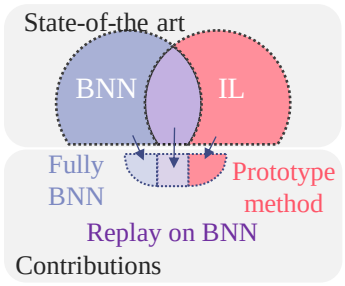
- **Design of fully binarized neural networks:**
  - Propose a baseline fully binarized neural network for continual learning,entirely eliminating floating-point operations and achieving state-of-the-art performance on offline benchmark.
  - Introduce BNN architectural innovations, including layer-wise scaling normalization, non-linear classifier, and thermometer encoding for input preprocessing.
  - Validated the approach on CIFAR-100 and CORE50 datasets, demonstrating comparable performance to state-of-the-art BNNs with minimal memory footprints.
- **Loss balancing for incremental learning in BNNs:** Investigated loss balancing strategies to address the trade-off between adaptability and retention in continual learning.
- **Semi-supervised pretraining for latent replay:** Developed a semi-supervised pretraining strategy combining Barlow Twins [233] and custom activation regularization to enhance feature transferability, improving accuracy in latent replay scenarios.
- **Generative binary memory:**
  - Introduced a novel pseudo-replay method leveraging Bernoulli Mixture Models (BMMs) to model binary embedding, enabling memory-efficient continual learning.
  - Designed custom embedding binarizers to ensure compatibility of GBM with any deep neural network.
  - Demonstrated significant accuracy improvements and memory reductions compare to state-of-the-art on CORE50 and TinyImageNet benchmarks.
- **Iso-memory comparisons of replay mechanisms:** Conducted a comprehensive evaluation of replay methods under equal memory conditions:
  - **Native vs. latent replay:** Compared these approaches, highlighting their trade-offs in adaptability, retention, and memory efficiency. Native replay excels in adaptability with larger memory buffers, while latent replay performs better under tight memory constraints.
  - **Latent vs. pseudo replay:** Evaluated pseudo replay (GBM) against latent replay, showing that pseudo replay provides better accuracy and retention for small memory buffers, leveraging its generative capabilities.
- **Hardware-friendly implementation of GBM:** Proposed an efficient, fixed-point online Expectation-Maximization (EM) algorithm for BMMs, enabling real-time deployment of GBM on resource-constrained hardware platforms.

Through these contributions, this thesis establishes a robust framework for memory-efficient continual learning with binary neural networks.

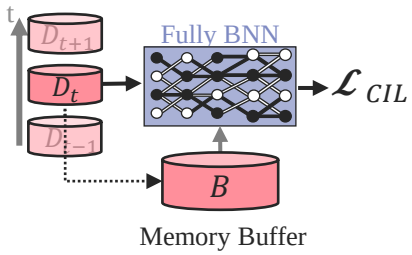
1.4 Manuscript organization



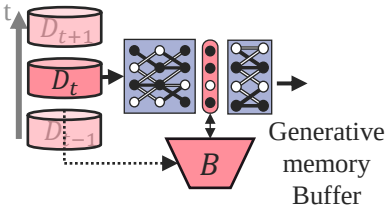
a) **Chap 1**: Smart sensors in dynamic environment. Motivation for low-power, incrementally learnable DNNs.



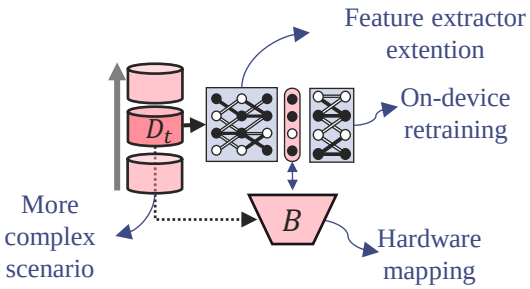
b) **Chap 2**: litterature review on BNN and Incremental Learning (IL) and scientific positioning.



c) **Chap 3**: Investigation of experience replay methods on fully binary neural network.



d) **Chap 4:** Proposal of a pseudo replay method tailored for BNNs and extendable to DNNs and RTL mapping.



d) **Chap 5:** Conclusion and research perspectives.

Figure 1.5: Schematic outline of the manuscript by chapter.

This manuscript is organized into five chapters, progressively addressing the challenges and opportunities of incremental learning in Fully Binary Neural Networks (FBNNs) and exploring novel replay methodologies.

Chapter 2 reviews the state-of-the-art in Binary Neural Networks (BNNs), focusing on fully binary inference and the associated challenges in design and training. It also surveys incremental learning methodologies, covering evaluation protocols, method taxonomies, and replay strategies, with an emphasis on their compatibility with binary networks. The chapter concludes by introducing the positioning used in the subsequent contributions chapters.

Chapter 3 proposes a baseline FBNN for class-incremental learning, establishing good practices for its training and evaluation. It investigates key aspects such as loss composition for pretraining and retraining, memory buffer design, and sample selection strategies to mitigate catastrophic forgetting. This chapter provides a foundational framework for studying FBNNs in incremental learning settings.

Chapter 4 builds upon the findings in Chapter 3 to develop a novel pseudo replay method, GBM, which exploits the binary nature of latent activations in FBNNs. By generating synthetic latent data, GBM significantly reduces memory requirements compared to traditional replay methods. This chapter compares GBM with native and latent replay approaches across FBNNs, BNNs, and DNNs, evaluating performance in iso-memory frameworks. Additionally, it explores the hardware feasibility of GBM, proposing early RTL-level designs that demonstrate its practical applicability in resource-constrained environments.

Chapter 5 concludes the thesis by summarizing the key contributions and offering perspectives on future research directions. It outlines short-, medium-, and long-term goals to advance the integration of compact, hardware-friendly neural networks with incremental learning techniques.

Highlights of the chapter

- Smart sensors, especially smart imagers, enhance remote signal processing by reducing bandwidth, latency, and energy costs but face challenges in human-like tasks, hardware constraints, and dynamic adaptation, requiring specialized neural network capable of low-power incremental learning.
- Fully Binary Neural Networks offer a promising solution for ultra-low power inference on emerging hardware but are difficult to scale up to the perf of their real-valued counter parts and even more challenging to train incrementally.
- Replay methods enable ANNs to learn incrementally, but their reliance on memory buffers raises challenges at the edge. Studying pseudo-replay methods alongside BNNs offers a promising alternative.

Chapter 2

Fully Binary Neural Network and Incremental Learning: Methodology and State-of-the art.

Introduction

In Chapter 1, we discussed how sensing systems can benefit from decision-making algorithms deployed near sensors. The constraints imposed by hardware and application requirements necessitate ultra-low-power inference models that are capable of adaptive learning while handling high-dimensional and complex data distributions. These constraints motivate our research on BNNs within the framework of incremental learning. This chapter provides a literature review covering these two domains and their intersection.

BNNs enable energy-efficient inference, with power consumption in the mW range on specialized integrated circuits thanks to reduced precision arithmetic. However, this efficiency comes at the cost of decreased performance and a more complex training process compared to floating-point networks. In Section 2.1, we introduce the necessary background and optimization formulations that allow BNNs to converge under discretization constraints, as used in this thesis. We then review methods to improve BNN classification performance and analyze them in the context of always-on applications. Many of these approaches focus on inference on generic computing platforms, which offer more flexibility than the hardware-constrained settings targeted in this thesis. Finally, we focus on research toward fully-binary neural networks, detailing our positioning within this research direction. As an opening, we also discuss emerging work on edge learning and alternative learning strategies for BNNs, and how our approach positioned with these developments.

Enabling BNNs to incrementally learn is a crucial step toward adaptive learning systems. Incremental learning refers to the ability to accumulate knowledge over time as new information is encountered. However, neural networks struggle with the stability-plasticity trade-off, leading to catastrophic forgetting when learning incrementally. In Section 2.2, we introduce the problem formulation and notations of incremental learning as used in the machine learning community. We then review incremental learning methodologies, with a focus on replay-based approaches, which have proven to be among the most effective techniques. We further examine emerging works at the intersection of BNNs and incremental learning, classifying them within the existing IL methodology taxonomy.

Finally, building on insights from this literature review, we identify key research gaps and position our thesis around three main areas:

- Advancing the development of fully binary neural network architectures.
- Improving the understanding of incremental learning within BNNs.
- Developing pseudo-replay strategies tailored for BNN-based incremental learning and extendable to any DNNs.

2.1 Binary neural networks

2.1.1 Background on BNNs

Binary Neural Networks (BNNs) are a specialized subset of Convolutional Neural Networks (CNNs) where the weights and activations of parametric layers, such as convolutional and dense layers, are constrained to binary values, typically -1 and $+1$. This binarization enables BNNs to replace computationally expensive multiply-accumulate (MAC) operations,

essential for matrix multiplications, with highly efficient binary operations (*e.g.*, XNOR and bit-counting) during inference as illustrated Figure 2.1. In theory, this results in a compute graph that relies exclusively on binary inner products, significantly reducing computational complexity, memory usage, and energy consumption. The reliance on canonical binary operations also makes BNNs particularly well-suited for always-on, low-power applications, paving the way for scalable hardware implementations.

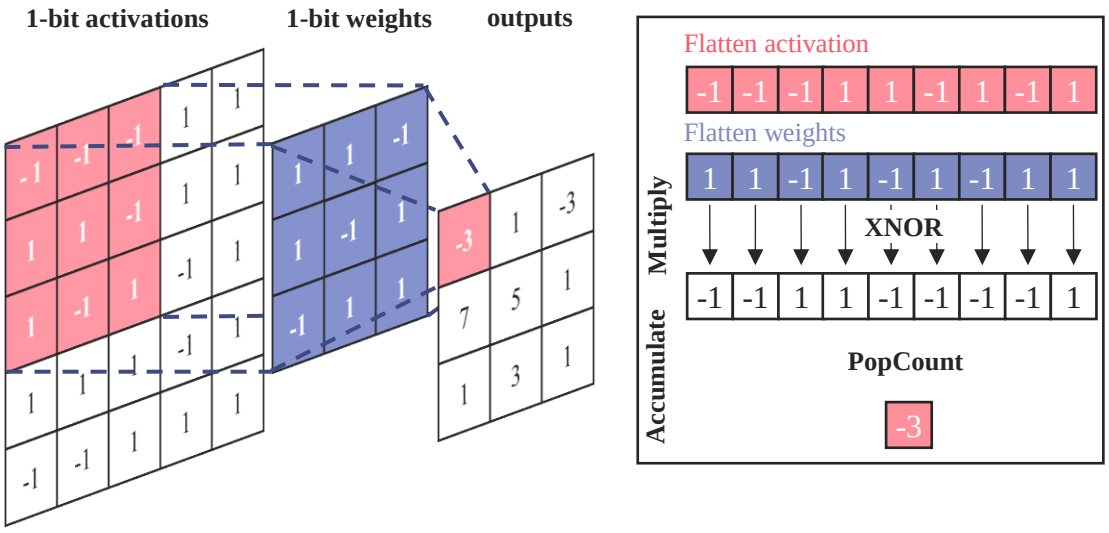


Figure 2.1: An example of multiply accumulate (MAC) operation in a binary convolution: a combinaison of bitwise XNOR and popcount. Note that the output is then binarized by only taking the sign bit.

Although BNNs offer promising theoretical efficiency, their binary nature introduces significant difficulties during training. These difficulties primarily stem from the non-differentiability of binary weights and their limited expressiveness, which complicate the use of standard backpropagation for optimization. Two factors contribute to this issue: First, gradient computation during the backward pass is limited because discrete binary multiplication prevents the use of conventional gradient-based methods. Second, weight representation constraints arise, as BNNs are restricted to 1-bit precision, making it impossible to accumulate gradients over weights as done in conventional neural networks with floating-point encoded weights.

Historical advances in BNN optimization

Efforts to address these challenges began with [199] who introduced Expectation Backpropagation, a probabilistic approach that replaced gradients with expectation-based updates to handle the non-differentiability of discrete weights. Subsequently, [57] focuses on energy-efficient backpropagation in neuromorphic systems, employing binary activations and weights to reduce power consumption while maintaining computational accuracy.

A major breakthrough came with BinaryConnect [46], which introduced weight binarization using the sign function as a quantizer. To enable gradient-based optimization

despite the non-differentiability of this function, the authors adopted the Straight-Through Estimator (STE) [20]. STE allows gradients to propagate through non-differentiable operations by treating the quantizer as the identity function during backpropagation. Activations were retained in full precision to preserve the network's representational capacity. Building on this approach, BinaryNet [85] extended binarization to both weights and activations, achieving significant gains in computational and memory efficiency while maintaining competitive accuracy.

This thesis focuses on developments derived from BinaryNet, which represents a significant portion of the BNN literature. Additionally, we acknowledge theoretical advancements over the STE for backpropagation, such as the Mirror Descent View [3], which frames optimization through quantization as a dual problem, and ProxQuant [14], which uses proximal optimization to enhance training stability and convergence for quantized networks.

Quantization-aware training

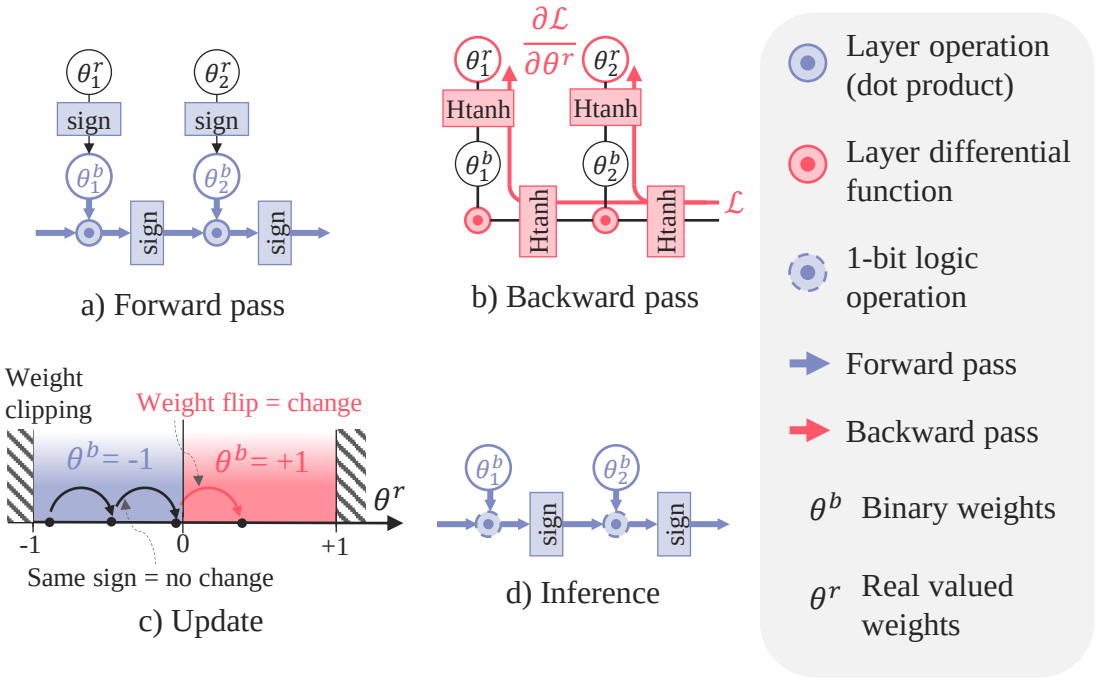


Figure 2.2: Backpropagation adaptation for BNN training.

BinaryNet [85] introduced a Quantization-Aware Training (QAT) procedure, which adapts the backpropagation algorithm for full-precision values by incorporating quantization layers. QAT addresses the challenges of gradient-based methods in the context of quantization by modifying the three phases of the training process (see Figure 2.2):

Forward pass: During the forward pass, each full-precision value (x^r), whether weights or activations is passed through a sign function (Sign) (Equation 2.1) to emulate binary operations. This ensures the loss is evaluated under conditions that closely mimic inference.

$$x^b = \begin{cases} +1, & \text{if } x^r \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{2.1}$$

Backward pass: Since the Sign function is non-differentiable, its derivative is approximated using a clipped identity function, also known as Hard Tanh ($Htanh$) (Equation 2.2).

$$\frac{\partial \mathcal{L}}{\partial \theta^r} \simeq \frac{\partial \mathcal{L}}{\partial \theta^b} \mathbb{1}_{[-1,+1]} \tag{2.2}$$

Update: Binary weights (θ^b) do not support gradient accumulation due to their discrete nature. Instead, updates are applied to the underlying full-precision values (θ^r), allowing the use of standard optimizers and regularizers.

Inference: Once training is complete, the binarized version of the full-precision weights is used for inference on the same computational graph, utilizing boolean operations.

Floating-point precision as convergence enablers

In their seminal work, BinaryNet [85] proposed retaining certain layers in full precision to address the performance gap between BNNs and conventional CNNs. These design choices have been widely adopted in subsequent research, presented in the next sections, as they significantly improve training stability and accuracy:

- **Full-Precision first convolution**: The first convolutional layer is retained in full precision to ensure the extraction of rich and detailed features from the input data. Early quantization could otherwise result in a loss of critical information, undermining the network's performance.

- **Full-precision last layer**: The final layer is preserved in full precision to enhance the network's representational capacity, particularly for classification tasks. This layer ensures accurate output predictions while compensating for the limitations of binary weights in earlier layers.

- **Batch normalization layers**: Batch normalization layers, which inherently involve multiplication operations, are kept in full precision to stabilize and accelerate training. Additionally, these layers help mitigate the scale variance of binary weights and introduce normalization noise, which may regularize the model. A shift-based batch normalization approach to reduce the number of multiplications, but these operations are still present.

Figure 2.3 shows a ResNet18 architecture and highlights where full-precision layers remain. While these practices improve classification accuracy with some computational overhead, they introduce challenges for hardware implementation. Specifically, the inclusion of full-precision operations breaks the promise of a fully binary compute graph, necessitating a more complex, thus less efficient, mixed-precision hardware.

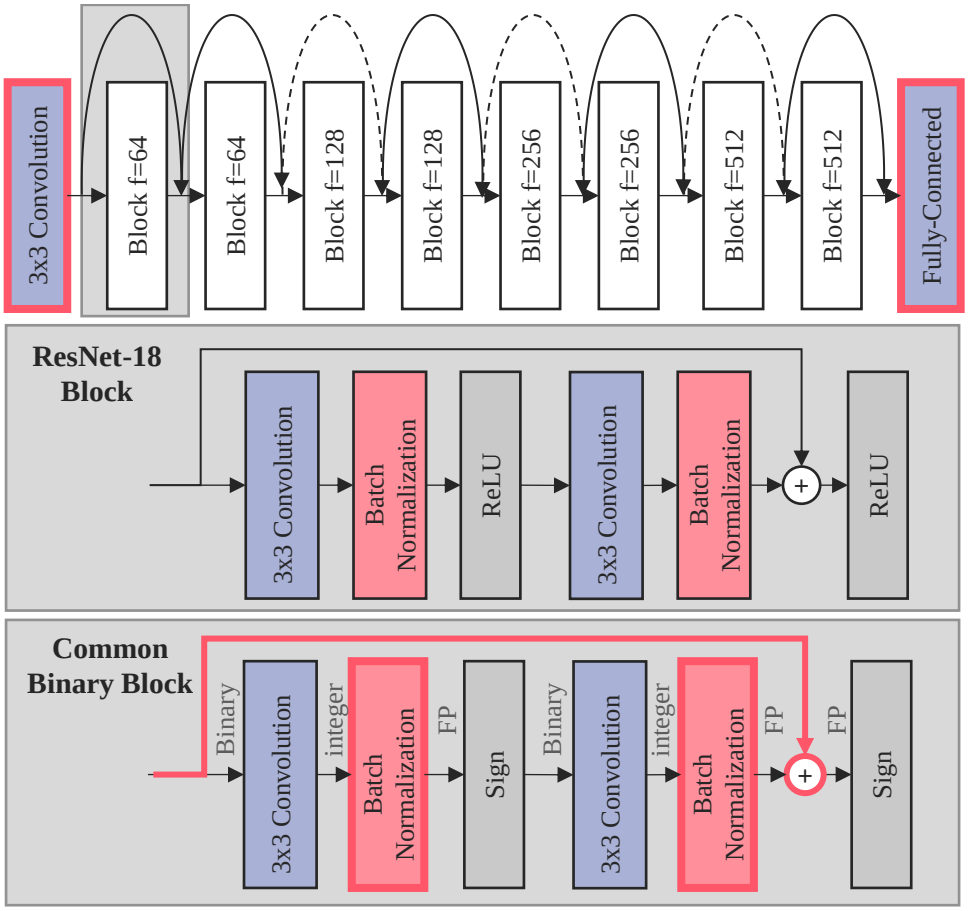


Figure 2.3: ResNet-18 architecture consists of multiple convolutional blocks, each comprising two convolutional layers with varying numbers of input channels, denoted by f. Dashed residual lines indicate downsampling, reducing the height and width of the feature map by a factor of 2. Additionally, a representative block for BNNs during training is shown. Red lines highlight components that include non-binary operations during inference.

2.1.2 Overview of BNN design and trainings strategies

Despite its groundbreaking introduction in 2016, BinaryNet [85] exhibits several limitations, particularly in terms of classification performance when compared to full-precision neural networks. These limitations arise due to the inherent challenges of reducing precision to a binary format, which compromises the network's overall representational and computational capabilities.

Over the years, a significant body of work has reviewed these challenges and proposed algorithmic improvements, with numerous surveys comprehensively analyzing advancements in this field [170, 189]. The limitations of BinaryNet can be broadly categorized into three main areas:

- **Quantization error**: The process of binarization introduces substantial quantization errors, leading to a loss in numerical precision and degraded network performance.
- **Loss of expressivity**: Binary weights and activations significantly reduce the rep-

resentational capacity of the network, limiting its ability to model complex patterns.

- **Training instabilities**: The non-differentiable nature of binarization and the constrained parameter space often result in unstable and suboptimal training.

To address these limitations, researchers have proposed various improvements that can be categorized into three broad strategies, as presented Figure 2.4: (1) **Binarization strategies**, *i.e.*, techniques aimed at reducing quantization errors, such as improved binarization functions and hybrid precision approaches; (2) **Optimization strategies**, *i.e.*, advances in optimization techniques, including better loss functions, gradient approximations, and regularization methods, to enhance training stability; (3) **Architecture improvements**, *i.e.*, Modifications of network architectures to compensate for the reduced expressiveness of binary representations.

In the next section, these improvements have been systematically summarized and analyzed in the context of their applicability to always-on applications, where efficiency, energy consumption, and robustness are critical considerations.

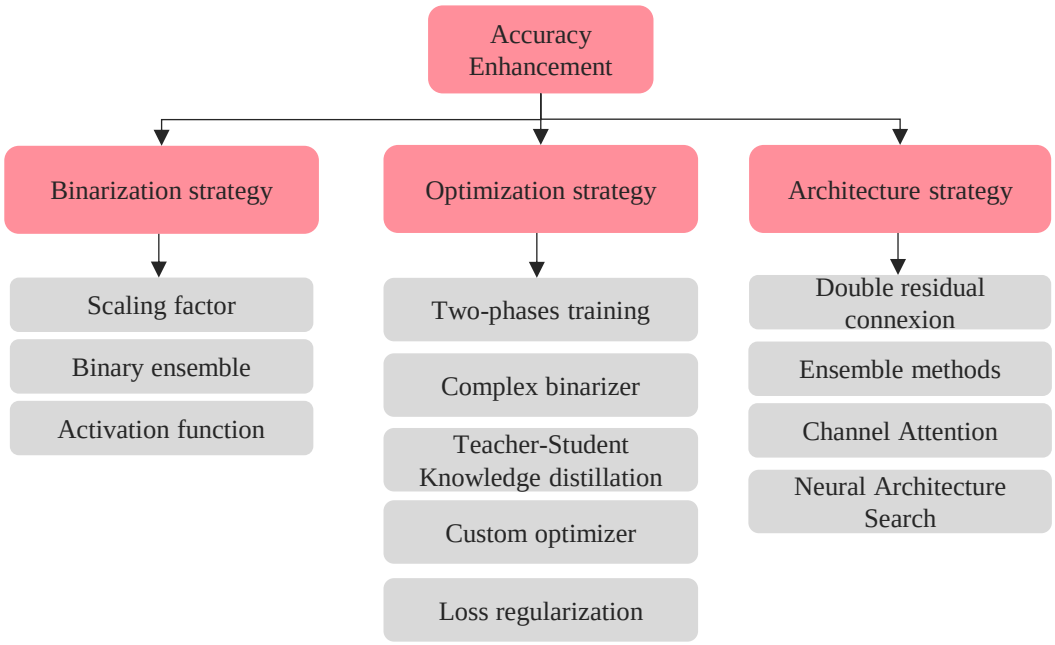


Figure 2.4: Taxonomy of classfication performance enhancement for BNN proposed in the computer vision literature community.

Binarization strategy

The quantization error introduced by sign binarization has been addressed in the literature through various enhancements to binarization techniques. XNOR-Net [174] introduces channel-wise scaling factors to minimize information loss when converting 32-bit values into 1-bit representations for both activations and weights. Building on this foundation, XNOR-Net++ [28] combines these scaling factors into a single learned parameter optimized via backpropagation. These approaches successfully extended BinaryNet to larger datasets

such as *ImageNet*, albeit at the cost of incorporating full-precision operations in each convolutional layer.

In contrast, ABC-Nets [122] adopts a strictly-binary-weights approach by employing 3 to 5 binary weight bases to approximate full-precision weights. While this method increases model complexity and size, it narrows the accuracy gap between binary and full-precision networks to around 5% on *ImageNet*.

Other methods focus reduce the approximation with more complex non-linear activation functions. For example, POKEBNN [237] and BNEXT [69] introduce clipping bounds to refine the activation function. REACTNET [126] proposes the *rsign* function as a flexible alternative to the standard sign function and introduces the *RPReLU* activation to mitigate distributional shifts in activations and weights, thereby enhancing overall performance.

Optimization strategy

Recent research has advanced optimization strategies for BNNs by addressing challenges such as gradient mismatch, improving loss functions, and refining gradient approximation methods. These efforts aim to counter the limitations introduced by binarization.

One common approach is multi-stage training, in which a full-precision model is trained first, and binarization is applied in subsequent stages [237, 69, 124]. This method improves performance but increases training time and computational requirements.

Another popular technique in model compression is knowledge distillation[79], where a low-precision student network is trained under the guidance of a pre-trained full-precision teacher model [145]. This technique have been adopted for BNNs [117]. More advanced methods, such as BNEXT [69], employ multiple teacher models to achieve state-of-the-art (80% top-1 accuracy on *ImageNet*), although these approaches often demand significant computational resources.

The STE is a widely used method for approximating gradients during training. However, STE struggles to update weights near the boundaries of -1 and $+1$, impairing the backpropagation process. To address this, alternative gradient approximation techniques have been proposed [187, 224]. For instance, Bi-Real Net's ApproxSign function [124] introduces a polynomial step function to approximate the forward sign function, reducing gradient errors and improving training stability.

Moreover, specialized optimizers have been developed for BNNs. For example, Helwegen et al. [77] proposed optimizers tailored for binary networks. Yet, studies such as [125] highlight the importance of second-order momentum, as in Adam, for BNN convergence.

Lastly, regularization techniques have been developed to aid training by adding loss terms that address specific challenges in BNNs. For example, distributional losses [51] mitigate issues such as degeneration, saturation, and gradient mismatch, while specialized weight regularizers [47] encourage proxy weights to align closely with binary targets. Additionally, entropy-maximizing regularization terms [87] have been introduced to maximize the information content of binary weights. These regularization techniques are particularly advantageous for lightweight inference, as they improve training without adding computational overhead during inference.

Architecture improvement

Advancements in BNN architectures have drawn inspiration from the progress made in CNNs. Many BNN designs are based on well-established layouts ResNet [74] and MobilNet [81], which offer strong performance and scalability. While other compact CNN architectures, such as ShuffleNet [131], have been developed as more efficient alternatives to MobileNet in terms of FLOPs with only a slight accuracy tradeoff on *ImageNet* [50], they have not been widely adopted in the BNN field. This suggests that further exploration of underutilized architectures in binary settings could provide additional opportunities for efficiency and performance gains.

One major area of improvement in BNN architecture lies in modifying basic building blocks. Bi-Real Net [124], for example, incorporates residual connections after each convolutional layer to facilitate information flow. However, these shortcuts operate in full precision. BinaryDenseNet [23] introduces specialized dense blocks that use concatenated residual connections. This modification enhances network expressiveness, according to the authors. Similarly, MeliusNet [24] adapts BinaryDenseNet's structure to a MobileNetV1 [81] backbone, balancing efficiency and accuracy for binary operations.

Other advancements in BNN architectures focus on improving representational power through attention mechanisms. Real-to-Bin [137] and PokeNet [237] utilize variations of squeeze-and-excitation mechanisms [82], which are designed to perform channel-wise attention and enhance the network's expressive capacity. While effective, these methods often introduce mixed-precision operations, as seen in PokeNet, which employs 4-bit arithmetic. QuickNet [16] further extends the work of BinaryDenseNet and Real-to-Bin by adopting smaller convolutional kernels in the input layer and using pointwise convolutions within its blocks. This design emphasizes both accuracy and latency, addressing key performance metrics for real-world applications. On the other hand, PokeNet reduces the number of multiply-accumulate (MAC) operations by also modifying the first layer with an increased stride and replacing pointwise convolutions with computationally efficient reshape-and-add operations.

A distinct line of research in BNN architecture improvements leverages ensemble-based methods to achieve better performance. These approaches rely on using multiple network instances or components to refine predictions. For example, ABC-Net [122] aggregates predictions at the per-layer level, which improves accuracy but requires significant computational resources. GroupNet [245], by contrast, aggregates predictions at the block level, reducing the number of operations while maintaining competitive performance. BENN [244] takes a different approach by combining multiple BNNs with boosting techniques, leveraging the fact that the reduced precision of 1-bit weights allows for a greater number of parameters within the same memory footprint. This approach highlights the importance of comparing memory-equivalent networks rather than simply matching architectural designs with non-binarized models.

A more recent direction in BNN architecture research involves the application of Neural Architecture Search (NAS) [56] to optimize binary networks for specific performance metrics, such as accuracy, latency, and energy efficiency [98][240]. These NAS-driven approaches represent a promising avenue for automatically discovering architectures that are well-suited for binary networks in conjunction with specific hardware plateforms.

Discussion regarding always-On applications

This review of advancements in BNNs highlights significant progress in scaling binary networks for large datasets, largely driven by the computer vision community. However, many of these improvements depend on mixed-precision or full-precision operations, which are incompatible with the requirements of always-on applications. While such operations are feasible on reprogrammable hardware, they pose challenges for designing efficient co-accelerators tailored to power-efficient, fully binary inference, a necessity for our target application.

In terms of inference compute graphs, techniques such as weight approximations, full-precision shortcuts, and channel attention mechanisms have proven effective for enhancing accuracy. However, these approaches rely on non-binary arithmetic, making them unsuitable for compact, fully binary neural networks. Architecturally, residual connections have demonstrated their utility in improving performance for deeper networks. BinaryDenseNet [23], for example, avoids mixed-precision operations by employing concatenative layer combinations that expand activation feature maps, albeit at the cost of increased network size. To address this, grouped convolutions, as proposed in ShuffleNet [131], offer a promising solution by reducing parameter overhead while maintaining efficiency, an avenue worth exploring in the context of fully binary networks.

Optimization strategies present additional challenges. While multistage training and knowledge distillation do not alter the inference compute graph, they often require longer training durations or larger teacher models, complicating the feasibility of retraining at the edge for incremental learning scenarios. By contrast, loss regularization emerges as a promising approach, offering performance improvements without altering the inference graph or introducing additional computational overhead during training, provided the loss terms remain simple. As demonstrated in [203], such techniques are particularly well-suited to compact network designs.

Furthermore, [22] demonstrates that carefully designed networks, combined with straightforward training procedures, without gradient approximations and multistage training, can achieve results comparable to or exceeding those of more complex methods. These findings align closely with our objective of developing architectures optimized for low-power inference while ensuring robust and simplified training processes.

A critical limitation of most prior work is their reliance on full-precision operations in the first layer, last layer, and batch normalization. This dependence raises significant questions about whether the proposed advancements in the literature can be effectively scaled to fully binary compute graphs, particularly for always-on applications.

In summary, the development of BNN models within the computer vision community has increasingly diverged from the disruptive potential of fully binary approaches, particularly in terms of enabling efficient hardware implementation. There remains a notable gap in the literature regarding fully binary, compact models. Analyzing such models in an incremental learning context could provide valuable insights into their feasibility for hardware mapping. Among the reviewed methods, loss regularization and careful network design stand out as the most promising strategies for successfully training fully binary networks. In the next section, we turn our focus to approaches that eliminate full-precision operations, specifically by targeting fully binary first and last layers and removing batch normalization, thereby addressing the limitations of current methods.

2.1.3 Fully-binary neural network enablers

Binarization of the first and last layers is often excluded in BNN models, as it typically results in a significant increase in error. Similarly, batch normalization removal poses challenges due to its critical role in stabilizing training and improving convergence. This section examines approaches that address the binarization of the first and last layers and the removal of batch normalization, either individually or in combination, with the goal of achieving fully binary neural networks.

Input layer

To eliminate the need for mixed-precision operations between non-quantized input signals and the binary weights of a BNN's first layer, the literature proposes various methods for encoding input data in binary form. These methods aim to preserve accuracy while enabling fully binary operations.

Penkovsky et al. [162] propose using stochastic computing to binarize input images. This method transforms the three color channels of CIFAR-10 images into over 1,500 binary channels, significantly increasing the parameter in the input layer by approximately 16 times. Although effective, this approach introduces a substantial computational overhead.

The Binary Input Layer approach [55] investigates an alternative using an 8-bit fixed-point representation, which is directly interpreted as binary code without requiring weights. While this method performs well on multimodal datasets, it incurs a 4% increase in validation error on *CIFAR10*.

The ILB (Input Layer Binarization) method [211] offers another approach by employing 8-bit bit-plane encoding combined with re-weighting, achieving binarization of the first layer. This method demonstrates promising results while addressing some of the challenges posed by prior techniques.

FracBNN [238] adopts thermometer encoding to represent input data. However, this approach is sparsely encoded, with many potential codes unused during conversion, leading to encoding inefficiencies. Bankman et al. [15] also utilize thermometer encoding, but apply it to uniformly quantized 7-bit input data. Their approach converts a 3-channel 8-bit image into a 255-channel 1-bit tensor (3×85 levels). This method not only achieves binary encoding but also demonstrates the potential for co-designing a BNN accelerator with Compute-In-Memory (CIM) hardware.

In our work, we extend the thermometer encoding approach introduced by [15] by applying it to the YCbCr color space. This extension, detailed in Section 3.3.4, allows for binary-only arithmetic while maintaining a relatively low number of operations in the first layer. Furthermore, we demonstrate the advantages of this method for replay-based learning scenarios, particularly in terms of reduced memory usage.

Batch normalization

Batch Normalization (BN [88]) is a parametric layer that is often seen as essential for high-performing BNNs. BN eases model convergence by properly scaling the values in forward and backward passes. However BN yet relies on full-precision operations that cannot be easily binarized.

BNN-BN [39] propose to remove BN and (1) integrates an adaptive gradient clipping [26] to mitigate gradient explosion and (2) normalize proxy-weights during training to ensure consistent variance and prevent mean shifts in activation distributions.

AB-BNN [132] further improve the work of BNN-BN [39] that still requires MAC operations. They improve the baseline network block with mask layer and specifically design activation function, quantized RPReLU. The quantized RPReLU structure enables more efficient bit operations by constraining its slope to be integer powers of 2. They bring closer BNNs to bit-operation-only neural network. However they do not quantize the first and last layers.

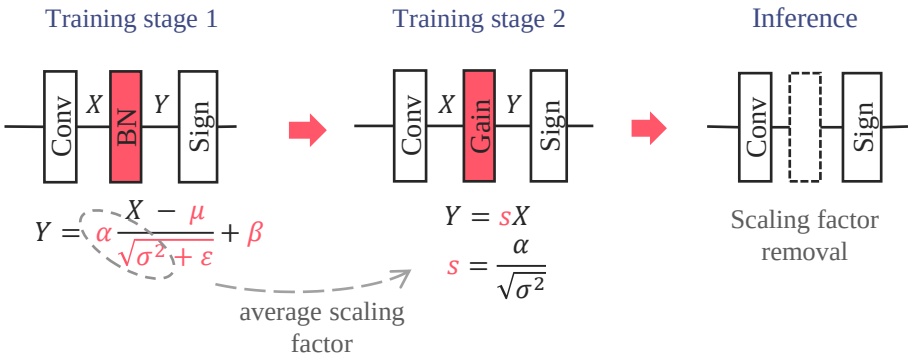


Figure 2.5: QAT multi-stage training for BatchNormalization removal.

Another typical training strategy consists in doing a QAT multi-stage training to progressivily binarize the network, and progressively replace Batch Normalization layers by per-layer scaling factors [154]. Here-under an example of such training procedure:

1. train with quantized weights, quantized activations, and BN.
2. train with quantized weights, quantized activation and scaling factors initialized from BN parameters.

On the last stage, the scaling factors can be computed as an average or median of the estimated moving averages and internal gain parameters. Offsets and biases are discarded as adding values to a binary +1/-1 may flip the sign and introduce large errors. The role of the scaling factors is to help to converge the training when the forward pass is performed only with binary values. The scaling factors are hence removed for inference, since it has no impact on the SIGN function.

Last layer

The last layer in BNNs is typically kept in full precision, as binarizing it introduces non-binary operations and poses challenges for incremental learning scenarios. In latent replay-based approaches, for instance, only the last layer is retrained during incremental updates (discussed in Section 2.2.2).

Tang et al. [203] address the challenge of binarizing the last layer by introducing an adaptive scaling parameter applied post-binarization. This scaling layer mitigates extreme output variation caused by the large number of input channels, stabilizing training and preserving accuracy. Without this adjustment, binarizing the last layer leads to a significant performance drop, with top-5 accuracy decreasing from 65.6% to 61.0%. However, incorporating the scaling layer improves accuracy to 64.6%, highlighting its effectiveness

In our work, we build upon this concept by introducing an adaptive scaling parameter tailored to incremental learning scenarios. Our empirical analysis reveals that the scaling factor depends on the number of classes, and we propose a heuristic to dynamically adjust this parameter as new classes are learned.

Summary and contributions

There is a motivation within the machine learning community to go toward fully binary neural networks, driven by the need for efficient and hardware-friendly models. However, existing works only partially address the removal of full-precision operations, and no approach eliminates them entirely on BNNs.

In this thesis, we propose a fully binary neural network baseline inspired by existing methods but tailored specifically for incremental learning and replay-based scenarios. To remove the full-precision first layer, we adapt thermometer coding techniques. For batch normalization removal, we modify architecture scaling factors to act as effective normalization mechanisms. Lastly, we introduce an adaptive output scaling factor, designed to account for class growth during incremental learning, enabling the removal of the full-precision last layer. These adaptations ensure compatibility with replay methods while addressing the limitations of prior approaches.

2.1.4 Perspective on BNN for adaptive sensors

The previous sections focused on the design and training of BNNs for deployment on resource-constrained platforms. An equally important consideration is on-chip training for adaptive smart sensors, particularly for adapting BNN models to handle data stream drift during deployment. This application requires learning strategies that can operate within tight resource constraints while effectively managing non-stationary data flows. Although on-chip training is not the focus of this thesis, we briefly review trends in the literature on BNN edge training before transitioning to alternative learning approaches, with a focus on incremental learning in BNNs.

Edge-training: Recent advancements have focused on minimizing computational and memory requirements in BNNs. Fully binarized neural networks, which extend binarization to include activations during the forward pass, significantly reduce inference complexity. However, the backward pass for training remains a key bottleneck due to its computational and memory intensity. The computational cost of the backward pass is typically 1.5x to 2x that of the forward pass because it requires additional steps such as weight gradient computation and error propagation through layers. The higher memory requirement arises from the need to store intermediate activations for gradient computation. Despite these challenges, reducing the cost of the backward pass has become a focal point of research, as it remains a significant limitation for resource-constrained edge platforms.

Efforts to reduce the cost of backpropagation have introduced several innovative approaches. For example, BNN-Edge [215] proposes a low-memory and low-energy training scheme by modifying both forward and backward propagation. This includes binarizing weight gradients, re-engineering batch normalization layers, and using low-precision floating-point data, which collectively reduce memory usage and computational complexity. Furthermore, alternative algorithms to backpropagation have gained traction, particularly those that avoid the chain rule for error propagation. These biologically inspired methods include equilibrium propagation, as adapted by BNN-EQ [108], which treats BNNs as energy-based models. Equilibrium propagation is compatible with neuromorphic CMOS technology and reduces backward-pass overhead. Other methods, such as Direct Feedback Alignment (DFA) [156] and the Forward-Forward algorithm [78], explored by works like [84], have shown promise in allowing parallel error propagation and mitigating precision overflow issues. While these approaches are promising, their application has so far been limited to small-scale datasets such as *MNIST* and *CIFAR10*, with scalability to more complex datasets like *ImageNet* or real-world applications remaining unproven.

Alternative learning strategy: Beyond resource optimization, alternative learning principles are being explored to adapt BNNs for dynamic and evolving environments. Most current approaches rely on offline supervised learning paradigms, which assume that training datasets are static, representative of inference scenarios, and fully labeled. These assumptions limit their applicability to real-world contexts where data is often unstructured, non-stationary, or unlabeled. Addressing this gap, researchers have proposed incorporating self-supervised learning and incremental learning into BNNs. For instance, S2-BNN [194] demonstrates how self-supervised learning can enable representation learning without labeled data, making it suitable for scenarios with limited annotations. Incremental learning strategies, which allow models to acquire new knowledge as additional data or experiences become available, have also emerged as a promising solution. Work such as [83], [107], and [213] investigate the application of incremental learning to BNNs, enabling continuous adaptation to dynamic environments.

In conclusion, BNNs represent a significant step forward for adaptive edge learning in resource-constrained, dynamic environments. However, several challenges remain. Alternative training algorithms must demonstrate scalability beyond toy datasets to larger-scale applications, while adaptive learning principles should be fully integrated into BNN frameworks to address non-stationary and unlabeled data streams. These limitations present opportunities for further research and development, as discussed in the following sections, including a detailed exploration of BNNs and incremental learning (see Section 2.2).

2.2 Incremental learning

DNNs have achieved significant success in various domains, such as image recognition, language processing, and game-playing, where their performance sometimes matches or even surpasses human capabilities. This success is largely attributed to their ability to extract and generalize patterns from large datasets. However, the learning mechanisms of DNNs differ substantially from those observed in humans.

Humans have the remarkable ability to learn sequentially, incorporating new knowledge from tasks, experiences, or environmental changes as they arise. This process allows them to acquire, update, and use knowledge continuously, enabling adaptation to dynamic environments. This capability, referred to as incremental learning (also known as continual or lifelong learning), is a key characteristic of human cognition.

In contrast, DNNs, like most other machine learning models, are typically designed to learn in a static manner. They are trained once on data that is assumed to be representative of all tasks the model may encounter during deployment. If a DNN is later trained only on data from a new task, it suffers from the catastrophic forgetting problem, where knowledge of previously learned tasks is overwritten. This limitation reflects the tension between learning **plasticity**, the model's ability to learn new information, and memory **stability**, its ability to retain prior knowledge. This trade-off is known as the plasticity-stability dilemma and represents a central challenge in incremental learning research.

A naive approach to mitigate catastrophic forgetting is to retrain the DNN on a combined dataset of all previously encountered tasks and the new task. However, this method is computationally expensive and requires significant storage, as the dataset grows with every additional task. Such an approach is impractical, particularly in resource-constrained or real-world settings.

Thus incremental learning research aims to develop methods that allow DNNs to learn continuously while minimizing computational and storage requirements. The objective is to enable DNNs to incorporate new knowledge with minimal reliance on previous training data, thereby mimicking the adaptability and efficiency of human learning processes.

In this section, we provide the background of the incremental learning problem, including its formulation and the evaluation methodologies used in the field. We review the taxonomy of existing approaches, categorizing methods based on their strategies to address the plasticity-stability dilemma. We also focus specifically on replay-based methods, which are a central topic of this thesis, and discuss the interplay between BNN and incremental learning, highlighting their relevance to the study.

2.2.1 Background

Problem formulation

Dynamic environment formulation: An incremental learning neural network, parameterized by θ, must learn successive tasks with limited or no access to previous training samples while maintaining high performance across all encountered tasks. Formally, a batch of training samples for task t is denoted as $\mathcal{D}^{t,b} = \{(x_i^{t,b}, y_i^{t,b})\}_i$, where $\{x_i^{t,b}\}_i$ represents the input data, $\{y_i^{t,b}\}_i$ the corresponding labels, $t \in \mathcal{T} = \{0, \ldots, T-1\}$ the task identity, and $b \in \mathcal{B}^t$ the batch index. A task is defined by its training set $\mathcal{D}^t$, which follows the distribu-

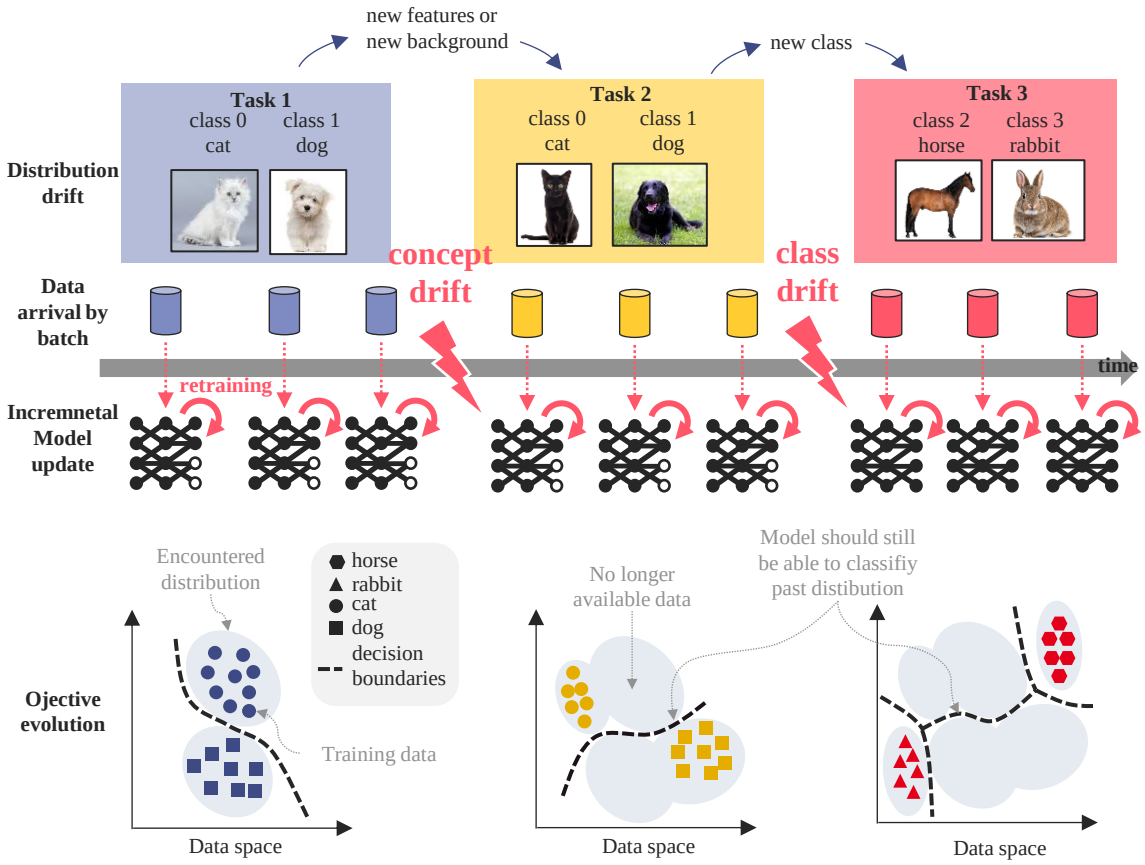


Figure 2.6: The incremental learning problem and objective.

tion $\mathbb{D}^t := p(x^t, y^t)$, where $\mathcal{D}^t$ represents the complete training set, omitting batch indices, with x^t and y^t denoting the full input and label distributions, respectively. Generalization is assessed using a separate test set drawn from $\mathbb{D}^t$. In realistic settings, task identity t and labels y^t may not always be available. Training data can arrive incrementally in batches, *i.e.*, $\{\mathcal{D}^{t,b}\}_{b\in\mathcal{B}^t}$, or all at once per task, *i.e.*, $\{\mathcal{D}^t\}_{t\in\mathcal{T}}$.

The notion of "environment changes" arises at the task level and is characterized by a drift between training distributions $\mathbb{D}^t$. This drift can manifest in two ways:

- **Concept Drift:** The input distribution x^t changes while the label space remains the same. For example, in task t, the objective is to classify white cats and white dogs, whereas in task $t+1$, the goal shifts to classifying black cats from white dogs. Here, the data distribution for the cat class has changed.

- **Class Drift:** The set of available classes changes across tasks. For instance, in task t, the training set consists of cat and dog classes, while in task $t+1$, only horse and rabbit classes are available.

Figure 2.6 illustrates this formulation in a classification problem. The model observes training batches sequentially as they arrive. Concept shifts or new classes may introduce changes. The model must continuously accumulate knowledge and maintain high perfor-

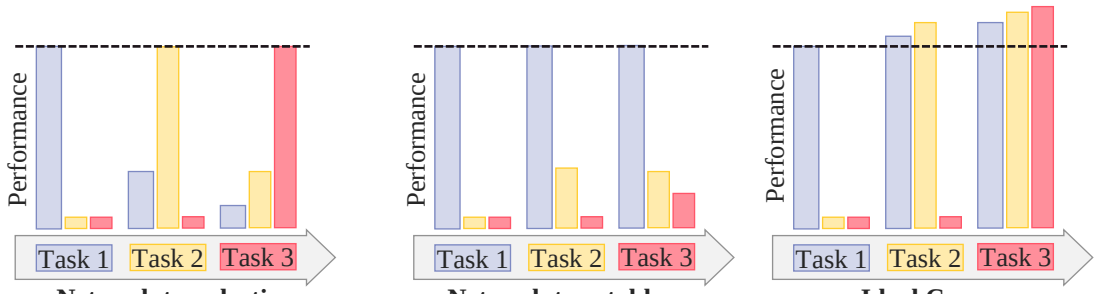
mance on all previously encountered modalities.

Incremental learning objective: The literature formulates the incremental learning problem as a regularization problem: how to regularize training to prevent an increase in loss on past classes while ensuring convergence on the current one.

The first term in the incremental learning (IL) loss $\mathcal{L}_{IL}$ (2.3) represents the standard classification loss on available data from the current task. The second term in (2.3) introduces a regularization mechanism to mitigate catastrophic forgetting. Since past data is no longer accessible, the model approximates this term using $\mathcal{L}_{ret}$, known as the retention loss.

$$\mathcal{L}_{IL} = \underbrace{\mathop{\mathbb{E}}_{(x,y)\sim\mathbb{D}^t}[\mathcal{L}(f(x,\theta),y)]}_{\text{available data distribution}} + \underbrace{\sum_{i=0}^{t-1} \mathop{\mathbb{E}}_{(x,y)\sim\mathbb{D}^i}[\mathcal{L}(f(x,\theta),y)]}_{\text{approximated by } \mathcal{L}_{ret}} \tag{2.3}$$

This formulation helps clarify the stability-plasticity dilemma, illustrated in terms of classification performance in Figure 2.7. If the network trains only on new data, it adapts its weights to the current task but "forgets" knowledge from previous ones. Conversely, if the regularization is too strong, the network struggles to learn new information. Ideally, the network should perform well on each task and, even better, consolidate its knowledge from past experiences.



Network too plastic
The network adjusts its weights too much to the current task, forgetting previous ones.

Network too stable
After the first training, the network fails to accumulate knowledge on new tasks.

Ideal Case
The network perform well on each task and succeed to consolidate its knowledge.

Figure 2.7: The Plasticity-Stability Dilemna. Adapted from [70].

Scenarios

Typical scenarios: From this general problem formulation, several scenarios emerge, each isolating different incremental learning capacities at varying levels of complexity. These scenarios differ based on batch access, task identity availability, and the type of distribution shift encountered. The following list is not exhaustive; for a more comprehensive overview, we refer interested readers to [239]:

- **Task-Incremental Learning (TIL):** Each task has a disjoint label space. Task identities are available during both training and testing.
- **Class-Incremental Learning (CIL):** Each task has a disjoint label space. Task identities are provided only during training.
- **Domain-Incremental Learning (DIL):** All tasks share the same label space (*i.e.*, the same classes) but differ in input distribution (concept drift). Task identities are not required.
- **Task-Free Incremental Learning:** Each task has a disjoint label space. Task identities are not provided, so the model must detect changes in the training data distribution.
- **Online Incremental Learning:** Each task has a disjoint label space. Training samples are processed only once per task.

In this thesis, we focus on Class-Incremental Learning (CIL). It is the most widely studied scenario in the literature, as it directly addresses catastrophic forgetting when introducing new classes and does not rely on the unrealistic assumption that task identities are known. Indeed, knowing task identities simplifies the problem by allowing predictions to be constrained to a specific task's output nodes or by using separate classifier heads. For comparison with state-of-the-art methods, we refer to Task-Incremental Learning (TIL) in Section 3.7, while we explore Online Incremental Learning (OIL) in Section 4.7 as a step toward more realistic settings.

To clarify the specific conditions under which we study CIL in this thesis, we outline the following key assumptions:

- **Unlimited data access within tasks:** Each task allows unrestricted access to its training data. The model can revisit samples multiple times without constraints on epochs or batch size.
- **Access to task boundaries:** Task boundaries are explicitly provided, eliminating the need for novelty detection mechanisms to identify new tasks.
- **Disjoint tasks:** Each task introduces a unique set of classes, such that $y_i^t \in \mathcal{Y}_t$.
- **Pre-training task:** A separate dataset, D^{PT}, is available for network pre-training .

Evaluation methodology

In the literature the performance in incremental learning scenario of a model can evaluated from three aspects: overall performance of the seen classes, retention capacity on past classes (stability) and adaptation capacity on new classes (plasticity).

The **overall performance** is typically evaluated by the global accuracy, denoted $\mathrm{a}_{\mathrm{seen}}$ or the average accuracy, denoted $\mathrm{aa}_{\mathrm{seen}}$. Let $a_{t,j} \in [0, 1]$ denote the classification accuracy evaluated on the test set of the j-th task after incremental learning of the t-th task ($j \leq t$).

The output space to compute $a_{t,j}$ consists of the classes in either $\mathcal{Y}_j$ or $\mathcal{Y}_{i<k}$, corresponding to the use of task identity (TIL) or not (*e.g.*, CIL). The two metrics at the t-th task are then defined as:

$$\mathrm{a}_{\mathrm{seen,t}} = \frac{1}{t+1}\sum_{j=0}^{t} a_{t,j} \tag{2.4}$$

$$\mathrm{aa}_{\mathrm{seen,t}} = \frac{1}{t+1}\sum_{j=0}^{t} a_{seen,t} \tag{2.5}$$

Note that the pre-training classes $\mathcal{Y}_{PT}$ can be taken into account as a task depending on the evaluation. We explicitly mention if it is the case in the proposition Chapters 3 and 4.

Adaptation is measured as the accuracy on the current, *i.e.*, new task D^t, and denoted as $\mathrm{a}_{\mathrm{new}}$. Note that in the literature other metrics as been proposed as the intransigence measure [38] and forward transfer [130]. We noticed that simply monitoring $\mathrm{a}_{\mathrm{new}}$ at each task is sufficient to characterize the adaptation capacity.

Retention is measured as the accuracy on the first task D^0, and denoted as $\mathrm{a}_{\mathrm{old}}$. We compute $\mathrm{a}_{\mathrm{old}}$ on D^0 rather than on all past classes $\mathcal{Y}_{i<t}$ to give an indication on the forgetting of the oldest task. In the same manner, more complex metrics as the forgetting measure [38] and backward transfer [130] have been proposed as global measurement of the forgetting and the capacity of learning a new to better the retention of past one, respectively. In this thesis we look at the forgetting behavior via graphical visualization.

CIL strategy taxonomy

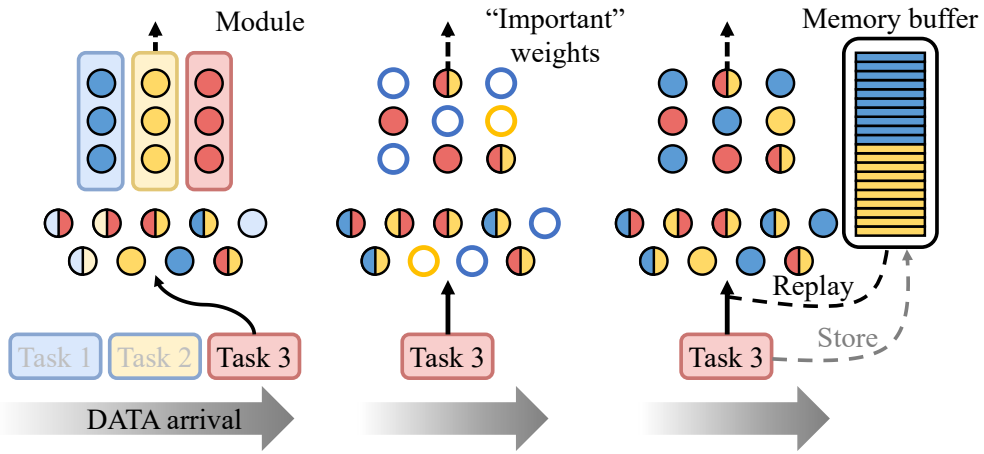


Figure 2.8: Class-Incremental Learning method Taxonomy. Adapted from [70].

Regularization-based methods Regularization-based methods mitigate catastrophic forgetting in incremental learning by constraining model updates to balance stability and plasticity. These methods can be broadly categorized into weight regularization and function regularization.

Weight regularization penalizes changes in critical parameters, ensuring the preservation of essential weights for previous tasks. A widely adopted approach, Elastic Weight Consolidation (EWC) [99], employs the Fisher Information Matrix to estimate parameter importance, preventing drastic modifications to significant weights. Extensions such as Synaptic Intelligence [234] and Memory Aware Synapses [9] refine this process by continuously tracking weight contributions over time.

These methods suffer from limited plasticity, as they overly constrain updates, hindering adaptation to new tasks. Additionally, they lack scalability to complex datasets, where task interference and the accumulation of constraints can degrade performance.

Function regularization preserves the model's predictive behavior by enforcing consistency with past outputs rather than constraining individual weights. This is typically achieved through knowledge distillation, where the previous model serves as a teacher to guide the current model's predictions. Notable approaches include Learning without Forgetting (LwF) [120] and iCaRL [179], which leverage distillation on newly encountered samples to retain prior knowledge without direct access to past data.

Function regularization enables better forward transfer, allowing models to leverage prior knowledge when learning new tasks. It also provides greater flexibility than weight regularization, as it does not strictly restrict parameter updates. However, these methods require maintaining a copy of the previous model, leading to a higher memory footprint, which can become prohibitive for large-scale applications.

Adaptation to BNNs: Regularization-based methods have also been extended to Binary Neural Networks (BNNs). Synaptic Metaplasticity [107] introduces a metaplasticity-inspired mechanism that modulates weight updates, selectively reducing changes to binary weights critical for past tasks. This adaptation enhances *stability* in BNNs, improving resistance to catastrophic forgetting without significantly increasing memory or computational costs. However, similar to other regularization-based approaches, this method struggles with *scalability*, particularly in handling fine-grained or large-scale datasets where limited parameter expressivity may hinder learning.

Architecture-based methods Architecture-based methods in incremental learning mitigate catastrophic forgetting by structuring the model to allocate task-specific parameters, reducing inter-task interference. Unlike regularization-based methods, these approaches explicitly modify the architecture to accommodate new tasks, enabling better knowledge retention. They can be categorized into parameter allocation, model decomposition, and modular networks[218].

Parameter allocation assigns isolated parameter subspaces to each task, ensuring that old task knowledge remains unaffected by new learning. Some methods, such as Piggyback [135], HAT [192], and HH [93], achieve this through binary masks that selectively activate parameters for different tasks. Other approaches, such as PackNet [136] and UCL [2], use iterative pruning and importance-based selection to free up network capacity for future tasks. While parameter allocation ensures minimal forgetting, its major drawback is

scalability, as the number of dedicated parameters grows with the number of tasks.

Model decomposition separates a network into shared and task-specific components instead of fully isolating parameters. This can be implemented using low-rank factorization (*e.g.*, IRU [86]) or adaptive layers (*e.g.*, DyTox [53]). Such methods strike a balance between parameter efficiency and task separation, allowing knowledge reuse across tasks. However, they require careful tuning of decomposition strategies to prevent information loss or redundancy.

Modular networks: These methods construct sub-networks that can be flexibly reconfigured for each task. Progressive Networks [184] introduce a separate sub-network for each new task while allowing knowledge transfer via lateral connections. More advanced modular approaches, such as Expert Gate [6], utilize a mixture of experts to dynamically allocate resources to tasks. Similarly, PathNet [58] and RPSNet [171] pre-define multiple sub-networks and optimize their selection per task. The main advantage of modular networks is their adaptability, but they require careful task identity management to determine the appropriate sub-network configuration.

Replay-based methods Replay-based methods mitigate catastrophic forgetting by storing past samples, "replayed" during training. They can be divided into: Examplars-Rehearsal which stores a subset of past examples in a memory buffer [179], and Pseudo-Rehearsal, which uses generative models to synthesize data from past classes in the input [196] or feature space [198]. While Pseudo-Rehearsal reduces memory requirements, it introduces significant issues for BNNs. In contrast, Examplars-Rehearsal methods are straightforward to implement, scale well while ensuring stable performance across tasks [239], making it an ideal framework for studying FBNNs in a CIL setting.

The usage of Exemplars-Rehearsal for CIL was first proposed on deep NNs in Incremental Classifier and Representation Learning (iCaRL), which uses herding for selecting exemplars, features knowledge distillation and a nearest-mean classifier. EEiL [35] brings further improvements by utilizing a fully-connected classifier and a balanced fine-tuning stage to correct the bias due to training dataset imbalance. After that, investigations have followed on buffer memory management [37], regularization on gradient [130], knowledge distillation (KD) on logits [29], and KD-induced-bias correction [223].

Moreover, there is a renew interest for the Experience Replay (ER) proposed in [175], which consists in interleaving past examples with the current training batch without additional sophisticated techniques or training protocol. Following [30], we adopt ER as baseline methods for our simulations.

2.2.2 Replay-based approaches

Replay-based approaches mitigate catastrophic forgetting by maintaining a bounded memory buffer, denoted B, which stores selected samples from past tasks for retraining. These methods approximate past data distributions by retaining and replaying information, helping ensure prior knowledge is not entirely overwritten when new tasks are introduced.

In incremental learning, memory serves as a structured mechanism for storing, retrieving, and reconstructing information. It can be implemented through explicit storage in a memory buffer or via generative processes that reconstruct past data distributions. The lit-

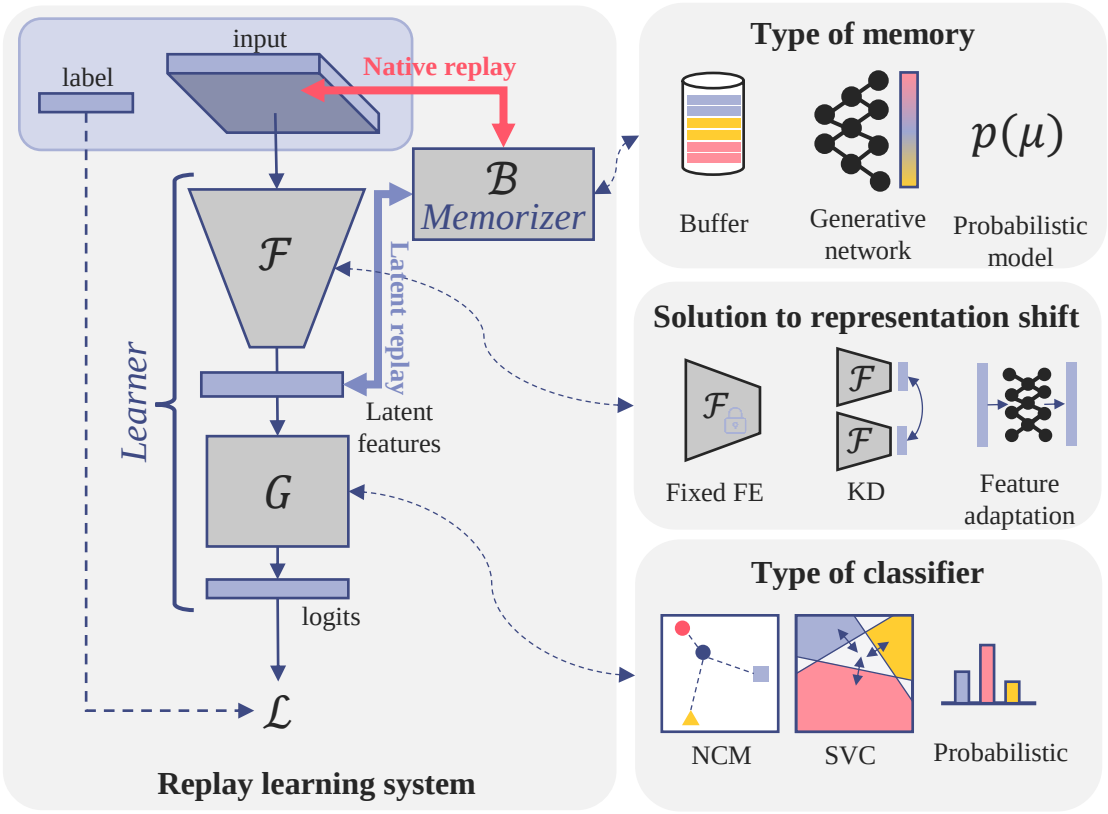


Figure 2.9: Schematic overview of replay-based methods.

erature distinguishes two primary forms of replay-based memory. The first is the *memory buffer*, which explicitly stores a subset of past training data for later use. The second is *generative memory*, where past experiences are reconstructed using statistical models or deep generative networks. In this work, we focus on replay methods utilizing memory buffers and refer to the stored dataset as B.

As proposed in [168], a incremental learning system can be conceptualized as comprising two key components: a *learner* (a deep neural network, DNN) responsible for feature extraction and classification, and a *memorizer* (the memory buffer B) that retains past data for replay. The learner, denoted as $f(\theta, \cdot) = G(F(\cdot))$, consists of two distinct parts. The first component, the *Feature Extractor* (FE), represented by F and parameterized by θ_F, is responsible for learning transferable representations. The second component, the *Classifier*, represented by G and parameterized by θ_G, maps extracted features to class labels. The feature extractor, particularly its early layers, has been shown to transfer effectively across classification tasks when the initial training dataset is sufficiently diverse [19].

This section systematically reviews different replay strategies, focusing on three primary approaches. *Experience Replay* directly stores and replays past samples, ensuring that previously learned distributions remain available for retraining. *Latent Replay* compresses and replays internal feature representations instead of raw inputs, optimizing storage efficiency while preserving useful information. *Pseudo-replay* replaces stored samples with synthetic exemplars generated by statistical models or deep generative networks, offering a model-based alternative to explicit memory storage.

Beyond replay strategies, we analyze key design considerations in memory buffer construction, explore the role of classifier architectures in replay-based methods, and examine the impact of representation drift in latent-space approaches. By investigating these aspects, we aim to provide a comprehensive understanding of the strengths and limitations of different replay mechanisms in incremental learning.

Experience replay approach

Experience replay is a widely used incremental learning approach that maintains a memory buffer storing a subset of past samples to approximate previous data distributions. It includes several variants, such as native replay, which stores data in the input space (*e.g.*, raw images), and latent replay, which stores intermediate representations (*e.g.*, feature embeddings). Given the limited storage capacity and the ever-growing past distribution as new tasks arrive, two key challenges arise: *how to update the memory buffer* and *how to effectively exploit the stored knowledge*. This section details the specificities of both native and latent replay, while the next section focuses more specifically on the characteristics and implications of latent replay.

Memory buffer update In real-world scenarios, memory is strictly limited, requiring careful selection of stored samples to best represent the past distribution. Several strategies exist, ranging from simple random selection to more sophisticated approaches based on sample difficulty, gradient information, or robustness to perturbations. In the case of extremely constrained memory (e.g., fewer than 100 samples), prior work [37] has explored various sampling strategies. One of the most common methods is *Reservoir Sampling*, which maintains a fixed number of old samples per task using uniform random selection. Another widely used strategy is the *Ring Buffer*, which ensures an equal number of samples per class, providing a more balanced representation. A more sophisticated alternative is *Herding* [179], which selects samples closest to the class feature mean, ensuring that the most representative examples are retained.

In this thesis, we adopt the *Ring Buffer* sampling strategy for several reasons. First, it is computationally efficient, as selection is based on per-class random sampling, avoiding the computational overhead of more complex strategies that rely on loss values, gradients, or feature means while maintaining competitive performance. Second, it ensures a balanced representation of all classes, unlike Reservoir Sampling, which may lead to the underrepresentation of certain classes in small memory budgets. This characteristic is particularly beneficial for mitigating catastrophic forgetting in low-memory regimes, ensuring that every class is represented by at least one sample. Finally, the Ring Buffer optimally utilizes the available memory by fully populating the buffer at each task, unlike growing buffer methods that pre-allocate a fixed number of samples per class. The procedure is summarized in Algorithm 1.

Knowledge representation in memory Traditionally, memory buffers store pairs of input samples and their corresponding labels. However, alternative representations exist, such as storing logits instead of discrete labels. Logits correspond to the raw, unnormalized outputs of the classification model before applying softmax. These outputs encode relative class probabilities and have been suggested to carry richer information than labels [147].

Algorithm 1 Ring buffer management for replay memory

Require: Replay buffer B, incoming data (X_i, Y_i), maximum buffer capacity m_b
Ensure: Updated buffer B

Step 1: Compute class statistics
$c_B \leftarrow$ number of unique classes in B
$c_{\text{new}} \leftarrow$ number of unique classes in Y_i
$c_{\text{total}} \leftarrow$ number of unique classes in $B \cup Y_i$
$m_{\text{per\_class}} \leftarrow \frac{m_b}{c_{\text{total}}}$
Step 2: Compute number of elements to remove per class
for each class cls in B **do**
    $r(cls) \leftarrow \max(\text{length}(B[cls]) - m_{\text{per\_class}}, 0)$
Step 3: Remove excess samples randomly per class
for each class cls in B **do**
    if $r(cls) > 0$ **then**
        Randomly select length$(B[cls]) - r(cls)$ samples
        $B[cls] \leftarrow$ selected samples
Step 4: Compute available space in buffer
buffer_space $\leftarrow m_b - \sum \text{length}(B[cls])$
Step 5: Compute how many new samples to add per class
if $c_{\text{new}} > 0$ **then**
    new_samples_per_class $\leftarrow \min(m_{\text{per\_class}}, \frac{\text{buffer\_space}}{c_{\text{new}}})$
else
    new_samples_per_class $\leftarrow 0$
Step 6: Add new samples from (X_i, Y_i)
for each unique class cls in Y_i **do**
    indices $\leftarrow$ Indices of Y_i where label $= cls$
    sampled_indices $\leftarrow$ Randomly select $\min$(length(indices), new_samples_per_class) indices
    if $cls \notin B$ **then**
        $B[cls] \leftarrow$ empty list
    Append X_i[sampled_indices] to $B[cls]$
return B

However, their use in incremental learning often requires bias correction mechanisms [223] and does not consistently improve performance [18]. Additionally, storing logits increases memory requirements significantly, as each sample would require storage proportional to the number of classes, whereas labels require only a single value per sample. Given these trade-offs, we opt to store labels only for all our experiments to optimize memory usage without introducing unnecessary complexity.

Memory buffer compression To further optimize storage efficiency, various methods have been proposed for compressing stored data. For instance, AQM [31] employs a VQ-VAE [207] framework to reduce the dimensionality of stored inputs, while MRDC [219] leverages determinantal point processes for memory-efficient selection. In this thesis, we consider data preprocessing strategies that align with both input compression and the constraints of first-layer binary computation, ensuring that the stored representations remain compatible with our model architecture.

Memory buffer exploitation Stored samples can be leveraged for training in two main ways: either through direct regularization, where past samples are replayed and interleaved with new training data, or through indirect regularization, where optimization is adjusted using strategies such as orthogonal gradient projection [130]. For simplicity and computational efficiency, we adopt the direct replay approach, where past samples are interleaved with new task data. At each training session, the memory buffer B is concatenated with the current task dataset D^t to form the training set. Mini-batches are then sampled randomly from $B \cup D^t$, effectively enforcing an implicit regularization term, as shown in Eq. (2.6).

$$\mathcal{L}_{IL} = \mathop{\mathbb{E}}_{(x,y)\sim D^t}[\mathcal{L}(f(x,\theta),y)] + \underbrace{\mathop{\mathbb{E}}_{(x,y)\sim B}[\mathcal{L}(f(x,\theta),y)]}_{\mathcal{L}_{ret}} \tag{2.6}$$

A key challenge in this setup is *class imbalance*, since the current task D^t contains significantly more recent samples than past samples stored in B, leading to biased learning. Various heuristics and multi-phase training strategies[35] have been explored to mitigate this issue, but these approaches remain underdeveloped in the literature, especially in the case of label storage rather than logits. We further analyze class imbalance and loss balancing in Section 3.4.

Latent replay approach

Latent replay, also called features replay, uses the first layers of the network to compress input data into latent features and stores those features. First introduced in latent replay works [161] for mobile applications, it significantly reduces memory footprint and computation during training. They propose storing latent samples instead of raw native input samples, referred to as native replay, and use a regularization term to further prevent catastrophic forgetting on the last layer. The authors also analyzed how the depth of the latent space, within a MobileNetV1 architecture, influences incremental learning performance. Interestingly, this method was extended with a TinyML platform for both inference and retraining, proving the concepts on more constrained hardware [177].

Building on this idea, Vorrabi *et al.* (2023) adapted latent replay to binary neural networks (BNNs) by using a fixed BNN feature extractor and a quantized classifier, combined with CWR* regularization to limit forgetting [212]. They later extended the model by introducing additional learnable binarized layers, which improved incremental performance [213]. However, the model still relies on real-valued first and last layers, as well as batch normalization.

In parallel, our work proposes to go a step further, addressing CIL in FBNNs, scaled up to the state-of-the-art *CORE50* dataset[128]. In this thesis, we discuss the FBNN design and its training procedure, introduce new metrics to analyze the CIL performance, explore loss balancing, detail a method to pre-train the feature extractor for transferable representations, and, finally, thoroughly compare latent and native replay.

Representation drift A major issue arises with replaying from a latent space, the *representations shift* or *aging problem*. Retraining the FE on new task changes the representation and the approximation of past distribution in the memory buffer does not correspond anymore to the new projection of past distribution. To address this issue, we found three strategies in the literature. (1) Feature distillation between a saved old model prevents the representation to drift too far from the former one. (2) feature adaptation is proposed in MemoryEfficient [89]. They learn the drift in representation with an additional MLP and transformed the exemplars stored in the memory buffer. (3) Keeping the feature extractor fixed, learned from the initial task or on large dataset.

In this thesis we choose the last option as a first approach, while considering the other one as future works. With fixed representation, arises the question of how to create and exploit the strength of representation for incremental learning. The link between representation and catastrophic forgetting has been exhibited since early works in the 90's[61] where sparse and semi-distributed representation has been proven to reduce forgetting. This idea has been extended to contemporary works using Meta-learning to learn sparse representation [91] or specialized regularizer [7]. However the notion of sparsity is not possible with binary arithmetic. Another line of work, studied in this thesis, is learning representation with self-supervised learning. Literature around self-supervised learning and its link to incremnental learning is discuss in Section 3.5.

Pseudo-replay approach

Alternatively to experience replay and latent replay that approximate past distribution via sampling and storing in a memory buffer, Pseudo-replay, also called generative replay, train a generative model to capture and replay past distribution. Generative model produces synthetic data point from random noise denoted as Pseudo-examplars. In the literature two types of models has been explored to continuously encode past distribution and generate pseudo-exemplars: deep generative models as generative adversarial networks (GANs) and (variational) auto encoders (VAE), and statistical models as Gaussian Mixture Models (GMM). Continual learning of deep generative model on hight dimensional data (*e.g.*, images) is difficult and requires significant resource overhead. Thus research has switched from input-level data pseudo-replay to feature-level data generative replay to target larger and more complex dataset. Work using fixed feature extractor with generative network as DreamNet[198] or FearNet[96], are promising with conventional floating point arithmetic,

but with the lack of studies around binary arithmetic and possible convergence issue we focus the literature review on generative statistical models.

The main approach, known as the **Prototype-based method**, stores the mean feature representation for each class distribution, also referred to as the *centroid*. It then performs data augmentation around this centroid using class statistics [242, 241, 243, 164]. We provide a detailed discussion of these methods in Chapter 4.

These approaches are described as exemplar-free because they do not store past samples. However, they still require memory to retain first-order statistics (mean) and sometimes second-order statistics (covariance matrix). Figure 2.10 compares the memory requirements of prototype-based methods that assume a Gaussian distribution (*e.g.*, [241] [65] [185]) with those of latent replay. The figure suggests that the claimed advantages of these methods over latent replay remain uncertain—an aspect that has been largely overlooked in the literature.

Similar to latent replay, some methods rely on a fixed feature extractor, while others use knowledge distillation to preserve the learned representation across updates.

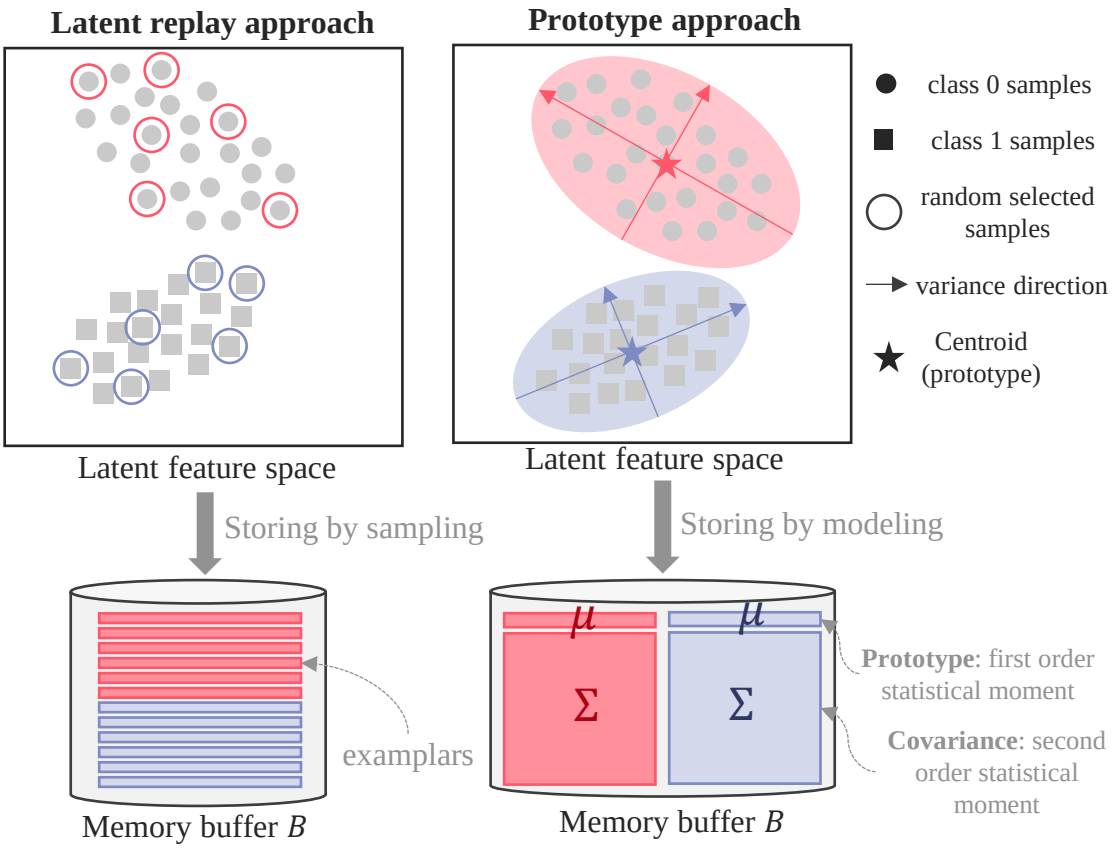


Figure 2.10: Comparison of disbribution approximation between latent replay and prototype-based approach.

Feature extractor choice

The performance of prototype-based and, more broadly, latent replay methods depends on the chosen network architecture. The standard choice in the literature is ResNet and its variants (ResNet18, ResNet32, ResNet50), ensuring a fair comparison of incremental learning methods. As research improves strategies to mitigate catastrophic forgetting, two main directions have emerged: scaling up architectures with larger datasets [239] and optimizing lightweight models for constrained platforms. Some studies question whether catastrophic forgetting remains an issue in large, pre-trained networks [172], while recent findings suggest that scaling up reduces its impact [115].

This thesis follows the second approach, extending incremental learning research to smaller architectures. The motivation stems from both smart imaging applications and the need to study catastrophic forgetting in settings where it remains relevant. Latent Replay [161] adopted MobileNet[81] for mobile CPU inference, while [206] explored replay techniques with PhiNet [158], a TinyML-optimized network requiring only 2MB of Flash and 1MB of RAM on an STM32H743 microcontroller. Similarly, [83] analyzed quantization effects on a WRPNet[146] with ICarL strategy, highlighting the challenges of training BNNs incrementally. Vorabbi et al. [213] examined latent replay on Hybrid-BNNs (QuickNet, BiRealNet, ReacNet) for on-device incremental learning but relied on a large feature extractor ($\simeq 30Mb$), unsuitable for frugal inference.

In contrast, our work on latent replay with Fully-BNNs [17] addresses catastrophic forgetting using a compact $4Mb$ feature extractor with binary-only arithmetic, serving as a realistic baseline for future ASIC implementations. More broadly, this thesis tackles the research gap in compact feature extractors for incremental learning at the extreme edge, demonstrating the feasibility of Fully-BNNs for efficient, low-resource inference.

Classifier choice

Two types of classifiers are considered in prototype-based methods: nearest class mean (NCM) and linear projection. In NCM [143], a new example is classified by assigning it the label of the class whose centroid is closest to the example's feature vector, according to a chosen metric, which may also be learned from data. This approach efficiently incorporates new classes by simply computing their centroids without requiring retraining.

Linear projection consists of a fully connected (dense) layer that maps input features to class scores through a learned linear transformation. Formally, given an input feature vector z, the output is computed as $s = \theta z + b$, where θ is the weight matrix, b is the bias vector, and s represents the class scores before applying any activation function. This output can be interpreted in two ways, as follows.

In a probabilistic framework, applying a softmax function to s produces class probabilities, and training with cross-entropy loss encourages the network to assign high confidence to the correct class. Training with backpropagation in this setting enables end-to-end optimization of both the feature extractor and the classifier, allowing the network to learn feature representations that are directly optimized for classification.

In a one-vs-all interpretation, each output node can be seen as a separate decision function, where $s_i = \theta_i^T z + b_i$ represents a class-specific score. Classification is performed by selecting the class with the highest score. The weights in this framework can be learned using a linear support vector classifier (LinearSVC), which optimizes a hinge loss, or by

training a neural network via backpropagation with hinge loss (*i.e.*, a multi-class SVM-like approach).

Notably, for FeTril methods, experiments have shown that using LinearSVC instead of a dense layer leads to better classification performance, likely due to its ability to learn more robust decision boundaries.

In more recent work (2024), non-linear classifiers have been proposed for prototype-based methods. For instance, FeCAM [65] introduces an optimal Bayes classifier based on the Mahalanobis distance, achieving state-of-the-art classification performance on *CIFAR100* and *TinyImageNet*. However, for fair comparison, we exclude these methods to ensure that all classifiers remain equivalent, allowing a controlled evaluation of prototype-based methods under the same assumptions.

2.3 Summary and positioning

Replay for studying incremental learning in BNNs

Incremental learning presents several challenges, with catastrophic forgetting being the most critical. Among the various approaches addressing this issue, replay emerges as suitable for studying catastrophic forgetting in highly quantized network architectures. Replay is straightforward to implement, effective, and compatible with emerging architectures. However, it comes at the cost of additional memory requirements for storing replay buffers.

Extending incremental learning research to lightweight models

The literature on replay methods highlights efforts to adapt replay strategies for lightweight network architectures, aiming for deployment on resource-constrained platforms such as TinyML and microcontrollers (MCUs). This thesis follows this research direction by addressing the gap in replay systems for neural network architectures designed for ultra-low-power platforms (consuming only a few mW per inference). We focus on compact models with minimal parameters and fully binarized arithmetic. Although there is a growing interest in binary-only arithmetic compute graphs, no existing work has explored this, even in the context of offline training. Furthermore, no baseline architecture exists for studying replay while meeting the constraints of ultra-low-power platforms. This thesis aims to design a fully binarized neural network (FBNN) baseline specifically tailored for incremental learning with replay.

Investigating best practices for experience replay in FBNNs

The removal of full-precision layers has been shown to hinder convergence in offline training. Developing replay techniques suited for BNNs requires a thorough understanding of how optimization choices impact incremental learning performance. To establish a solid foundation, we conduct a comprehensive study of experience replay techniques—both native and latent—focusing on key system components: pre-training loss, incremental learning loss, and buffer size influence. This analysis serves as a basis for developing improved replay strategies.

Proposing a prototype-based pseudo-replay method

The literature highlights prototype-based pseudo-replay as a promising approach for resource-constrained platforms. This method extends latent replay while reducing memory footprint, providing a step toward generative replay without the challenges associated with deep generative models. However, current prototype-based methods are not directly applicable to binary latent spaces. Their reliance on statistical models using continuous variables conflicts with the categorical nature of features extracted by FBNNs. This gap presents an opportunity to develop a prototype-based method tailored to binary latent spaces, involving statistical models suited for categorical data. Furthermore, with quantization-aware training (QAT) and the straight-through estimator (STE) (see Section 2.1.1), any latent space can be binarized. This opens the possibility of extending pseudo-replay techniques beyond BNNs to conventional neural networks.

Memory footprint analysis

Memory constraints are a critical factor in resource-limited platforms. Several studies evaluate memory footprint in megabits (Mb), using it as a comparative metric [89]. However, binary feature precision differs from input image precision or prototype precision, making direct comparisons between methods challenging. A fair evaluation requires an iso-memory comparison approach. In this thesis, we analyze memory footprint to compare native and latent replay in Chapter 3 and latent replay against pseudo-replay in Chapter 4.

Class-incremental learning as an evaluation framework

For evaluation, we adopt the Class-Incremental Learning (CIL) scenario to analyze catastrophic forgetting. Given the constraints imposed by our network architectures, optimization remains challenging. CIL allows for a rigorous assessment of sequential training and forgetting without introducing conflicting optimization objectives. To align with the field's shift toward more realistic settings, we present a preliminary study on online incremental learning in Chapter 4.

Highlights of the chapter

- Research on Binary Neural Network has primarily focused on improving accuracy, often at the cost of introducing mixed-precision operations. Recently, there has been renewed interest in fully binary compute graphs to enable more energy-efficient inference models.
- Replay methods are a key area of research in incremental learning, providing a practical framework to study sequential training, catastrophic forgetting, and synthetic data generation in BNNs.
- This thesis aims to extend current work on Fully Binary Neural Networks and prototype-based methods. It explores the intersection of BNNs and incremental learning, specifically investigating Experience Replay for FBNNs and designing a prototype-based approach tailored for BNNs.

Chapter 3

Toward Experience Replay on Fully-Binary Neural Network

Summary

This chapter investigates the application of Fully-Binary Neural Networks (FBNNs) to Class Incremental Learning (CIL) using experience replay (ER) methods, addressing their potential and challenges for incremental learning (IL) in resource-constrained environments. Building on the foundation laid in Chapter 2, it explores how FBNNs, despite their extreme memory and compute efficiency, can be optimized to perform well in IL scenarios while preserving their hardware-friendly properties.
A reference FBNN architecture is introduced, fully binarized to eliminate floating-point dependencies while incorporating efficient design elements, such as layer-wise scaling normalization, trainable global average pooling, and thermometer-encoded inputs. These innovations ensure compactness and robustness, enabling the proposed 3Mb-BNN model to achieve comparable performance to state-of-the-art.
The chapter focuses on balancing adaptability and retention in FBNNs during incremental learning. It investigates the impact of loss functions, identifying Categorical Cross-Entropy as the most effective for achieving this balance. Semi-supervised pre-training is proposed to enhance feature transferability in latent replay, combining supervised learning with unsupervised methods like Barlow Twins and custom regularization. This approach boosts CL accuracy while mitigating the limitations of fixed feature extractors. Moreover, this semi-supervised approach open the way to a more relevant framework for a real-life solution deployment.
Experience replay is analyzed in detail, comparing native replay (storing raw inputs) and latent replay (storing binary feature representations) under iso-memory constraints. Latent replay proves more memory-efficient for small buffers, while native replay excels with larger buffers, offering greater adaptability especially in the context of a relatively small feature extractor, with limited expressiveness. A comparative analysis against floating-point networks highlights the superior memory efficiency of FBNNs, with competitive performance achieved at significantly reduced memory footprints.
Finally, the methods are validated on the *CORE50* dataset, demonstrating their applicability to real-world scenarios. The *3Mb Res-BNN*, an improved variant model with skip-connexion, following the principles established in this chapter, achieves comparable or superior performance to larger floating-point models while relying solely on binary computations and drastically reduced memory usage. These findings underscore the potential of FBNNs for incremental learning in edge-device settings.

3.1 Introduction

Fully Binarized Neural Networks (FBNNs) are promising candidates for ultra-low-power and memory-efficient inference on edge devices. As discussed in Chapter 1, their minimal footprint makes them highly attractive for embedded applications. However, their integration into incremental learning (IL) pipelines remains largely unexplored. This chapter addresses specific challenges arising from the binary nature of FBNNs. While these models provide substantial hardware inference efficiency, their training is more complex due to limited representational capacity and the non-differentiability of their operations. Chapter 2 identified two major research gaps: (1) the lack of evaluation of experience replay (ER) strategies on fully binarized models, particularly in terms of memory efficiency and performance trade-offs; (2) the absence of well-established design and training practices to support incremental learning in FBNNs, including the role of pretraining and architectural choices. To address these gaps, this chapter proposes a structured methodology to identify best practices for retraining compact FBNNs in an incremental learning setting. Building on the findings from Chapter 2, we investigate the following key questions:

- Can FBNNs be incrementally retrained using experience replay strategies?
- How can we design and pre-train FBNNs to facilitate adaptation without catastrophic forgetting?
- What trade-offs exist between native and latent replay under fixed memory budgets?

Contributions and objectives

Building on the analyses and opportunities derived from Chapter 2, this chapter presents the first approach to incremental learning in Fully Binarized Neural Networks (FBNNs) and addresses the above questions through four key contributions, each exemplified on the *CIFAR100* dataset. Finally, the overall approach is validated on the *CORE50* dataset [128].

FBNN design for incremental learning: Firstly, this chapter discusses several salient points to guide FBNN design for CIL. Most BNNs from the literature are not fully binary, with their first and last layers in richer, real-valued or integer representations [16]. Furthermore, to ease convergence, existing BNNs may include Batch Normalization (BN) layers, typically in floating point. To avoid channel-wise BN, we propose training the network in a single stage, relying on layer-wise scaling factors only as normalization. The values of these scaling factors are set not from BN layer parameters, but using the dimensions of the network topology itself. Furthermore, for the network's bottleneck, we propose a trainable global average pooling to reduce latent space dimensionality, making the network robust to spatial translations of the input, despite the low expressiveness of binary values. Last, to avoid real-valued input data, we transform image pixels into a luminance/chrominance domain, which is then converted into a thermometer encoding [15].

Our FBNN is compared with the two closest NN designs, without BN [39, 132] or claiming only binary arithmetic [132]. FBNNs of three model sizes (parameters) are set: a 3 Mega bit (Mb), a 9Mb, and a 24Mb. Experiments indicate that our network reaches state-of-the-art performance in an offline setting, *i.e.*, when all the dataset is available. This paper contributions mainly focus on our tiny *3Mb-BNN* model. We implement two CIL types of ER,

Native replay (storing raw inputs in the buffer) and Latent replay (storing latent features).

Loss balancing for adaptation and retention: Secondly, we address the main CIL trade-off, namely adaptability vs retention, *i.e.*, performance on past tasks vs. on the current task. Loss balancing, a possible strategy to control adaptability and retention, is barely addressed in the CIL literature. Along with loss balancing, here we explore three loss functions suitable to BNNs, namely, the conventional Categorical Cross-Entropy (CCE), the squared-hinge, potentially better for BNNs [153], and a focal loss . We find that a common CCE loss suits well CIL in FBNN, for both Latent and Native replay. Balancing the loss significantly enhances the final performance of all tasks, while slightly decreasing adaptation.

Feature extractor pretraining for latent replay: The proposed FBNN is split into a Feature Extractor (FE) and a classifier, as in any conventional NN for image classification problems. The common practice for learning feature representations is to perform supervised learning on a pre-training dataset [213]. Extracted features can be discriminative for pre-training classes; however, there is no guarantee they are also discriminative for later tasks. This becomes an issue for latent replay, where the feature extractor is not updated.

Thirdly, the supervised pre-training approach is questionned, proposing a semi-supervised alternative to learn more transferable features in FBNNs. Supervised training is combined with Barlow Twin (BT) [233] and custom activation regularization that forces features to be more distributed. It increases the final test accuracy by 1.17pts in CIL for the *CIFAR100*. One would expect a significantly greater performance improvement on more extensive benchmarks, particularly when applied to real-world use cases.

Comparative analysis of replay methods and model types: Fourthly, the FBNN exploration covers both Native replay and Latent replay. The performance of Native replay and Latent replay are compared for a given memory buffer footprint. The performance against full-precision network is also compared to our FBNN. For a small memory buffer ($<$25Mb), Latent replay yields higher final accuracy than Native replay. In this buffer size range, Latent replay offers a better retention but lower adaptation than Native. For larger buffer sizes, Native is better.

Validation on more complex benchmarks: Finally, all the proposed methods are put together and experimented on a *CORE50* dataset [128]. *CORE50* serves to benchmark IL strategies in real-life scenarios. To fit this dataset, we design a deeper architecture, *Res-BNN*, following the principles above, but still having only a 3Mb memory footprint. Experiments on *CORE50* show that our *3Mb-Res-BNN* with Latent replay achieves performance comparable to ER methods using a 352Mb ResNet18 [166]. More impressively, Native replay surpasses other replay methods, with a model memory footprint that is 119$\times$ smaller and a buffer size 1.8$\times$ smaller, while relying solely on binary scalar products.

The outline of this chapter is as follows. Section 3.2 defines the evaluation metrics and the overall methodology. Section 3.3 introduces the FBNN design guidelines, followed by Section 3.4, which discusses loss balancing. Section 3.5 proposes a semi-supervised pre-training of the FE for CIL in FBNN. Section 3.6 compares Native and Latent replay at iso-memory. While Section 3.7 finally presents the evaluation of the methods on the *CORE50*

dataset.

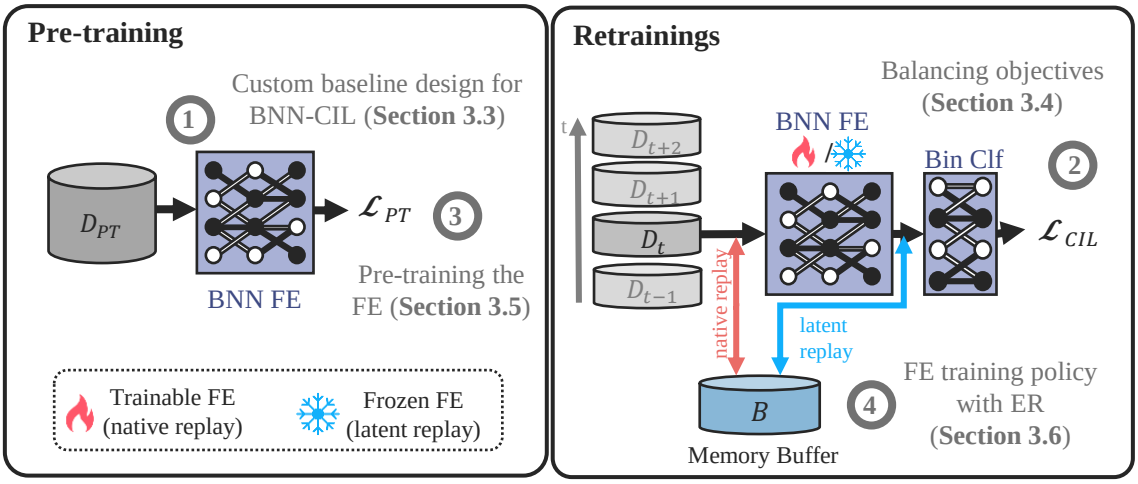


Figure 3.1: Typical life-cycle of a FBNN in CIL: a pre-training followed by retraining. We highlight the challenges tackled in this chapter and the section where there are investigated.

3.2 Evaluation methodology

In this section, we detail the research methodology used throught all experiments in this chapter. In Section 3.2.1, we detail the datasets employed and the way they are splited for CIL evaluation. We secondly introduce the conventional metrics used for CIL evaluation and propose two new metrics specific for our learning system, *i.e.*, a compact FBNN with a memory buffer in Section 3.2.2. Finally in Section 3.2.3, we detail the optimization protocol set for each learning phase.

3.2.1 Datasets and CIL set-up

All experiments on FBNN in CIL in this chapter are exemplified on *CIFAR100* [103] dataset, which is described as follow:

- **CIFAR100**: *CIFAR100* consists of labeled images from 100 classes, each image having 32×32 pixels and 3 color channels, with each channel encoded as 8-bit integers. Each class contains 500 training images and 100 test images. For validation, we reserve 10% of the training images. The large number of classes and the relatively small number of images per class makes it a challenging benchmark, especially for FBNNs. Prior work on BNNs in CIL [107] has demonstrated their approach on simpler datasets like *MNIST* or *CIFAR10*, raising questions about the scalability of these methods.

- **CIL set-up**: In all experiments except those in Section 3.6.2, the first $N_{PT} = 50$ classes are used for pre-training, $\mathcal{Y}_{PT} = (0, \ldots, 49)$, while the remaining 50 classes

are split into $T = 5$ tasks, each with 10 classes. We refer to this configuration as *CIFAR50+5X10*.

- **CIL set-up in Section 3.6.2**: In Section 3.6.2, we further evaluate the influence of the number of pre-training classes (N_{PT}) and the number of tasks (T) on incremental performance. The configurations are labeled pt10 and pt50, corresponding to $N_{PT} = 10$ and $N_{PT} = 50$, respectively. Retraining is also performed on the last 50 classes, split into T tasks with $\frac{50}{T}$ classes each. This configuration allows for comparing the influence of different pre-training datasets, $\mathcal{D}_{PT}$, on the same scenario.

Section 3.7 finally reports an in-depth evaluation of the proposed approach on *CORE50* [128] with a pre-training on *STL10* [44] for a dedicated FBNN (*3Mb-Res-BNN*). Both *CORE50* and *STL10* datasets are described in Section 3.7.

3.2.2 Metrics

In the CIL community, the goal is a high final accuracy, after a given number of learned tasks. Previous work investigates the final accuracy along two directions, adaptation and retention, as defined below. We argue that other metrics allow to better understand the details of a CIL system, as presented in the core of this section. Finally, our experimental setup with respect to CIL and FPNN is extensively detailed.

Conventional metrics. The three important properties of CIL are the ability to adapt to new classes (adaptation), to remember past classes (retention), and to generalize well on all seen classes (global accuracy). These can be evaluated by the accuracy of the corresponding subset of classes, on the test set. **Adaptation** is measured as the accuracy on the current, *i.e.*, new task $\mathcal{D}^t$, and denoted as $\mathrm{a}_{\mathrm{new}}$. **Retention** is measured as the accuracy on the first task $\mathcal{D}^0$, and denoted as $\mathrm{a}_{\mathrm{old}}$. We compute $\mathrm{a}_{\mathrm{old}}$ on $\mathcal{D}^0$ rather than on all past classes $\mathcal{Y}_{i<t}$ to give an indication on the forgetting of the oldest task. **Global accuracy** $\mathrm{a}_{\mathrm{seen}}$ is the accuracy on all seen classes $\mathcal{Y}_{i\leq t}$. For the sake of clarity, $\mathrm{a}_{\mathrm{final}}$ refers to the global accuracy evaluated once all tasks have been learned.

Beyond conventional metrics. Our learning system comprises a compact FBNN and a bounded memory buffer, both introducing challenges in training. Table 3.1 summarizes relevant metric notations. Training accuracy is often omitted in the literature, though it offers significant insights into model expressiveness and provides useful information on overfitting issues. We report accuracy figures for the train and test cases when necessary. As the FBNNs studied are compact, a plateau in training performance may indicate that test performance is capped by network size, whereas high training performance may suggest that further regularization could improve test performance.

Furthermore, the buffer size should be low (*e.g.*, $\leq$ 100Mb), implying a high risk of overfitting. To quantify this, we introduce two metrics: $\mathrm{a}^{\mathrm{train}}_{\mathrm{buffer}}$, the accuracy computed on the buffer samples in the train set, M, and $\mathrm{a}^{\mathrm{test}}_{\mathrm{buffer}}$, the accuracy computed on the test set of past classes $\mathcal{Y}_{i<t}$, at task t.

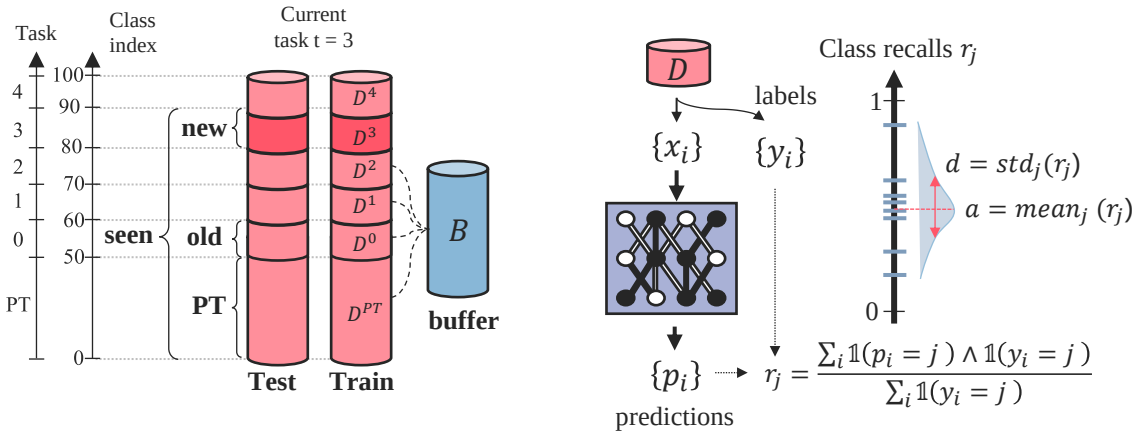
Finally, the network may be initially pre-trained on a dataset $\mathcal{D}^{PT}$ and then incrementally retrained. To have an indication of the performance at the start of the incremental

subscript	PT	pre-training task $\mathcal{D}^{PT}$
	old	first task $\mathcal{D}^0$
	new	current task $\mathcal{D}^t$
	seen	all encountered tasks
	final	all tasks
	buffer	samples in buffer M
superscript	train	samples seen during training
	test	samples in the test set

Table 3.1: Notations for the accuracy, a, and dispersion, d, metrics. The evaluation is done on test set when the superscript is omitted.

retraining we also introduce the accuracy at the pre-training task a_{PT}.

Dispersion metric. The CIL problem can be framed as a detection problem, instead of a classification problem, where the model's goal is to detect new classes, as well as past classes. The accuracy on a subset of classes only gives the average of classes recall. To provide more insight about each class detection, we propose a dispersion metric, denoted as d, defined as the standard deviation on class recalls. If the model must detect every class equally well, the ideal case is the one where the dispersion is minimal. Combined with the accuracy, dispersion also gives insights on the lower-bound detection performance. Lower accuracy with smaller dispersion can be preferable to higher accuracy with bigger dispersion, if the minimum detection performance is higher. As with accuracy, dispersion can be estimated on a subset of the data, as visible in Table 3.1.



a) Subset notation used during evaluation in CIFAR50+5X10

b) Accuracy and dispersion computation

Figure 3.2: Metrics and evaluation methodology.

3.2.3 Training recipe

The training protocol is consistent across all tasks, including pre-training. We use the Adam optimizer with momentum parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, a batch size of 64, and an initial learning rate of 10^{-4}. We chose a small batch size and low initial learning rate after observing that higher values can lead to training instability. The type of loss function is discussed in Section 3.4, with Categorical Cross-Entropy (CCE) as the baseline. Additional considerations include a selective softmax activation, an adaptive learning rate scheduler, and a regularization protocol tailored for FBNNs in CIL.

- **Selective softmax**: During training, logits are normalized using a softmax activation, but only for the classes encountered so far. The output layer is designed to accommodate a predetermined maximum number of classes(*i.e.*, 100 classes for *CIFAR100*). To prevent the weights associated with future classes (*i.e.*, not yet encountered) we restrict the softmax normalization to seen classes output. To ensure that the model does not incorrectly optimize for future classes, softmax normalization is restricted to the classes already encountered. This approach ensures that (1) unused outputs do not affect the loss computation and (2) optimization does not cause the weights for future classes to drift, which helps avoid undesirable initialization points for those future classes.

- **Learning rate scheduler**: We employ an adaptive "Reduce On Plateau" learning rate scheduler [43] combined with early stopping [169] during each retraining phase. If no improvement of at least 10^{-3} is observed on the training set over 10 epochs, the learning rate is reduced by a factor of 0.7. Early stopping is triggered if no improvements are observed over 25 consecutive epochs. To prevent premature termination during initial training phases, early stopping is disabled for the first 50 epochs. This adjustment helps avoid stopping optimization due to convergence issues with the initial learning rate (*e.g.*, exploding gradient encontered during preliminary studies), while allowing the learning rate scheduler to update if needed . This protocol ensures efficient optimization, proper convergence, and avoids unnecessary training steps, critical factors for incremental learning.

- **Regularization**: Regularization is required to avoid over-fitting, especially on the buffer B [30]. Data augmentation is applied during retraining, including random horizontal flips, translation (up to 6.25% in all directions, *i.e.*, $+/-2$ pixels for *CIFAR100*), and random contrast, which effectively enhances validation performance during offline training. In native replay, data augmentation is applied on both $\mathcal{D}^t$ and $\mathcal{B}$, where as in latent replay, data augmentation is applied on both $\mathcal{D}^t$. This distinction is further described in Section 3.6.

- **Notes on regularization**: Preliminary tests with other data augmentation techniques, such as cut-out, random zoom, and random rotation, showed no improvement in offline test performance for our training setup. Likewise, architectural regularization methods like spatial dropout and standard dropout did not yield significant gains, possibly due to the relatively small number of parameters in our network. Additionally, weight decay, a common regularization technique for deep neural networks, was

not used, as it only affects the magnitude of the proxy weights and not their sign, which does not influence the forward pass in a Binary Neural Network (BNN).

3.3 Fully BNN design for ER

Despite their efficient execution, BNNs are challenging to design and train [170]. The literature proposes various methods for sizing, designing, and optimizing BNNs [16, 124]. This section outlines five key design best practices to accommodate CIL with ER applied to FBNNs. These are based on a simple yet efficient proposed VGG-like FBNN model. Next, our FBNNs are evaluated against equivalent floating-point networks in offline settings and with CIL baseline strategies. The five best practices can be succinctly summarized as follows: using scaling factors as normalization, incorporating a learnable global average pooling bottleneck, employing a YCbCr input thermometer encoding, implementing a non-linear ternarized classifier, and applying an output scaling heuristic.

3.3.1 VGG-like feature extractor with shuffled group-conv

The feature extractor of the network (F) faces two contradictory constraints. On one hand, it needs to have a sufficient expressiveness to learn diverse semantic features from the pre-training task. On the other hand, due to hardware limitations, we cannot utilize large-scale backbone structures [74].

In this work, for the sake of simplicity and repetability, we opted for a "basic" VGG-like [197] (but efficient) structure for the feature extractor F with three blocks of two convolutions, each with n_f, $2n_f$, and $4n_f$ filters, followed by a max-pooling layer. Figure 3.3 illustrates the architecture with $n_f = 128$. In addition, inspired by ShuffleNet [235], we further reduced the overall number of parameters introducing grouped convolutions combined with channel shuffling. The number of groups is defined as $\lfloor \frac{n_{filter}}{128} \rfloor$ where n_{filter} is the number of filters in the layer. This heuristic keeps approximatively the same kernel depth through the network. In each convolution block, conventional batch normalization layers are replaced by layer-scaling factors as detailed Section 3.3.2.

This FE head is followed by a dedicated custom bottleneck and classifier which are described in the following sections.

3.3.2 Scaling factors as normalization

In the offline setting, layer-wise scaling factors efficiently replace channel-wise BN without loss in performance employing multi-stage QAT. This protocol raises two issues in CIL:

- In incremental learning, multi-stage QAT would imply re-initializing BN parameters at the beginning of each retraining (see Figure 3.4). There is no obvious way to reinitialize BN layers from rescaling factors.

- BN has been shown to exacerbate catastrophic forgetting by having its parameters, means and variances, biased towards over-represented new classes in training batch task [36, 129]. Thus it is preferable to not derive scaling factors from the BN parameters in a multi-stage QAT during retrainings.

3Mb BNN Encoder

Block	Repetitions
ConvBlock (128,1)	×1
ConvBlock (128,1)	×1
MaxPooling 2×2	×1
ConvBlock (256,2)	×1
ConvBlock (256,2)	×1
MaxPooling 2×2	×1
ConvBlock (512,4)	×1
ConvBlock (512,4)	×1
MaxPooling 2×2	×1

3Mb BNN Classifier

Block	Repetitions
DenseSkip (128,2)	×1
QuantDense(100,1)	×1
Masking	×1
Normalisation	×1

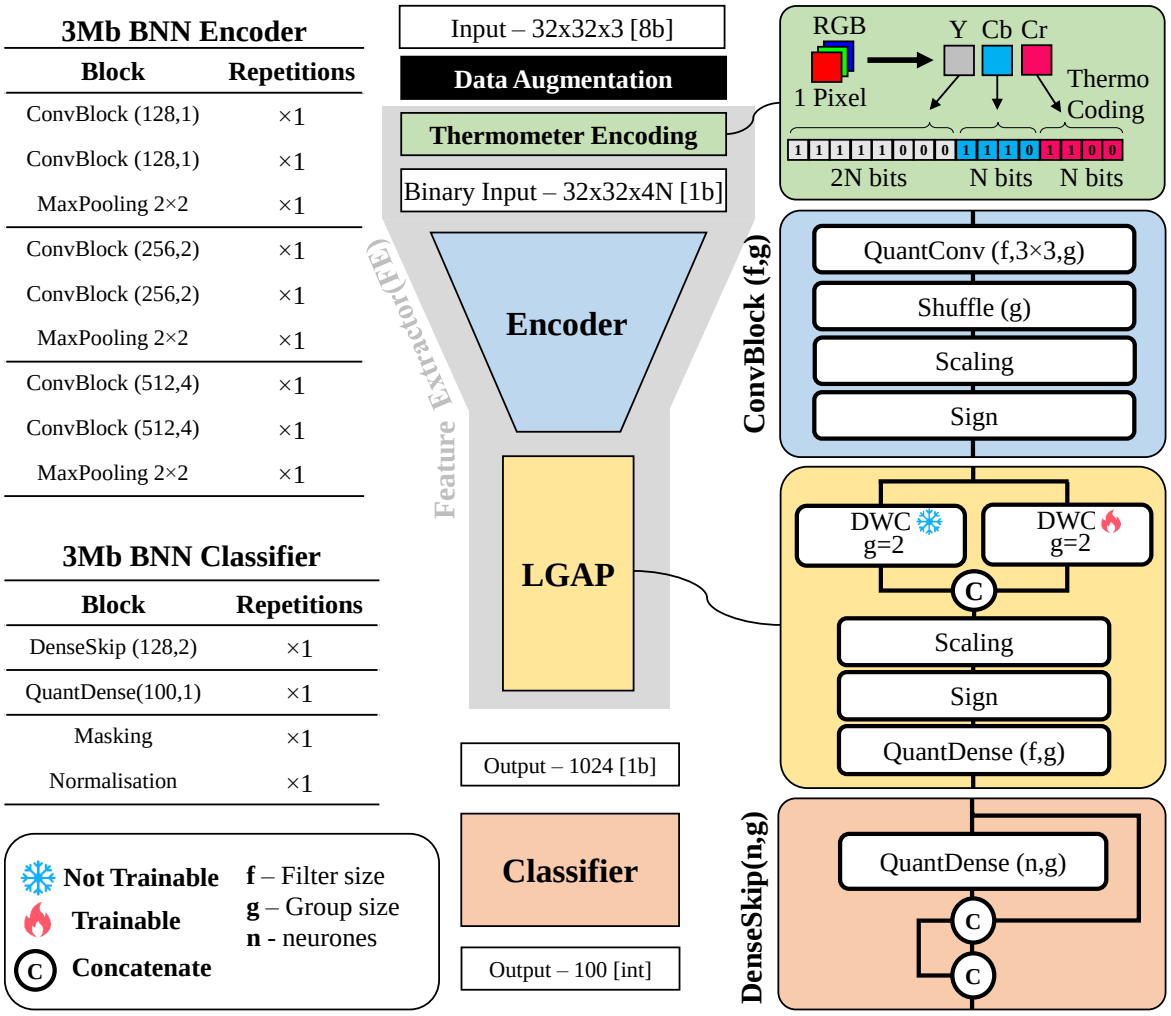


Figure 3.3: Baseline model structure. *3Mb-BNN* version reported here.

To circumvent these issues, we propose to train the network in a single-stage, and thus set the scaling factors' values not from the BN layer parameters, but utilizing the network topology itself, making them task-agnostic. Inspired by the supposed BN's goal of limiting the covariance shift [88], we aim for scaling factors that give unitary activation variance.

At the initialization, since weights and activations in their forward representation are assumed to be equally distributed between -1 and +1, it means that activation A^b and weights θ^b follow a Rademacher distribution of parameter $p = 0.5$. The resulting dot product between θ^b and A^b thus follows a binomial law of mean 0 and variance $Var = 2^2np(1-p) = n$, where n corresponds to number of inputs of the dot products, also called $FanIn$. An appropriate scaling factor S could therefore be defined depending on $FanIn$ and a certain scalar K:

$$S = \frac{K}{\sqrt{FanIn}} \tag{3.1}$$

Similarly to [188], we conducted experiments to determine the best value for K for the *3Mb-BNN* model, so that maximum train accuracy is achieved. Maximum train accuracy is desired to leave room to the network to optimize further by regularization such that

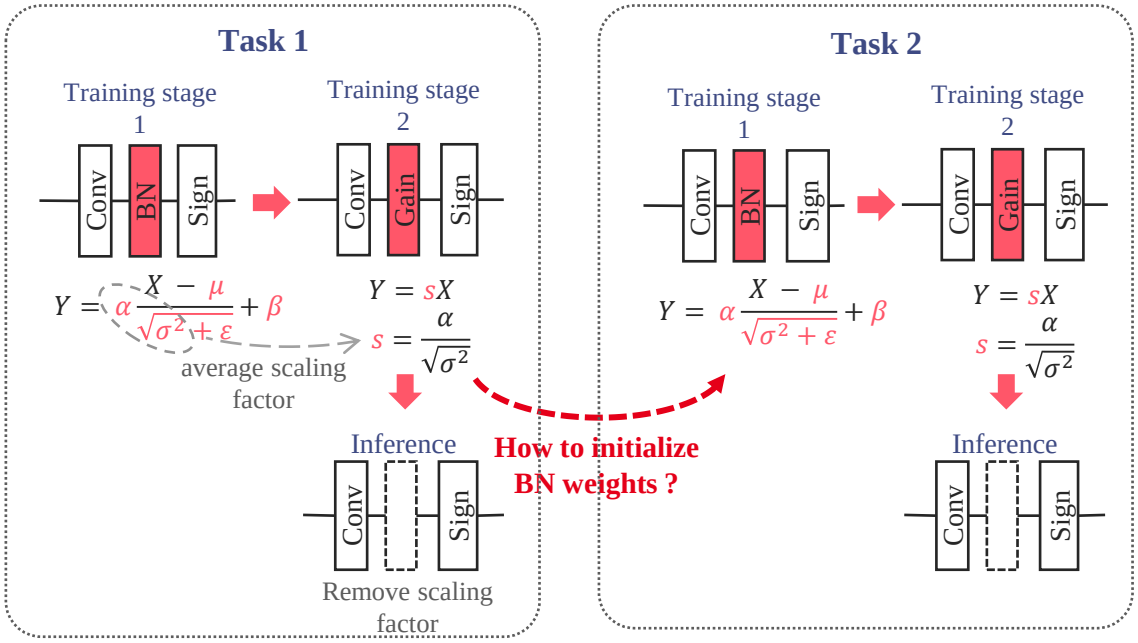


Figure 3.4: Multi-phase training protocole in an incremental learning scenario.

good test performance can be also be reached. Table 3.2 presents the train accuracy for a case with and without BN and for three K values. In our deep learning training setup, the unitary variance ($K = 1$) brings comparable results to the use of BN, in a single-stage training.

	BN	K=0.3	K=1	K=3
Train accuracy	78.7%	74.7%	78.5%	75.5%

Table 3.2: Influence of variance scaling in normalization layer.

3.3.3 Bottleneck design

Including a Global Average Pooling layer (GAP) at the end of the convolutions blocks is a common practice to both reduce the dimensionality of latent representation before the classifier and make the network robust to spatial translation of the input [121]. However, with binary features as input, the importance of a given spatial region cannot be encoded thought the activation magnitude. To keep spatial information at the output of the FE we propose to add a path where the pooling weights are learned. The whole projection block is noted LGAP (learnable GAP) and depicted in Fig. 3.3. In practice, binary weighting is learned as a Depth Wise Convolution of the spatial size of input filters. Both projections are concatenated, scaled, and binarized. Finally, features are recombined by a dense layer in a 1024 dimension binary latent space. It results in a compression factor of 24 compared to the input 8-bit image.

3.3.4 Input data encoding

A common practice in BNN design is to keep the first layer in full-precision at the cost of its computational overhead. Pixel normalization, centering and standardization are thus used to pre-process the input especially to favor a proper data scaling improving training stability. We aim for binary-only arithmetic for each layer, comprising the first and last ones. [15] proposes to code each RGB channel with a thermometer coding, denoted as $TRGB$, on 85 binary channels after a uniform quantization, resulting in an input with 255 channels. Each pixel in an image thus requires $3 \times log_2(\lceil 85 \rceil) = 21$ bits of storage in the case of ER with a buffer.

Optimization of input coding. In a fully-binary implementation, the hardware mapping can be even more eased when the number of bits per pixel are an integer power of 2. To further optimize the input data encoding, *RGB* values are first transformed into a *YCbCr* (YCC) color space and then converted into a thermometer coding, *TYCC*. Since the luminance channel, Y, contains more discriminative information that the chrominance channels, *CbCr*, we allocate 2 times more channels to the Y ($2N$) than to the C channels (N). Therefore the number of bit to code a pixel in *TYCC-N* is:

$$\mathcal{M}(pixel) = \log_2(2N) + log_2(N) + log_2(N) \tag{3.2}$$

Another important parameter to take into account is the size of the first convolution layer. In the case of our *3Mb-BNN*, the first convolution's kernel size is $3 \times 3 \times 128 \times (2N + N + N)$ in the case of *TYCC-N* input encoding. We investigate both memory requirement and number of parameters in Figure 3.5.

Dynamic range scaling. As shown in Figure 3.5a), chrominance range of values do not fully cover the $[0, 1]$ interval for *CIFAR100*, but it is also known to be generally the case for real-world pictures. To maximize bit usage after quantization we scale empirically scale the dynamic range by a factor 4. On the *CIFAR100* test dataset it gives an appropriate compromise between bit usage and saturation (Figure 3.5b)). The final color transformation is given in equation 3.3.

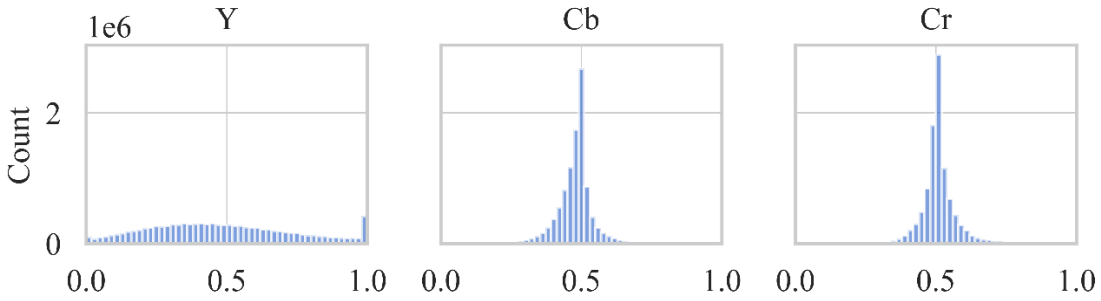
$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0 & -2 & +2 \\ 2 & -2 & 0 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \end{bmatrix} \tag{3.3}$$

We investigate the impact of thermometer coding $TRGB$ and $TYCC$ with N =4,16,32,64 on offline performance and memory requirement to store one pixel in Figure 3.6. We add TY thermometer coding, where only the luminance is kept, the evaluate the need to store chrominances.

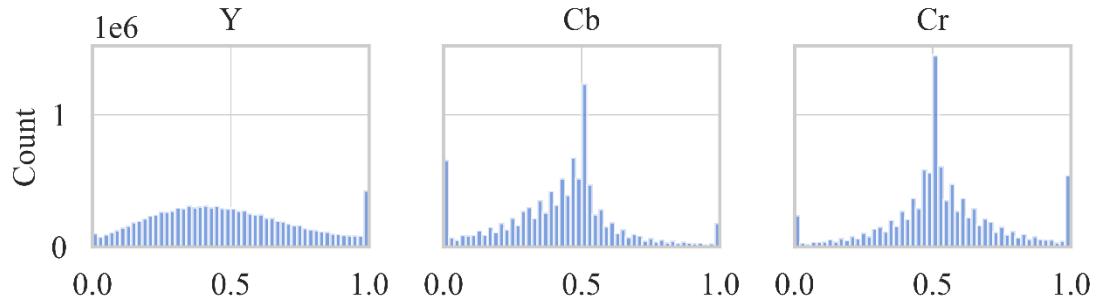
We validate that *TRGB* coding improves offline performance compared to *RGB* with input normalization and standardization. On an equal number of channels,*i.e.*, 255,*TYCC* do not improve final performance compare to *TRGB*. However with TYCC-32, the accuracy difference is within the repetability gap.

Moreover TYCC-16 results in 8 bits saving per pixel compared to TRGB, which is a 38% reduction of the Input Memory Buffer size for the same number of image sampled.

Finally we validate the necessity to keep chrominances channel with an increase of 11% in accuracy compared to luminance-only TY.



(a) Pixel value distribution in YCbCr space



(b) Pixel value distribution in YCbCr space with scaled chrominances (× 4)

Figure 3.5: Effect of scaling the chrominances. Distribution are computed on the whole *CIFAR100* test dataset.

In the rest of this work we hence utilize *TYCC-16* for it gains in terms of memory saving in native replay with it relativily low drop in performance compared to *TYCC-32* and *TRGB*.

RGB	TRGB	TYCC-64	TYCC-32	TYCC-16	TY-128
55.4%	59.5%	58.4%	59.3%	58.7%	54.0%

Table 3.3: Influence of Encoding on Offline performance.

Ternarized Non-linear Classifier

We consider that the classifier begins after the LGAP block. In latent replay, this defines the frontier between trainable and frozen layers. The question that arises is: how deep the classifier should be? On the one hand, many approaches choose a single dense layer [70] for a classifier. Hence the FE output must be linearly separable and thus it is easy to analyze. Finally, a single dense layer eases the optimization as a stack of dense layers is harder to optimize. On the other hand, in case of Latent replay, a single layer does not leave a lot of room for adaptation to future tasks. We propose to use a multi-layer classifier with skip connections as depicted Fig. 3.3 with the DenseSkip block with n neurons and g groups. In a full precision dense layer, skip connections provide shortcuts ensuring the classifier

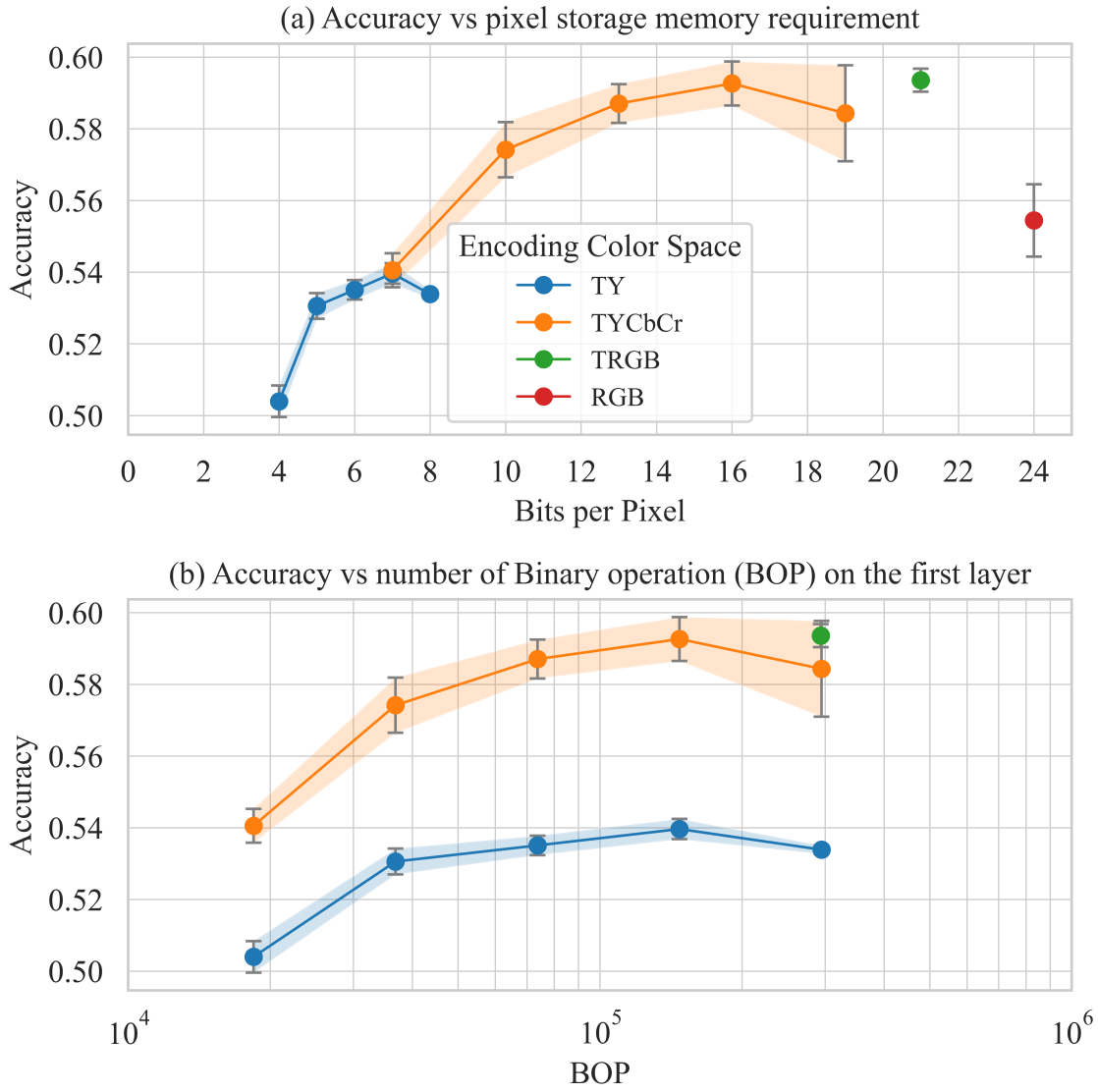
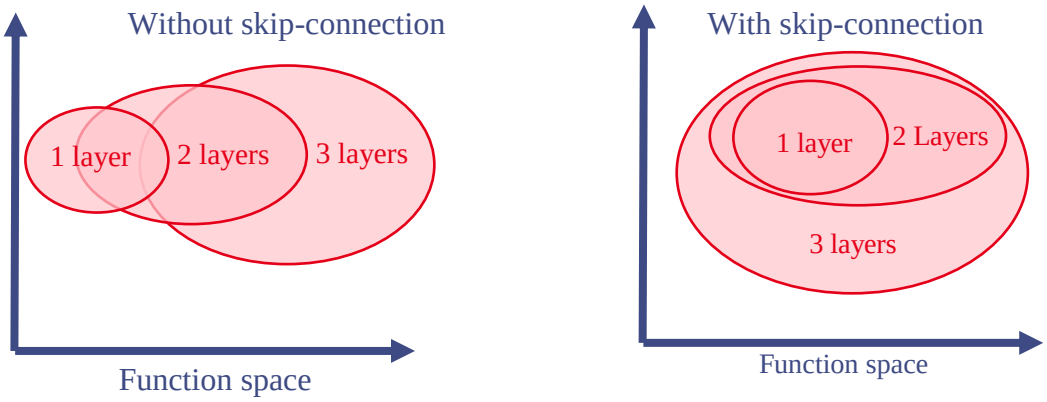
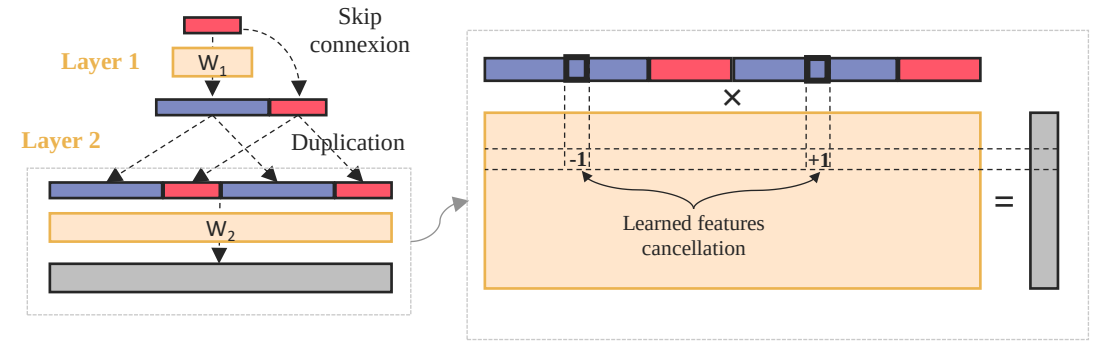


Figure 3.6: Offline test accuracy and memory requirement per pixels for different encoding strategy tested on 3Mb-BNN.



a) Function modeling capacity without and with skip-connexion



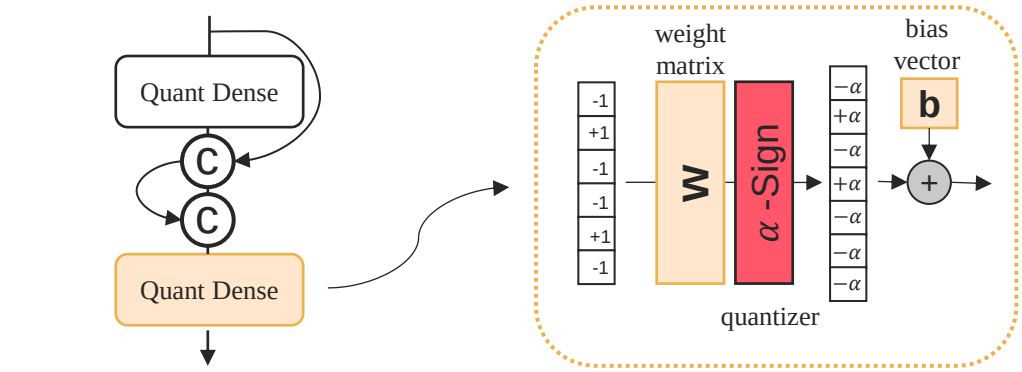
b) Ternarized Non-linear Classifier

c) Duplication for features cancelation with binary weights via fake-ternarization

Figure 3.7: Motivations and design principe behind our proposed Ternarized Non-linear classifier.

will perform at least as well as a linear one. Here, binary weights values do not allow null projections and thus do not allow path annihilation. To that end, we duplicate the input activations, so that the 0 value can be obtained by opposite sign weighting on the same activation node (Figure 3.7).

Finally, grouped dense layers (with $g = 2$ groups) are introduced to limit the classifier's model size. These layers are implemented using grouped convolutions with a 1×1 kernel along the feature dimension, as 1×1 convolutions are mathematically equivalent to dense layers. This layer design scales better with larger networks, as the number of parameters grows linearly with the number of neurons n for $g = 2$, compared to quadratic growth in a standard dense layer. Different model size of our BNN are evaluated in Section 3.3.6.



a) Our Non-linear ternarized classifier

b) Specifities of last dense layer: $\alpha - sign$ quantizer and bias vector

Figure 3.8: Particularities of the last dense layer: bias and output-scaling factor.

3.3.5 Last layer design

Instabilities occur during training without BN when the number of classes increases. After augmenting the number of classes by group of ten a few times, the training loss tends to diverges. Assuming that it comes from output gradient scaling due to the proposed activation scaling, an α-scaled binarization, *i.e.*, $\alpha \times \dot{sign}()$ is used at the end of the model to tackle convergence issues by taking into account the number of classes N_c as described in equation 3.4. The factor 5 have been found empirically by grid search to ensure convergence on *CIFAR100*. Unlike other layers, we keep biases on the last layer. Note that in case of an hardware mapping it is not the weights that are scaled but the biases with a factor $\frac{1}{\alpha}$. Results Table 3.2 are given with $N_c = 100$.

$$\alpha = \frac{1}{\sqrt{5FanIn \times N_c}} \tag{3.4}$$

3.3.6 Evaluation

This section first evaluates the proposed FBNN *3Mb-BNN* in an offline setting to put in perspective the performance in comparison with state-of-the-art BNNs, as well as with their full-precision counterparts. Second, we define and evaluate two CIL baselines, that represent the lower- and upper-bound performances for any incremental learning strategy.

Baseline

In the offline setting, in order to assess the effect of binarization, we evaluate our *3Mb-BNN* against two floating-point variants: (1) a memory-footprint equivalent network, *FPNN*$_b$, and (2) a topology equivalent network *FPNN*$_p$. Therefore, *FPNN*$_b$ and *FPNN*$_p$ have real-valued weights and activations in a floating-point (*e.g.*, considered coded on 32b). On the one hand, *FPNN*$_b$ keeps the same topology as our *3Mb-BNN* but has $32\times$ less parameters. The number of parameters reduction is done by scaling down the number of filters. For a fair

comparison, *FPNN$_b$* uses BN and conventional input normalization between $[-1, +1]$ (no thermometer coding). Sign quantization layers are replaced by standard ReLU activations. On the other hand, *FPNN$_p$* has the same architecture and the same number of parameters as *3Mb-BNN*, and hence a $32\times$ larger memory footprint while using BN. This evaluation (Fig. 3.9) comprises three network sizings: 3 Mb (*Tiny*), 9 Mb (*Medium*), and 24 Mb (*Large*). Fig. 3.3 presents *3Mb-BNN*, while 9 Mb and 24 Mb FBNNs are variants obtained by multiplying respectively by 2 and 4 the number of filters. *3Mb-BNN* is also compared to *BNN-BN* [39] and *BNN-AB* [132], two training approaches without BN but with input and output layers using full-precision. *BNN-BN* an *BNN-AB* test accuracy results both rely on using two different backbones: BiResNet18 [124] (11.7 Mb) and ReActNetA [126] (29.3 Mb).

In the CIL setting, we evaluate our model on the *CIFAR50+5X10* scenario on four experience replay baselines: *naive*, *naive-reset*, *cumulative* and *cumulative-reset*. The *naive* baselines correspond to the case of retraining only with the new task data, without any incremental learning strategy, hence highlighting catastrophic forgetting; they are thus equivalent to experience-replay with an empty buffer. The *naive* baselines offer a lower bound on the performance of any incremental learning strategy. The *cumulative* baselines correspond to the case of an infinite memory buffer, where all past labelled data are stored, hence indicating the upper-bound performance. The *-reset* suffix indicates that weights are randomly initialized at the beginning of each retraining. The *reset* baselines assess the influence of parameter initialization.

Offline comparison: binary vs. 32-bits floating-point

Fig. 3.9 reports offline accuracy as a function of the model size (Mb), indicating that *3Mb-BNN* provides better performance than $FPNN_b$ on train set and test set for the 3 Mb and 9 Mb configurations. We also note that the 24 Mb variant does not improve test accuracy compared to the 9 Mb, but it still improves the training accuracy. This plateau where increasing the model size above 10-Mb does not lead to better test accuracy, is common [16]. Large BNNs seem more difficult to regularize. *FPNN$_p$* models achieve higher test accuracy by a bit more than 10% compared to our FBNNs, while training accuracy remains capped to 100% (confirming results of [170]).

When compared to existing BNNs without BN, our *3Mb-BNN* model shows better performance than [39] with more than $3\times$ less parameters, and similar performance to [132] in the *Large-BNN* configuration. In conclusion, our FBNN reaches state-of-the-art performance in this particular offline setting.

In all the experiments of this paper we only consider *3Mb-BNN*, unless explicitly stated otherwise. Moreover, since the training accuracy does not cap to 100%, the *3Mb-BNN* is the most challenging candidate to study compact FBNN for CIL. It particularly allows to evaluate the forgetting issues in the context of a very limited expressiveness of the model.

CIL baselines analysis

Fig. 3.10 reports the accuracy, for the four experience replay baselines, on each task, per epoch, during the whole *CIFAR50+5X10* incremental dataset, for *3Mb-BNN*. Table 3.4 complements the results with metrics at the end of all 5 retrainings. When looking at the validation curve on the current task dataset, we can notice that our training protocol enables a robust adaptation at each task and for each baseline.

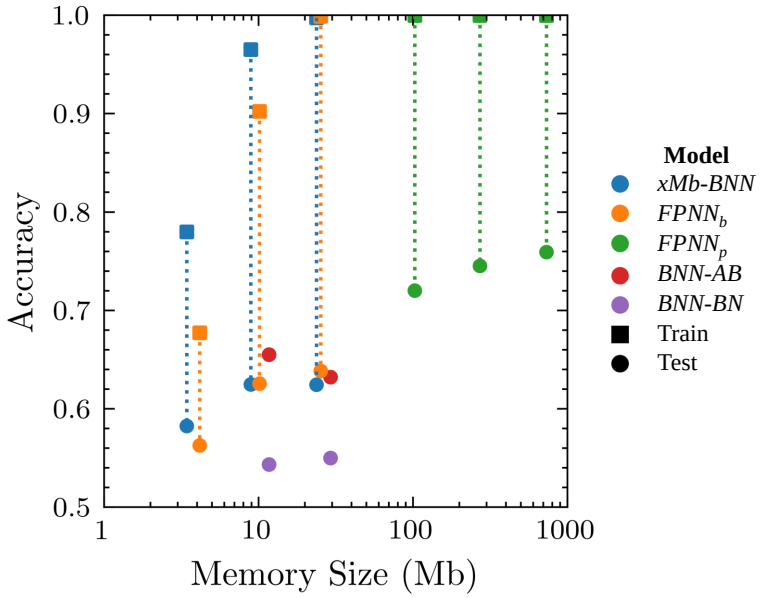


Figure 3.9: Offline accuracy for different configurations of our architecture. Test accuracy is reported with circle and train accuracy with squares.

As expected, the *naive* baselines retain only the task of the last retraining, whereas all tasks are learned in the *cumulative* baselines. For each task, t, the validation accuracy on the current task $\mathcal{D}^t$ is higher for *naive* than *cumulative*. For *cumulative*, the training includes every past task's training set, $\mathcal{D}^0, ..., \mathcal{D}^t$, whereas *naive* includes only the current task, $\mathcal{D}^t$. Therefore, simultaneous learning of multiple tasks does not contribute to improve learning performance for individual tasks. Resetting the weights in the *naive* baseline has little influence on the performance; the network forgets all the tasks except the last one regardless on how the weights are initialized.

On the cumulative baselines, weight initialization has a crucial impact. When the weights are initialized with their value from the previous training, the network achieves both higher accuracy on current and old tasks. The same result is observed in Table 3.4. $\mathrm{a_{new}}$ and $\mathrm{a_{old}}$ are significantly higher without *reset*. This indicates that *3Mb-BNN* benefits from weight initialization from the previous task, to increase performance on following tasks. At the end of all tasks, the *cumulative* setting has a 3pts increase in final accuracy compared to the *cumulative-reset* setting. Moreover, the number of epochs is reduced by 41% for the entire retraining for the 5 tasks when the weights are not reset. It indicates our FBNNs capitalize on past knowledge to better learn new tasks.

baseline	$\mathrm{a_{old}}$	$\mathrm{d_{old}}$	$\mathrm{a_{new}}$	$\mathrm{d_{new}}$	$\mathrm{a_{final}}$	$\mathrm{d_{final}}$	epochs
naive	0.0%	0	89.8%	0.06	17.9%	0.36	236
naive-reset	0.0%	0	79.7%	0.07	15.9%	0.32	428
cumulative	71.0%	0.16	67.1%	0.12	67.4%	0.15	318
cumulative-reset	66.0%	0.14	62.9%	0.13	63.9%	0.14	545

Table 3.4: Task-level metrics during last retraining (Task 4).

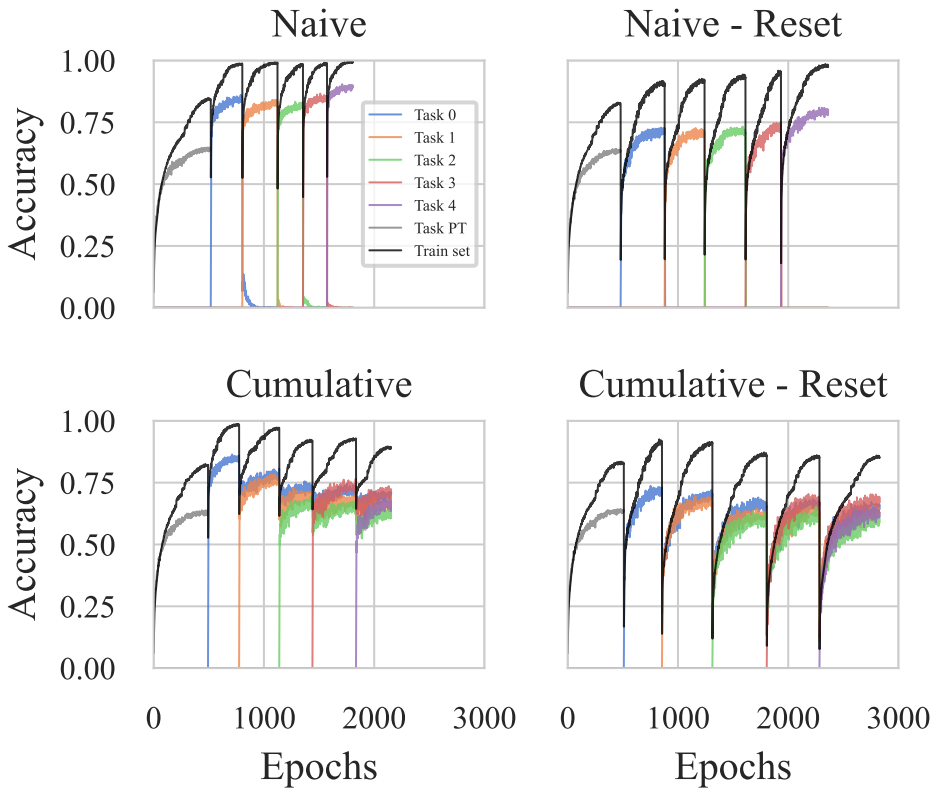


Figure 3.10: Training curves of *3Mb-BNN* on the CIL *CIFAR50+5X10*, for the four baselines strategies. Accuracy on train set is reported on black, pre-training test accuracy in grey, and the test accuracy of each task in a different colors.

3.4 Loss balancing

In general, we can consider that each incremental training session with a replay buffer is a training on an imbalanced dataset containing all seen classes $\mathcal{D}^t \cup \mathcal{B}$. Typically, there are more samples per class in the dataset of the new task t, $\mathcal{D}^t$, than in the buffer with past task's samples, $\mathcal{B}$. The common practice to compensate for an imbalanced dataset is to weight the loss. However, CIL introduces two additional biases that can challenge this common practice. The first bias concerns weight initialization at each retraining, which is biased toward past classes. The second bias concerns pre-training of model weights in Latent replay. Indeed, some real-life use-cases may require to retain the classes in the pre-training task. As the feature extractor is trained on $\mathcal{D}^{PT}$ then fixed, there is a strong bias toward the retention of the pre-training task.

As methods to deal with these two biases, in what follows we investigate: (1) the influence of inverse-frequency loss class-balancing, (2) the utilization of a focal term, and (3) the choice of the classification loss. These three factors are studied for two use-cases, Retained Pre-training Task (RPT) and Forgotten Pre-training Task (FPT), which, as the name suggests, deal with remembering of not the initial pre-training task.

3.4.1 Losses and loss-balancing strategies

Inverse class-frequency adjustment is a common strategy to solve the class-balancing problem. The loss terms associated to training samples of class $i \in \{1, ..., C\}$ are weighted by a factor w_i computed as defined in (3.5). We note the class-cardinality and the total number of samples in the re-training dataset, $\mathcal{D}^t \cup \mathcal{B}$, as n_i and N, respectively. f_i additionally denotes the frequency of a class i in $\mathcal{D}^t \cup \mathcal{B}$ as in (3.6).

We evaluate the inverse class-frequency weighting for three losses: Categorical Cross-Entropy (CCE), Focal Categorical Cross-Entropy (FCCE), and Squared-Hinge (SH).

$$w_i = C \times \frac{(f_i)^{-1}}{\sum_{j=1}^{C}(f_j)^{-1}} \tag{3.5}$$

$$f_i = \frac{n_i}{N} \tag{3.6}$$

The CCE is conventionally defined as in (3.7) where B is the batch size, $y_{i,c}$ the one-hot encoded label for sample i and class c and $p_{i,c}$ the predicted probability for class c for sample i obtained after applying *softmax* function to logits.

$$\mathcal{L}_{CCE} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c=1}^{C} w_c.y_{i,c}.log(p_{i,c}) \tag{3.7}$$

The FCCE loss [182] is a variant of the standard categorical cross-entropy loss that is designed to address class imbalance by down-weighting the loss assigned to well-classified examples, focusing more on harder-to-classify examples. FCCE introduces a tunable focusing parameter ν that adjusts how much attention is paid to difficult examples. (3.8) defines FCCE using the same notation as above:

$$\mathcal{L}_{FCCE} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{c=1}^{C} w_c(1 - p_{i,c})^{\nu} y_{i,c}.log(p_{i,c}) \tag{3.8}$$

The SH loss, designed for the Support Vector Machines (SVM) algorithm, has shown promising results with BNN [153]. Moreover, as many CIL methods use margin loss for classification [239], we consider it in the loss comparison for BNN-CIL. SH penalizes misclassifications with a quadratic cost, leading to a smoother gradient and encouraging larger margins between classes for better classification performance compared to Hinge loss. (3.9) defines SH where the label $y_{i,c}$ is in $\{-1, +1\}$ (1 if the sample belongs to class c, -1 otherwise) and $\hat{y}_{i,c}$ is the associated logits.

$$\mathcal{L}_{SH} = \frac{1}{B}\sum_{i=1}^{B}\sum_{c=1}^{C} w_c.\left(max\left(0, 1 - \hat{y_{i,c}}.y_{i,c}\right)\right)^2 \tag{3.9}$$

3.4.2 Experimental results

3Mb-BNN is evaluated on *CIFAR50+5X10* for both native and latent replay. Unlike FPT, for RPT, the classifier is not reset after pre-training and the memory buffer includes samples from $\mathcal{D}^{PT}$. As in [182], we set $\nu = 2$ for FCCE loss.

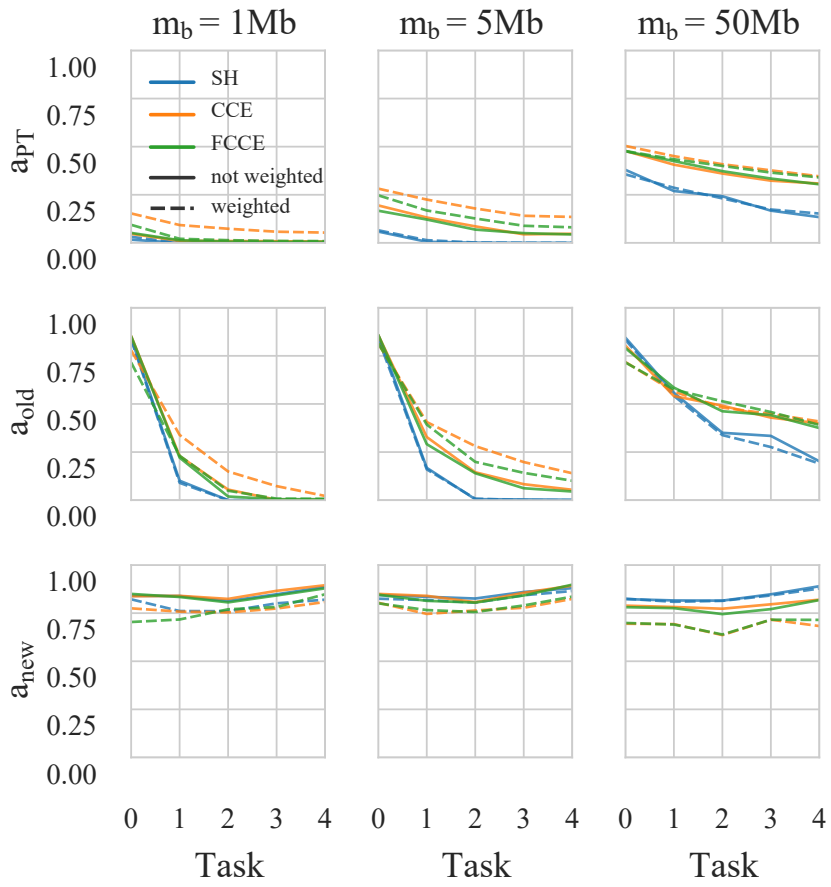


Figure 3.11: Loss investigation on native replay in RPT scenario.

The SH loss

With respect to Native replay, Fig. 3.11 and 3.13 indicate that SH leads to better performance on the current task, a_{new}, than other losses, at the expense of deteriorating retention, a_{old}, regardless of buffer size. SH seems to exacerbate plasticity to the point of degrading final accuracy. Figures also indicate that SH is relatively insensitive to class-weighting, likely because it attempts to find margins under a one-vs-all classification problem. In Latent replay, even with a 50Mb buffer, SH loss exhibits convergence issues on current tasks while accuracy on pre-training stays high.

The cross-entropy losses

Let's first analyze the adaptation and retention capacity in Figures 3.11. As expected, inverse frequency balancing improves retention, *i.e.*, a_{old}, for both Native and Latent replay, especially with smaller buffers. By giving higher weight w_i to the under-represented past classes, the model preserves knowledge from earlier tasks, enhancing retention. However, this comes at the cost of reduced adaptation to new classes, *i.e.*, a_{new}, as the increased focus on past data limits the model's ability to adjust to new information. Latent replay is more sensitive to balancing. For instance, we can observe a 41.0% reduction in adaptation on the

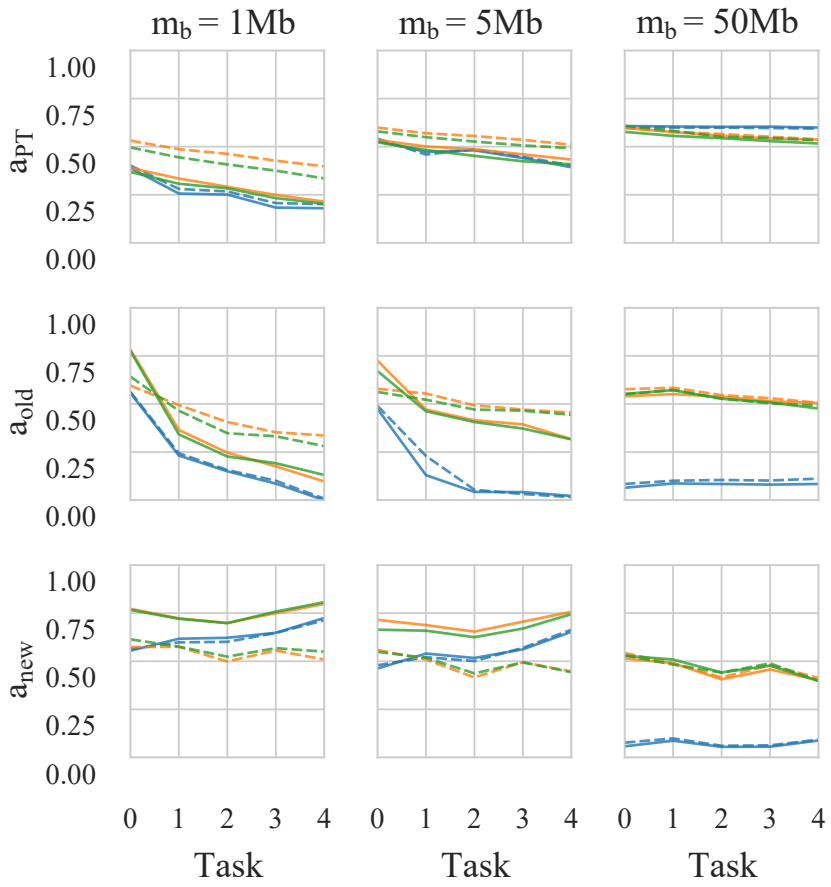


Figure 3.12: Loss investigation on latent replay in RPT scenario.

last task for Latent replay compared to a 7.7% reduction for Native replay with CCE loss in RPT scenario m_b=5 Mb.

Secondly, the final performance is shown in Table 3.5 and Table 3.6 for RPT and FPT. For clarity, we report final metrics for Latent 5 Mb and Native 50 Mb, *i.e.*, a similar buffer capacity in terms of number of samples. In both configurations, approximately 50 samples per class are stored. In both configurations, loss balancing leads to improvement of a_{final}. Particularly in the RPT scenario, loss balancing increases the final performance by at least $+5$pts, in Latent replay (5Mb) for both losses. Weighting the loss decreases class dispersion by a factor of 9.7% in average for FPT scenario and by 18.2% in RPT scenario. Loss balancing is thus effective at both enhancing final accuracy and reducing class dispersion in both scenarios.

Loss focalization

The focal loss does not seem to bring any substantial advantage in the final accuracy (Table 3.6 and 3.5). However, it reduces classification dispersion at the cost of small final accuracy decrease. If the use-case where the CIL is employed requires that each task is classified with (more or less) the same accuracy, one can consider utilizing FCCE.

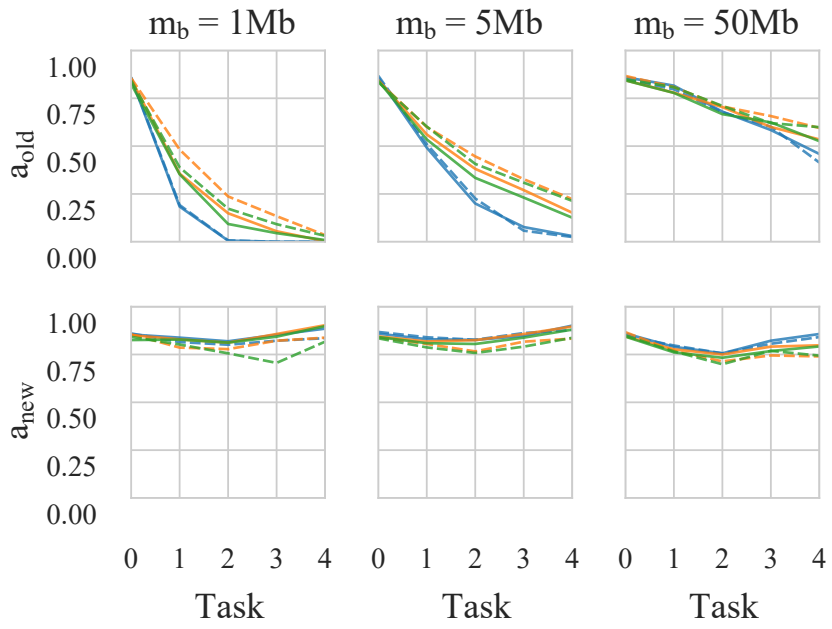


Figure 3.13: Loss investigation on native replay in FPT scenario.

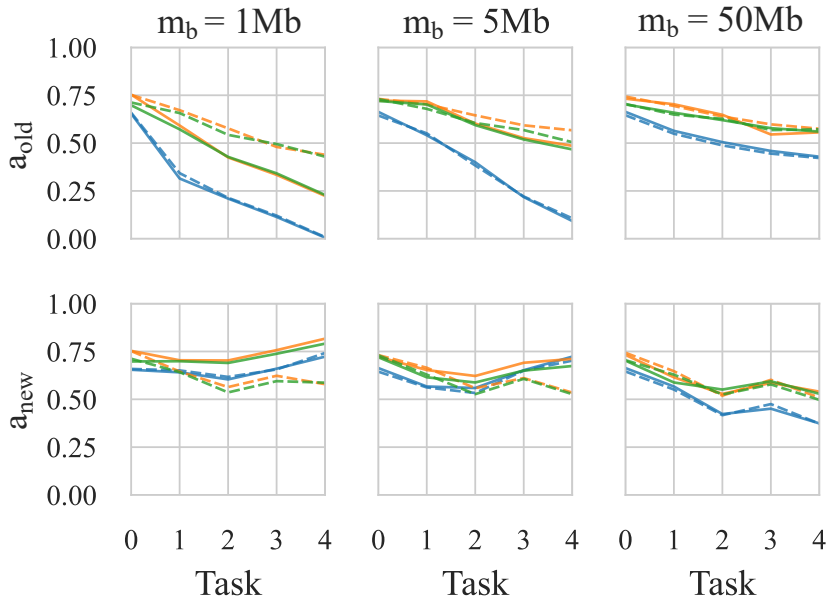


Figure 3.14: Loss investigation on latent replay in FPT scenario.

Main takeaways

Despite being a good practice in offline learning in BNN, the SH loss is not highly relevant for FBNN-CIL as it might considerably impair the final accuracy. According to these preliminary results, the common CCE loss suits well FBNN-CIL. In this case, balancing the

strategy	loss	weighted		not weighted	
		a_{final}	d_{final}	a_{final}	d_{final}
Latent (5Mb)	SH	28.0%	0.30	27.4%	0.29
	CCE	45.5%	0.18	40.9%	0.22
	FCCE	44.7%	0.16	38.7%	0.21
Native (50Mb)	SH	23.5%	0.28	22.3%	0.29
	CCE	40.2%	0.19	38.1%	0.23
	FCCE	38.9%	0.19	37.5%	0.22

Table 3.5: Comparison of balancing strategies in the RPT scenario.

strategy	loss	weighted		not weighted	
		a_{final}	d	a_{final}	d
Latent (5Mb)	SH	25.6%	0.28	25.6%	0.29
	CCE	51.6%	0.19	48.6%	0.22
	FCCE	48.7%	0.17	47.1%	0.19
Native (50Mb)	SH	48.8%	0.28	48.6%	0.28
	CCE	57.5%	0.19	56.5%	0.21
	FCCE	57.5%	0.19	54.5%	0.20

Table 3.6: Comparison of balancing strategies in the FPT scenario.

loss significantly enhances performance in every configuration while slightly compromising the adaptation, *i.e.*, decreasing the accuracy on the new task. The dispersion in per-class performance can be reduced by adding a focal term. Latent replay appears more sensitive to the type of loss especially on a small buffer (*e.g.*, 1Mb) than Native replay. In the rest of this paper all experiments use the CCE loss with inverse-frequency class-weighting. In the following sections we focus on FPT, where the pre-training only learns the FE.

3.5 Semi-supervised pre-training of the FE

A common practice for learning feature representation is to perform supervised learning on a pre-training task, with data $\mathcal{D}^{PT}$. In this manner, extracted features are dimiscrinative for the pre-training classes, however, there is no guarantee that they are also discriminative for the tasks that are incrementally learned after the pre-training, named downstream tasks. This is particularly problematic for conventional Latent replay, in which the feature extractor is not updated. In this section we challenge the common fully supervised pre-training approach, elaborating an approach to learn richer, transferable features. We first discuss Self-Supervised Learning (SSL) as a promising task-agnostic alternative. We then propose an activation regularization term that promotes features diversity. Finally we evaluate the influence of both on the *CIFAR50+5x10* dataset.

3.5.1 Multi-objective approach

Self-Supervised Regularization

Learning representation with self-supervision can boost performance in CIL settings [242][166]. Introducing a proxy task, such as clustering [34], rotation prediction [62], contrastive loss [75], or maximizing the information content of the embedding [233], makes the feature representation more task-agnostic. SSL thus prevents FE to over-specialize on a given task.

To validate the interest of SSL for BNN pre-training, we adapt the state-of-the-art Barlow Twin (BT) loss [233]. SSL on BNN is a recent research field and early work [194] has highlighted the difficulties to adapt contrastive approaches, *e.g.*, MOCO-V2 [75], without complex distillation loss and multi-phase training. Furthermore, existing results are obtained on larger BNN architecture with BN layers. To be compatible with our compact FBNN with single-phase training we look for self-supervised approach that adds minimal mechanisms. The BT approach achieves state-of-the-art performance with minimal computational overhead. Unlike other methods, the BT has the advantage to work with small batch size, which decreases the chance of training issues in FBNN.

The overall approach is illustrated in Fig. 3.16. Two augmentations, x^A and x^B, of the same image batch x, are passed through a feature extractor (FE) followed by a projection block. Let the outputs of the projection block be H^A and H^B, corresponding to the projections of x^A and x^B, respectively. The cross-correlation matrix $\mathcal{C}$ between the two outputs is computed along the batch dimension:

$$\mathcal{C}_{ij} = \frac{\sum_b H^A_{b,i} H^B_{b,j}}{\sqrt{\sum_b (H^A_{b,i})^2}\sqrt{\sum_b (H^B_{b,j})^2}}, \tag{3.10}$$

where b indexes the batch samples, and i, j index feature dimension respectively in projections H^A and H^B.

The self-supervision proxy task minimizes the distance between $\mathcal{C}$ and the identity matrix $\mathcal{I}$. The loss function $\mathcal{L}_{BT}$, adapted from Barlow Twins, is decomposed into two terms, as shown in Eq. (3.11). The first term encourages the diagonal elements of $\mathcal{C}$ to be close to 1, enforcing feature invariance to data augmentation. The second term penalizes off-diagonal elements, reducing redundancy among features. The second term is weighted by $\lambda = 10^{-5}$, as in the original paper:

$$\mathcal{L}_{SSL} = \sum_i (1 - \mathcal{C}_{ii})^2 + \lambda \sum_i \sum_{j \neq i} \mathcal{C}^2_{ij}. \tag{3.11}$$

Feature Regularization

Binary latent features often exhibit strong intra-class correlations. To address this, we hypothesize that encouraging these features to be more evenly distributed across the binary latent space (i.e., reducing their correlation) can help capture a broader range of characteristics while potentially easing the optimizer to converge. To achieve this, we introduce a regularization term, $\mathcal{L}_{FR}$, which promotes opposing binary values among components within a batch of samples.

Let $z \in \{-1, 1\}^{B \times D}$ represents the latent binary features in a batch of $\mathcal{B}$ samples with $\mathcal{D}$ the features dimension. This activation regularization loss, defined in Eq. (3.12), minimizes

the sum of the component values along the batch axis $\mathcal{B}$ for each of the $\mathcal{D}$ latent features, encouraging a balance between -1 and +1 values:

$$\mathcal{L}_{FR} = \frac{1}{D \times B} \sum_{d=1}^{D} \sum_{b=1}^{B} z_{b,d}. \tag{3.12}$$

Figure 3.15 illustrates the intuition behind $\mathcal{L}_{FR}$ on an early version of *3Mb-BNN* (without output scaling; see Section 3.3.5). Insufficient regularization leads to degenerate latent features fixed at -1 or +1, which provide no useful information for classification. Inversely, excessive regularization disrupts intra-class correlations, resulting in noise-like features. Striking the right balance is critical for preserving discriminative information while enhancing classifier performance and generalization.

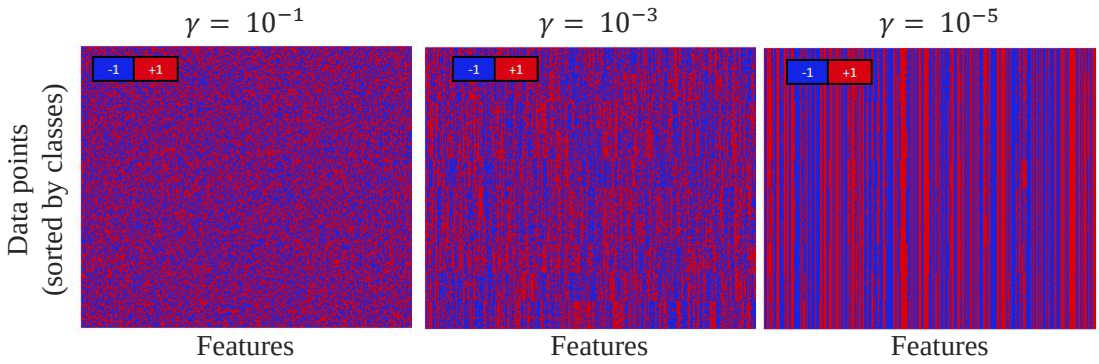


Figure 3.15: Qualitative effect of feature regularization on features correlations.γ denotes the weighting factor of the regularization term $\mathcal{L}_{FR}$.

Multi-objective weighted loss

SSL and latent features regularization adds two proxy tasks that require no label, which we believe is key to learn more transferable features. However, since the data are annotated, we might as well use the labels. Moreover, Section 3.4 indicates that that supervised learning with CCE is a good baseline for representation learning. We therefore propose a combination of both supervised and self-supervised learning, and hence the loss minimization is formulated as a multi-objective optimization problem. $\mathcal{L}_{CCE}$, $\mathcal{L}_{SSL}$ and $\mathcal{L}_r$ are then combined in a single loss with α, β and γ weights:

$$\mathcal{L}_{PT} = \alpha\mathcal{L}_{CCE} + \beta\mathcal{L}_{SSL} + \gamma\mathcal{L}_{FR} \tag{3.13}$$

The proposed pre-training loss is computed as described in Fig. 3.16. $\mathcal{L}_{CCE}$ and $\mathcal{L}_{FR}$ are computed using the previously mentioned data augmentation T_p (Section 3.2) on each input batch X. Since $\mathcal{L}_{SSL}$ requires a stronger data augmentation, we thus added random color transformations and random crop, denoted as T_c. As the pre-training can be done without compute resource constraints, the BT SSL projection block is composed of the stack of two full-precision dense layers with 2048 neurons followed by a BN and a ReLU activation.

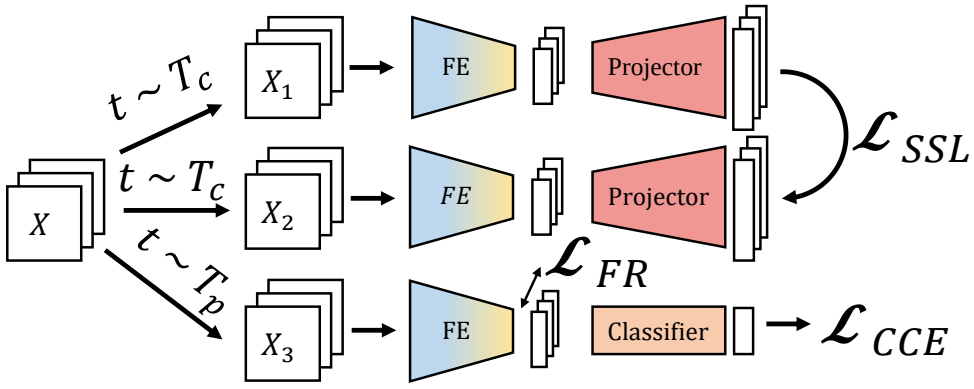


Figure 3.16: Multi-objective pre-training framework.

3.5.2 Experimental results

Loss terms compromises are evaluated on *CIFAR50+5X10* for Latent replay, a 10Mb memory buffer, the *3Mb-BNN* model and a class-balanced CCE. Table 3.7 reports metrics for the end of the pre-training stage and at the end of the 5 retrainings.

paradigm	α	β	γ	a_{PT}^{train}	a_{PT}^{test}	d_{PT}	a_{final}^{train}	a_{final}^{test}	d_{final}
supervised	1	0	0	90.74%	62.90%	0.15	56.33%	51.28%	0.17
regularization	1	0	10^{-3}	90.82%	64.06%	0.14	55.78%	51.86%	0.18
	1	0	10^{-2}	90.52%	63.26%	0.14	55.58%	51.96%	0.19
	1	0	10^{-1}	86.85%	61.04%	0.14	51.97%	48.62%	0.19
SSL	1	10^{-6}	0	90.09%	64.16%	0.13	56.47%	51.08%	0.18
	1	10^{-5}	0	89.57%	63.58%	0.14	57.63%	51.92%	0.18
	1	10^{-4}	0	86.12%	60.34%	0.16	57.88%	51.26%	0.18
	1	10^{-3}	0	63.84%	48.40%	0.17	50.04%	41.76%	0.19
reg. +SSL	1	5.10^{-6}	5.10^{-3}	90.24%	63.54%	0.14	56.53%	52.34%	0.20
	1	10^{-5}	10^{-2}	90.02%	63.51%	0.14	57.43%	52.45%	0.19

Table 3.7: Final retraining performance of *CIFAR50+5X10* using Latent replay (10Mb buffer) for various regularization setups.

The SSL term

The BT loss effectively improves the final performance for $\beta = 10^{-5}$, for both train and test accuracy (+1.3pts and +0.7pts). Hence, the representation learned with SSL is better suited to downstream tasks. Indeed, SSL eases the convergence on new tasks, boosting generalization on the test set. The SSL term also enhances the test accuracy on the pre-training task, a_{PT}^{test}, while reducing the train accuracy, a_{PT}^{train} for $\beta \leq 10^{-5}$. This observation indicates that the SSL term forces the representation to be less discriminative for the training set and it helps regularizing the network.

The latent features regularization

Feature regularization improves both pre-training test accuracy and final test accuracy for $\gamma \leq 10^{-2}$ by 1.16pts and 0.58pts. A too strong regularization impairs both initial and final performances.

However, contrary to $\mathcal{L}_{SSL}$, $\mathcal{L}_{FR}$ improves the pretraining accuracy and reduces the final accuracy on train set. It indicates that the loss helps the optimization on the pre-training task but not on downstream tasks. This can be alleviated by combining both SSL and feature regularization loss with best performance reached with $\beta = 10^{-5}$ and $\gamma = 10^{-2}$ increasing final test accuracy by 1.17pts. Note that the best configuration (α, β, γ) tested appears to be close to the optimum reached for β and γ when used independently.

Main takeaways

This experiment demonstrates the direct impact of pre-training quality on CIL performance. Alternative learning paradigms, such as self-supervised learning, provide efficient approaches to obtain transferable features without adding significant computational overhead during deployment. While these results remain preliminary, it is reasonable to anticipate that these training methods could have a greater influence in real-world applications, particularly with larger inference models that benefit from stronger regularization.

3.6 Native vs Latent replay

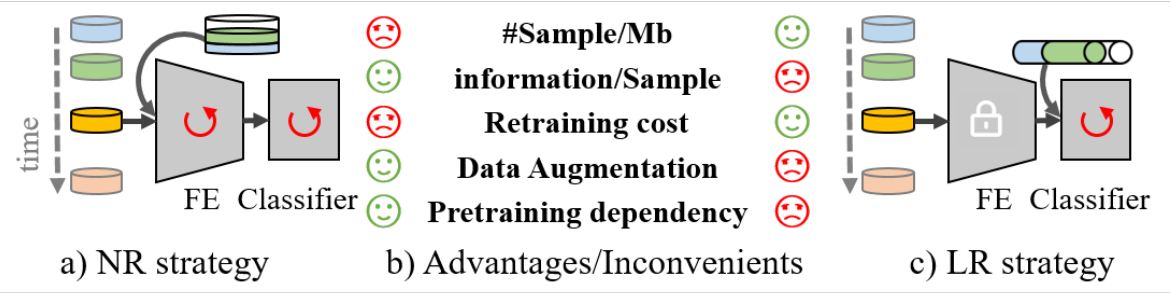


Figure 3.17: Principle of NR (a) and LR (c), and their pros and cons (b).

This section examines the performance of native replay and latent replay under equivalent memory constraints, referred to as "at iso-memory".

We begin by discussing the qualitative trade-offs between these two approaches, summarized in Figure 3.17. The analysis includes two key studies:

1. **Preliminary experimentation:** Using a simpler VGG-like FBNN, we conduct an initial investigation in Section 3.6.2. This study explores the impact of memory size, the number of retrainings, and the number of pre-training classes within a fully pre-trained (FPT) scenario. Additionally, we compare the performance of FBNNs against equivalent floating-point neural networks to establish a baseline.

2. **Comprehensive evaluation with *3Mb-BNN*:** Building on the insights from the preliminary experiment, we evaluate native and latent replay "at iso-memory" using the *3Mb-BNN* model. This evaluation incorporates the best practices discussed in Sections 3.4 (loss balancing) and 3.5 (pre-training strategies).

3.6.1 Tradeoffs between latent and native replay

The trade-offs between native replay and latent replay are summarized below, highlighting their respective advantages and disadvantages across key aspects:

- **Number of samples per Mb:** Latent replay can store $\times 13$ more samples per megabit compared to native replay, allowing it to better approximate past distributions under the same memory constraints, particularly as the number of classes increases.
- **Reconfigurability:** Latent replay increase in samples storage capacity in the buffer is obtained at the cost of a decrease in model reconfigurability. The latent representation can not be updated, otherwise the samples in the buffer would not be representative anymore of the image classes. In particular, this problem is referred to the aging problem in the relevant literature.
- **Information per sample:** Native replay operates on raw input data, retaining richer information per sample, which directly contributes to better learning quality with a trainable feature extractor. Latent replay, however, relies on compressed latent features optimized for discriminative tasks, thus containing less information.
- **Computational cost:** Latent replay significantly reduces computational costs by requiring 90% fewer parameters to update and achieving faster convergence with fewer training iterations than native replay (see Section 3.3).
- **Data augmentation:** Native replay offers the possibility to augment stored data in buffer with standard data augmentation, which is vital for compact networks. Data augmentation is particularly needed on the memory buffer as only a few samples are kept. Latent replay, in contrast, lacks a conventional method for augmenting binary latent representations in the buffer, limiting its flexibility in this regard. Note that data from the current task can still be augmented at the input of the fixed Feature extractor.
- **Pre-training dependency:** Native replay is less dependent on pre-training quality, as it directly processes raw inputs without relying on pre-trained feature extractors. Latent replay, however, highly relies reliant on effective pre-training, as the quality of latent representations directly impacts the relevance of stored samples.

3.6.2 Evaluation on a Simple 4Mb-BNN

Evaluation set-up

For this study, we use a simplified version of the *3Mb-BNN*, illustrated in Figure 3.18 as *Vanilla-4Mb-BNN*. Unlike *3Mb-BNN*, this model employs *TRGB* for input coding, a bottleneck with a trainable path using a depthwise convolution (DWC) with multiplier coefficient

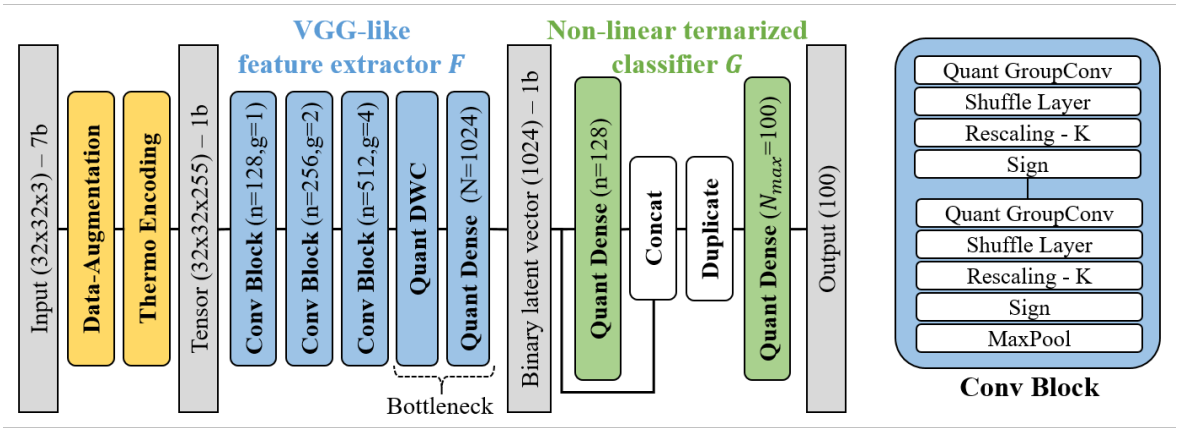


Figure 3.18: Baseline model structure. *Vanilla-4Mb-BNN* version reported here.

of 2, and standard (non-grouped) dense layers, resulting in a 4.1Mb BNN.

We assess classification performance using the precision, recall, and accuracy metrics. Precision and recall provide insights into performance spread among classes as details in Section 3.2.2

The training protocol is similar to previous sections. We use the focal categorical cross entropy as optimization loss. Experiments are conducted under the FPT scenario, where the first 50 classes are used exclusively for pre-training, and retraining is evaluated on the remaining 50 classes divided into tasks. We denote the experiments as *pt50* when pre-training is performed on all 50 classes and *pt10* when pre-training is limited to 10 randomly selected classes from the pre-training dataset.

The offline baseline, trained once on all retraining tasks dataset, gives the upper-bound performance. The Cumulative baseline (denoted as "cumu") discards the limited sampling effect by allocating infinite memory for the buffer. As a reference for our BNN, we evaluate a FPNN with $\times 32$ fewer parameters (4.1M versus 122k) to ensure similar memory-footprint. It has the same structure with [32, 32, 128] filters and [128, 64] hidden neurons for dense layers, including BN after each convolution layer.

Experiment 1: influence of memory-footprint

This experiment evaluates NR and LR across buffer sizes ranging from 1 sample per class to full-dataset storage, corresponding to memory footprints of [50kb–24Mb] for LR and [1Mb–543Mb] for NR. Figure 3.19 shows the final average recall and precision after training on 5 tasks, each introducing 10 new classes. Results are provided for both the test set and the buffer set.

A 1-standard-deviation band highlights the variation in metrics across classes. Additionally, we include an ideal LR baseline (*Ideal LR (pt100)*), which represents the scenario where F is pre-trained on the entire dataset with the classifier reset.

The average classification performance improves with buffer size, reaching the *offline* upper bound of 59% precision and 60% recall in NR. For LR, when the buffer exceeds 0.5Mb, average recall increases while average precision plateaus. Figure 3.19.b highlights the im-

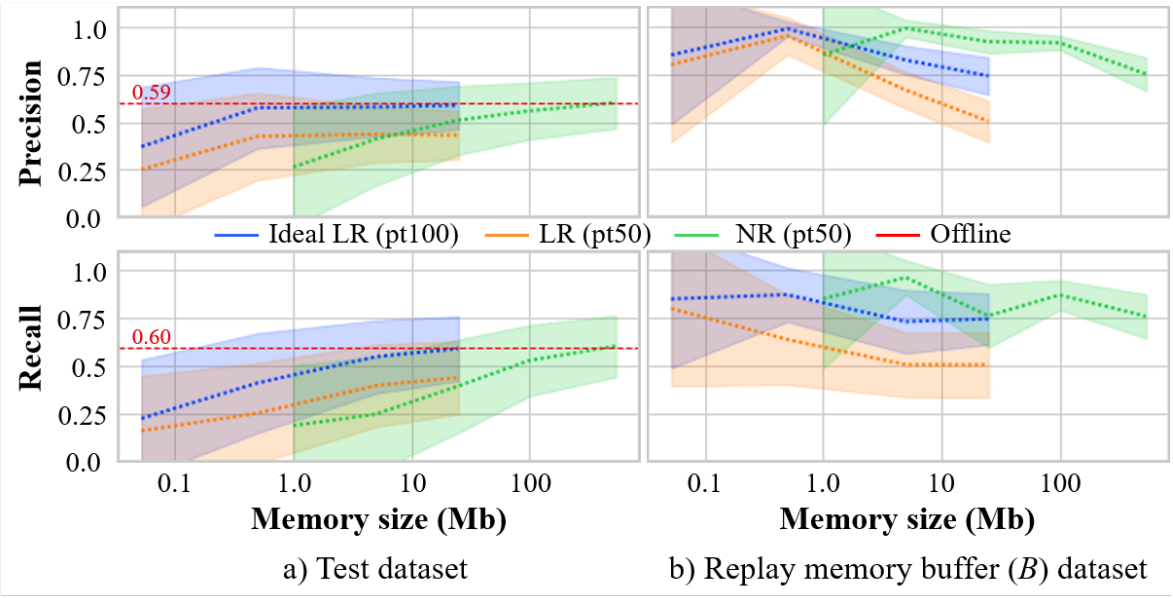


Figure 3.19: Fixed-memory size comparison between NR and LR on *Vanilla-4Mb-BNN*.

pact of buffer-based regularization. Although loss class-weight balancing ensures buffer accuracy should remain constant, the observed performance drop with larger buffers might stem from increased sample variance, which is harder to fit.

A deeper analysis of the results reveals a turning point where the performance of LR and NR intersects: at 5Mb for precision and 50Mb for recall. For small buffer sizes, LR outperforms NR, but this trend reverses for larger buffers. Notably, recall and precision on the buffer are higher for NR than LR, suggesting overfitting to buffer samples in NR, which may explain its lower overall average performance.

For larger buffers, *LR (pt50)* remains capped at 43% recall, whereas *Ideal LR (pt100)* achieves the *offline* baseline. This indicates that features learned on $\mathcal{D}^{PT}$ are not fully discriminative for classes in retraining tasks $\mathcal{D}^{i}$, making the pre-training of F the primary performance bottleneck in LR.

Experiment 2: impact of the retraining

Increasing the number of retrains can exacerbate undesirable phenomena, such as overfitting and proxy weights drift, especially for BNNs. We explore several configurations involving 25 tasks, each consisting of 2 classes, in addition to the previous 5-task scenario. For both our BNN and FPNN models, we assessed the Cumulative baseline (*NR pt50 - cumu*), as well as LR (*LR pt50 – 25Mb*) and NR (*NR pt50 - 25Mb*), using a 25Mb Memory buffer. The average test accuracy at the end of each task is depicted in Figure 3.20 and results are reported Table 3.8.

A 25Mb LR pre-trained on only 10 classes is added to further explore the impact of pre-training on the number of tasks to be learned.

Figure 3.20 indicates that the number of retraining instances has minimal impact on final accuracy, regardless of the configuration. This finding contrasts with common reports in incremental learning literature [18][179]. The state-of-the-art does not offer sufficient

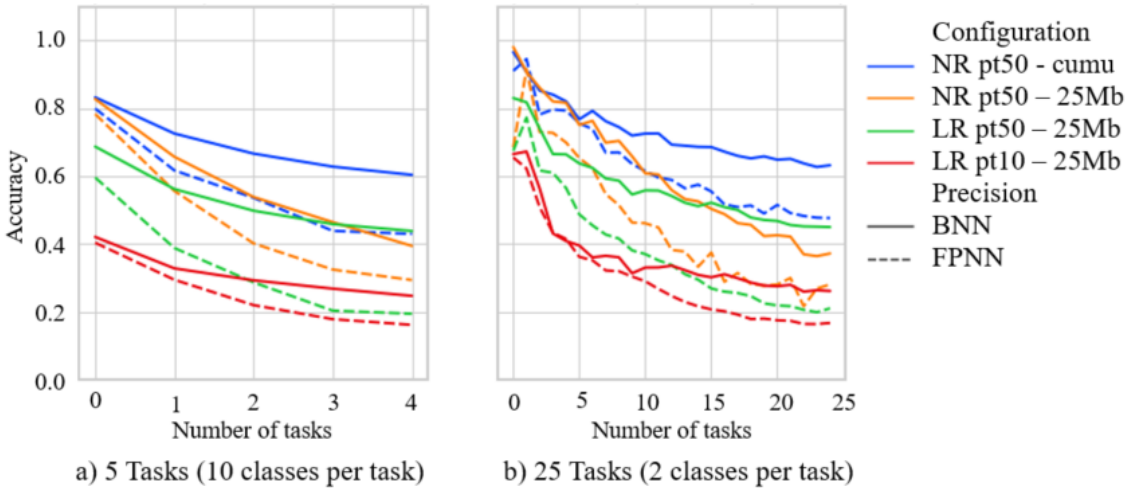


Figure 3.20: Effect on number of successive incremental tasks.

insight in the training protocols, making it challenging to fully explain these phenomena.

A possible explanation is that the advanced LR scheduler and focal loss contribute to consistent convergence, regardless of the number of retraining steps or data splits. Notably, Figure 3.20.b shows *NR pt50 - 25Mb* outperforming its LR counterpart in the first 13 tasks, highlighting the importance of retraining step considerations when selecting a replay strategy.

Model	Config.	$\mathcal{B}$ size (Mb)	Final accuracy		Final dispersion	
			5 tasks	25 tasks	5 tasks	25 tasks
BNN	NR pt50	543	0.60	0.63	0.16	0.15
	NR pt50	25	0.39	0.38	0.24	0.17
	LR pt50	25	0.44	0.45	0.19	0.19
FPNN	NR pt50	543	0.43	0.48	0.21	0.20
	NR pt50	25	0.29	0.28	0.28	0.20
	LR pt50	25	0.20	0.21	0.20	0.23

Table 3.8: Detection metrics for BNN and FPNN under 5 and 25 tasks.

Comparing FPNN and BNN, FPNN's performance declines more sharply as tasks progress. This is likely due to FPNN having fewer parameters than BNN for the same memory footprint, limiting its capacity to learn as the number of classes grows. For the Cumulative baseline, the final accuracy differs by 17% between the two models.

As expected, pre-training on a smaller dataset results in weaker performance in LR (pt10 vs. pt50), underscoring the critical role of the pre-training phase in this scenario.

Main takeaways

On *CIFAR100*, our results show that Latent Replay outperforms Native Replay when memory is highly constrained. Furthermore, BNNs can achieve a 10% higher incremental learning performance compared to an FPNN with the same memory footprint.

These findings highlight the strong potential of BNNs for adaptive, ultra-low-power embedded devices, making them well-suited for scenarios requiring efficient learning under tight memory and energy constraints.

3.6.3 Evaluation on BNN-3Mb

In this second experiment, we extend the iso-memory comparison between latent and native replay using our advanced *3Mb-BNN* model, incorporating the best practices outlined in Sections 3.4 and 3.5. The study is conducted in the more challenging RPT scenario, providing a deeper analysis of the model's adaptation and retention capabilities.

Fig. 3.21 presents the impact of the replay buffer size on the final performance for Native and Latent replay with *CIFAR50+5X10*: (a) a_{seen} represents the final performance on all seen classes, (b) a_{new} represents the adaptation capacity, *i.e.*, on new training data (without buffer samples), and (c) a_{buffer} represents the retention capacity, *i.e.*, on past classes. The previously presented loss balancing and pre-training are applied. To exhibit the influence of the buffer size on the FE training policy, the Latent and Native strategies are evaluated for buffer sizes ranging from 0 to the entire dataset.

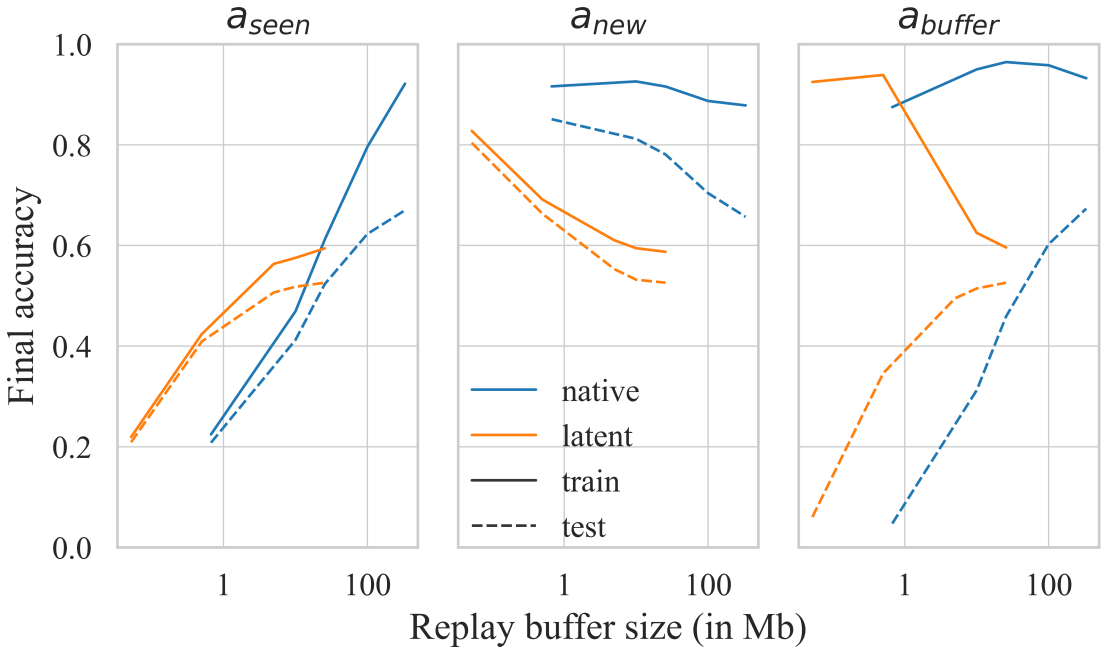


Figure 3.21: Iso-memory comparison between native and latent replay.

Final performance: a_{seen}

We observe that, under 25Mb, Latent replay offers the best performance. Above 25Mb, Native replay gives better performance. One can note that the 25Mb point corresponds to the upper bound for latent memory buffer, *i.e.*, all the training dataset can be stored, it brings no improvement to Latent replay to have a larger memory size.

Adaptation: a_{new}

When we consider the performance on new classes, *i.e.*, $\mathrm{a_{new}}$ accuracy, two distinct interpretations of replay regularization can be made. For latent replay, the augmentation of $\mathrm{m_b}$ decrease both train and test performance by the same magnitude (approximately 30% from 1 sample/image to full dataset). Learning new classes is getting more difficult even if the weighting loss is supposed to even out the effect of unbalanced training set. On the contrary, for Native replay only the test accuracy decreases. The FBNN still has the ability to model the new training distribution but it has lost its ability to generalize on it.

Retention: a_{buffer}

When looking at the test accuracy on old classes, *i.e.*, dashed lines, we first note, as expected, a sharp increase in retention for larger buffer size for both strategy. However, when the FE is non trainable, latent replay performance decreases on a larger buffer, highlighting the limited modeling capacity. On the contrary, a trainable FE (native replay) maintains high accuracy across the buffer size, on buffer samples. We nevertheless observe a slight decrease for small buffers, likely explained by the strong imbalance in the training set $\mathcal{D}^t \cup \mathcal{B}$.

Main takeaways

The choice to let the FE trainable depends on the memory budget. For a small memory buffer ($<$25Mb), Latent replay yields higher final accuracy than Native. In this case, Latent offers a better retention but lower adaptation than Native. If obtaining the best performance on the current task at the cost of less retention is preferred, the FE can be trainable, regardless of the buffer size.

3.7 Evaluation on *CORE50* dataset

In this section, we evaluate our approach developed in the previous sections with a more realistic benchmark, the *CORE50* dataset [128]. Recently introduced in the CIL community, *CORE50* serves to benchmark strategies in real-life scenarios. *CORE50* comprises 128×128 pixels images belonging to 50 classes of objects acquired across 11 recording sessions, with 300 consecutive frames (images) per session.

We adopt the scenario of [166] to compare our approach to prior work. It consists in 10 retraining tasks of 5 classes obtained by randomly splitting the dataset. Sessions #3,#7, and #10 represent the test dataset and the others the train dataset. Compared to *CIFAR100*, *CORE50* enables us to extend our conclusions to larger, more complex images, with temporal correlations during acquisition and under various backgrounds.

A deeper architecture, *3Mb-Res-BNN*, is designed to comply with *CORE50* requirements, adhering to the previously stated principles and remaining within the 3Mb model memory size budget. We subsequently evaluate *3Mb-Res-BNN* Latent and Native replay against existing approaches. As there is no pre-training task in the *CORE50* scenario, we propose to pre-train our *3Mb-Res-BNN* model on the *STL10* dataset [44].

This is a well-known benchmark for evaluating unsupervised and semi-supervised learning algorithms. It consists of 10 classes spanning various object categories. The dataset includes 5000 labeled 96×96 color images (500 images/class) and 100000 unlabeled images.

We first detail the new architecture in Section 3.7.1, then we motivate changes made on the pre-training protocole in Section 3.7.2

3.7.1 Deeper feature extractor with skip-connections

3Mb-Res-BNN accommodates larger images thanks to a deeper architecture composed of residual blocks, as depicted in Fig. 3.22. After a first down-scaling with a convolution block of stride 2, we stacked residual blocks that preserve the spacial feature size, and down-sampling blocks that halve the spatial feature size, inspired by ShuffleNetv2 [235]. For the residual block, the main and residual inputs paths are obtained by splitting the feature map in half in the channel dimension. The main path consists in a stack of 1×1, 3×3, and 1×1 convolution blocks with a higher number of groups on the 3×3 convolutions. The down-sampling blocks are similar to the res-blocks but with a stride of on 3×3 convolutions. The residual path contains a 3×3 and 1×1 convolution block. In both blocks, the path aggregation is done by concatenation, as opposed to addition which could lead to bit overflow. After, feature maps are shuffled with a shuffle layer of group 2.

When putting it all together, *3Mb-Res-BNN* results in a 3M parameters architecture with 17 blocks and a 1024 dimensions latent space. With this FE, Latent replay therefore allows to store $80\times$ more examples than Native, at iso-memory.

Res-BNN Feature extractor

Block	Repetitions
ConvBlock (128,3,1,2)	×1
ResBlock (64, 1)	×2
DownSampling (64, 1)	×1
ResBlock (128, 2)	×2
DownSampling (256, 2)	×1
ResBlock (256, 4)	×3
DownSampling (512, 4)	×1
ResBlock (512, 8)	×4
DownSampling (512, 8)	×1
LGAP	×1

Res-BNN Classifier

Block	Repetitions
DenseSkip (128,2)	×1
QuantDense(100,1)	×1
Masking	×1
Normalisation	×1

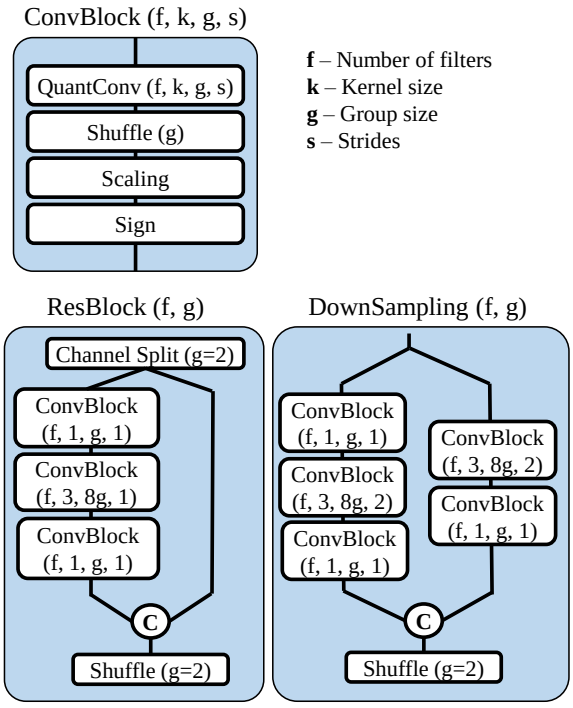


Figure 3.22: *3Mb-Res-BNN* architecture: Deeper FE to comply with *CORE50*.

3.7.2 STL10 pre-training protocole

We propose pre-training the feature extractor on the *STL10* dataset, leveraging its similarly sized images to *CORE50* and its large, diverse unlabeled set, which complements our SSL

loss. We first conduct a preliminary study to evaluate the possibility to do supervised learning with *3Mb-Res-BNN* on *STL10*.

Preliminary study - convergence issues: Table 3.9 presents offline accuracy for the test set and labeled training set. Results indicate low training performance, suggesting potential optimization issues. When replacing scaling factors with Batch Normalization (BN) layers, both training and test accuracy improve.

To investigate why our scaling factor heuristic under-performs compared to BN, we analyzed scaling factor values for each normalization layer (Figure 3.23). Except for the first layer, scaling factor ratios align well. Further, Figure 3.24 shows that our scaling factor values remain within the range of BN scaling factors across layers.

First phase (from random initialization)		
Normalization	**Acc train**	**Acc test**
scaling factor heuristic	51.1%	46.5%
batch normalization	98.0%	77.4%
Second phase (from BN initialization)		
(α, β, γ)	**Acc train**	**Acc test**
$(1, 0, 0)$	98.6%	75.2%
$(1, 10^{-5}, 10^{-2})$	98.0%	77.4%

Table 3.9: Train and Test accuracy results during pre-training on STL10.

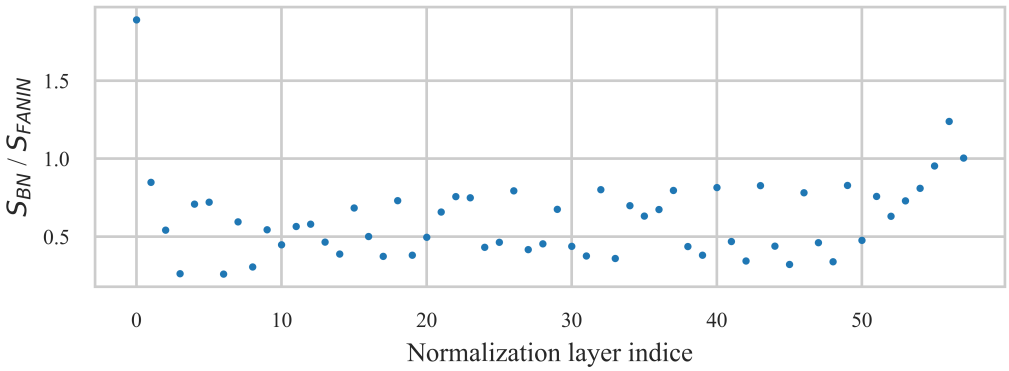


Figure 3.23: Investigation of scaling factor in *3Mb-Res-BNN* architecture.

Two-stages pre-training protocol: As the scaling factor heuristic is not the root issue, we hypothesize that limited training performance arises from weight initialization. Since training with BN provides a robust initialization but introduces data dependency (Section 3.3.2), we propose a two-phase pre-training protocol, benefiting from our proposed multi-objective training loss while keeping architecture-based scaling factors:

1. **Phase 1:** The first phase consists in a standard supervised training on labeled images only. BN layers are then replaced by scaling-factors, as introduced in Section 3.3.2. We divide by 2 the first scaling-factor with respect to observation in Figure 3.23.

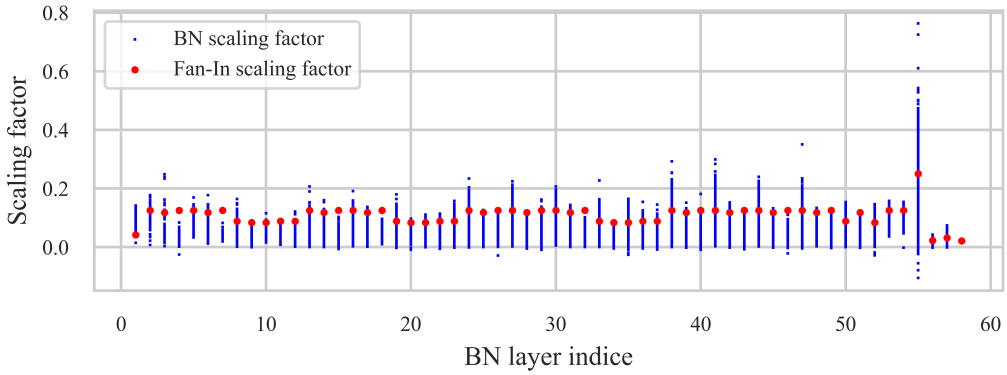


Figure 3.24: Investigation of BN scaling factor distribution in *3Mb-Res-BNN* architecture.

2. **Phase 2:** A second training phase is performed on the entire dataset, benefiting from unlabelled data for the unsupervised training loss terms. $\mathcal{L}_{CCE}$ is computed on labeled images and $\mathcal{L}_{FR}$ and $\mathcal{L}_{SSL}$ on unlabelled images, for regularization. The training hyperparameters are kept the same as in the previous sections.

This two-phase protocol aims to address initialization challenges and improve model performance while adhering to the core principles of our approach. This does not contradict the statements of Section 3.3.2 and does not change the re-training single-phase training. We also validate in Table 3.9 the interest of our multi-objective loss over supervised loss with an absolute 2.2%pts increase in test accuracy.

3.7.3 Experimental results

Method	*CORE50*-TA	*CORE50*-TF
ER [175] (reported from [166])	41.72 ± 1.30	21.80 ± 0.70
ER-Aug	44.16 ± 2.05	25.34 ± 0.74
DER++ [29] (reported from [166])	46.62 ± 0.46	22.84 ± 0.84
DER++-Aug	45.12 ± 0.68	28.10 ± 0.80
CTN [167] (reported from [166])	54.17 ± 0.85	N/A
CTN-Aug	53.40 ± 1.37	N/A
DualNet [166]	57.64 ± 1.36	38.76 ± 1.52
3Mb-Res-BNN Latent replay (ours)	**53.96** ± 1.17	**17.22** ± 0.16
3Mb-Res-BNN Native replay (ours)	**59.07** ± 1.24	**44.52** ± 0.54

Table 3.10: Evaluation metrics on *CORE50* Benchmark. A buffer of 50 samples is used for TA and 100 samples for TF.

The comparison to prior work is rather difficult, as few approaches investigate exactly the same CIL scenario. Nevertheless, we compare our approach to incremental learning approaches with a replay buffer [166]. The baselines are ER [175], DER++ [29], an ER variant with knowledge distillation on logits, CTN [167] a TIL approach with fixed FE and DualNet [166] a CIL method that uses SSL to slowly update the FE. The first three methods

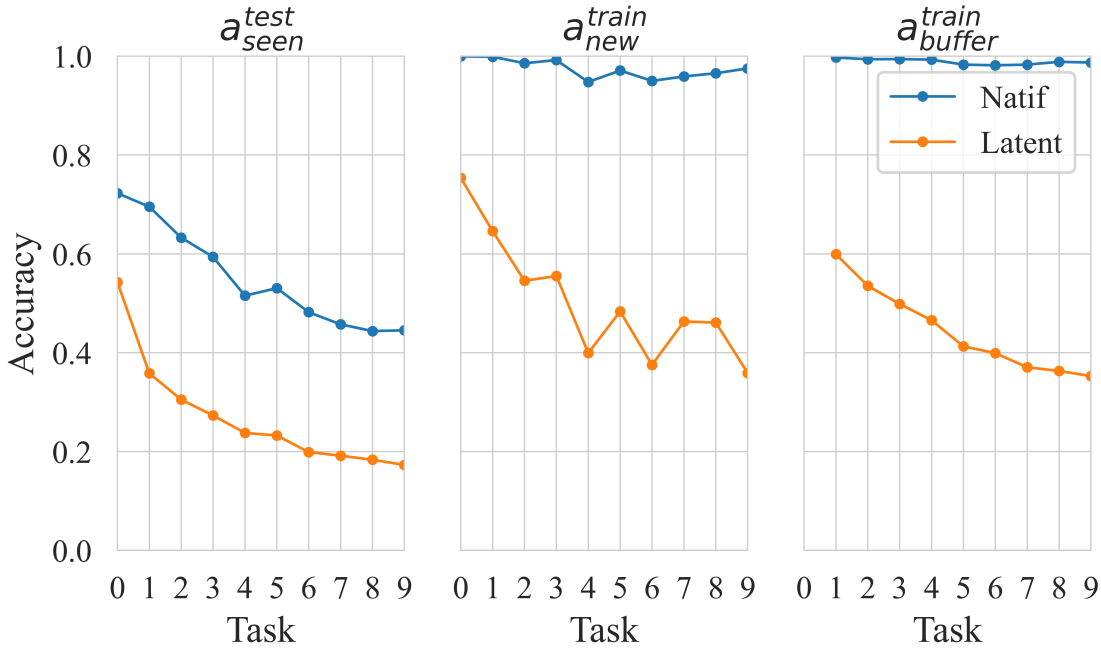


Figure 3.25: Task-Free (TF) CIL performance for *3Mb-Res-BNN* in Native and Latent replay on *seen*,*new* and *buffer* subset. 100 samples per class are stored for Native replay and we use an equivalent buffer size for Latent replay.

are evaluated (Table 3.10) with or without data augmentation (-Aug suffix). The network used in these articles relies on ResNet18 [74] with a 352Mb memory size, *i.e.*, more than $100\times$ larger than *3Mb-Res-BNN*. Unlike our approach, in [166] floatting-point based model CIL is optimized on 3 epochs with a very high learning rate of 0.03. However, *3Mb-Res-BNN* requires smaller learning rate, *i.e.*, 10^{-4}, (as empirically observed, higher learning rates, $> 10^{-3}$, may cause training instability) and hence a larger number of epochs (approximately a hundred) is necessary for convergence.

Table 3.10 reports the final accuracy as proposed in the literature, without task label (Task-Free, TF) and with task label (Task-Aware, TA), corresponding to CIL and TIL, respectively. We report final training accuracy for Native replay with 100 images per class for the TF setting and 50 per class for the TA setting, similarly to other methods in [166]. Likewise, we report accuracy for Latent replay using an equivalent memory buffer size.

In the TF case, *3Mb-Res-BNN* with Latent replay is slightly below the range of performance of the state-of-the-art ER methods without relying on any floating point operation and with $100\times$ less parameters. Nevertheless, in TA setting, *3Mb-Res-BNN* Latent replay outperforms ER, DER and CTN with its 53.96% final accuracy. *3Mb-Res-BNN* with Native replay presents state-of-the-art performance in TF with 44.52%, which is far better than DualNet, despite of using a $100\times$ smaller model memory footprint, a $7.4\times$ smaller buffer footprint thanks to our $TYCC - 16$ image encoding and binary-only arithmetic that tremendously reduces the associated computational complexity.

We further investigate incremental training dynamics in Fig. 3.25 (TF). With Native replay, *3Mb-Res-BNN* succeeds to maintain a high level of adaptation with more than 90% on new task training sets. It also fully captures the knowledge in the buffer with perfect

classification score on buffer samples. On the contrary, with Latent replay, *3Mb-Res-BNN*'s ability to fit the training data (new task and buffer) diminishes through retraining from 70% to 37%. The classifier alone has not the capacity to fit the whole classification problem. However, we can conclude that the FE can still extract meaningful features while having been trained on a completely different dataset. Indeed on the first CIL task, Latent replay is already 12%pts below Native replay while having a fixed FE.

3.8 Conclusion and perspectives

Conclusion about the considered experiments

This study presents an exploration of Fully-Binarized Neural Networks (FBNNs) within the context of Class Incremental Learning (CIL), particularly through the lens of Experience Replay (ER). By addressing critical challenges such as network design, loss balancing, and pre-training, we have demonstrated the viability of FBNNs in dynamic environments where computing resources are constrained. Our findings emphasize the importance of tailored network architectures for BNN-CIL, which are crucial for maintaining expressiveness, mitigating convergence issues and keeping a small memory buffer. The integration of carefully designed loss functions and pre-training strategies enhances adaptability and reduces catastrophic forgetting. Furthermore, our experiments highlight the trade-offs between Native and Latent replay methods, with Latent replay proving to be more effective in memory-constrained settings. Finally, our *CORE50* results shows that even for a challenging benchmark, a careful design of a high-end FBNN model (*3Mb-Res-BNN*) as well as its incremental training protocol, can outperform other CIL frameworks that rely on full-precision network models.

Limitations

One of the main limitation of this work is the bottleneck block design. While it offers more expressiveness compared to a Global Average Pooling layer by learning how to recombine spatial features, it makes the model dependant on input image sizes. We can not do the pre-training task on larger images, like *ImageNet* [50], which could ease richer and more transferable latent representations.

Another limitation of this work lies in the optimization protocol used during the retraining tasks, particularly the choice of the Adam optimizer and batch size. Adam requires storing the first and second-order moment estimates of the gradients, in addition to the gradients themselves, for each parameter. This memory-intensive requirement makes it unsuitable for on-device training on embedded platforms such as MCUs. Lighter optimizers, such as conventional SGD (even with first order momentum [183]), would be more appropriate for these constrained environments [212].

However, S^2BNN [194] highlights the advantages of using Adam over SGD for BNNs, including better weight calibration and improved optimization stability with small learning rates, an essential aspect of our study. It questions the suitability of our heuristics, especially scaling factors, in the context of on-device training. Future work should investigate the influence of our heuristics on more hardware-friendly optimizers like SGD or other

lightweight alternatives and with smaller batch sizes during re-trainings. Small batch size can exacerbate can issues like gradient noise or instability.

Perspectives

A promising approach to improve optimization during the pre-training phase is teacher-student knowledge distillation [217]. The idea is to constrain the activations of a compact network (the student), such as our FBNN, to closely match those of a larger, more expressive network (the teacher), which could be a large BNN or a real-valued CNN (*e.g.*, a ResNet architecture) pre-trained on a large dataset like *ImageNet*.

Guided learning has been shown to significantly benefit BNN training in both supervised [145] and unsupervised settings [194]. This approach leverages the absence of hardware constraints during pre-training, enabling the use of larger networks.

Additionally, it offers a way to address the limitations related to the bottleneck block design and the inability to perform direct pre-training on large datasets. By transferring knowledge from the teacher to the student, knowledge distillation circumvents these challenges without requiring direct training on the original dataset.

Highlights of the chapter

- We demonstrated the viability of Fully Binarized Neural Networks (FBNNs) in Class Incremental Learning (CIL) by addressing network design, loss balancing, and pre-training challenges.
- We highlighted trade-offs between Native and Latent Replay, with Latent Replay proving its superiority in memory-constrained settings.
- We exhibited that a well-designed FBNN model and incremental training protocol can outperform full-precision networks on challenging benchmarks like *CORE50*.

Chapter 4

Pseudo replay with BMM on binary embedding space

Summary

This chapter introduces **Generative Binary Memory (GBM)**, a novel pseudo-replay approach for CIL that generates synthetic binary pseudo-exemplars. GBM leverages **Bernoulli Mixture Models (BMMs)** to effectively capture the multi-modal characteristics of class distributions in a latent binary space. Building upon the latent replay strategies presented in Chapter 3, GBM stores past-distribution statistics rather than random latent vectors, achieving superior performance with a reduced memory footprint for buffer storage. Additionally, thanks to a specifically designed feature binarizer, the approach is compatible with any conventional deep neural network (DNN).
GBM natively supports **Binary Neural Networks (BNNs)**, making it particularly suited for highly constrained model sizes in embedded systems. Experimental results show that GBM surpasses state-of-the-art methods in terms of average accuracy, with improvements of **+2.9% on *CIFAR100*** and **+1.5% on *TinyImageNet*** using a ResNet-18 equipped with the binarizer. Moreover, GBM outperforms emerging CIL methods for BNNs, achieving a **+3.1% gain in final accuracy** and a $4.7\times$ **reduction in memory usage** on CORE50.
Finally, we propose a lightweight implementation of GBM compatible with **online updates** and fixed-point arithmetic, paving the way for hardware deployment. We conclude by discussing the integration of GBM into an **ASIC** for a fully autonomous continual learning pipeline, enabling complete hardware-based continual learners.

4.1 Introduction

The previous chapter (Chapter 3) demonstrated that both latent and native replay are feasible on Fully-Binary Neural Networks (FBNNs) through careful network architecture design and training protocols. However, we observed a critical limitation in latent replay: when the number of stored samples per class is small (e.g., ≤ 100), performance on past tasks degrades sharply. This suggests that random sampling alone is insufficient to maintain robust decision boundaries when only a few exemplars are available.

This raises the question: *Is there a more efficient way to store and represent information about past class distributions?* Specifically, rather than storing randomly selected samples, can we instead capture class statistics using a model that accounts for the binary nature of the embeddings? Such a model could generate synthetic pseudo-exemplars that retain class information. The challenge, then, is to determine under which memory constraints such statistical models can outperform random sampling of binary latent examplars with the latent replay approach.

In the class-incremental learning (CIL) literature, prototype-based methods have addressed similar issues by computing a single prototype per class, typically as the mean of the embedding's distribution. These prototypes are then used to generate pseudo-exemplars via specific heuristics. In Chapter 3, visualizations of the latent embedding revealed a strong class-dependent correlation of features. This observation opens the possibility of computing a representative class prototype by leveraging these correlations. Mathematically speaking, each binary feature can be interpreted as a realization of a Bernoulli process, where the statistical parameter (*i.e.*, the probability of observing a +1) can be empirically estimated as the mean of the observed realizations.

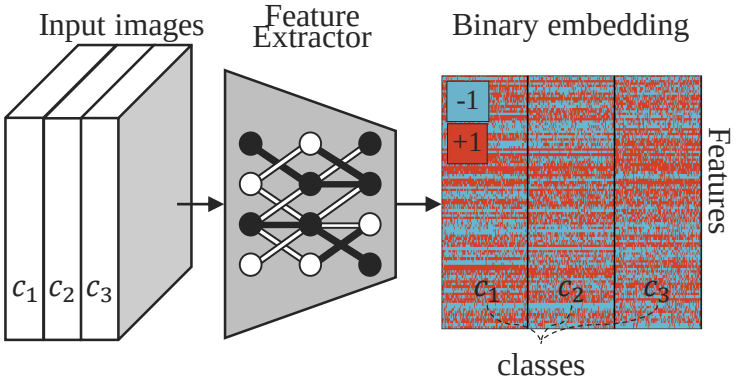
However, a single prototype may be insufficient to capture the full diversity within the latent distribution of a class, as such distributions often exhibit multiple statistical modes, particularly when the feature extractor is fixed. This observation motivates the use of a more expressive statistical model, such as a Bernoulli Mixture Model (BMM), which generalizes the single-prototype representation to a multi-prototype framework.

To illustrate this point, Figure 4.1 presents an analysis of binary embeddings. Figure 4.1a) illustrates the latent activations generated by our *3Mb-BNN* FE for the training sets of classes $\#50$, $\#51$, and $\#52$ from *CIFAR100*. The FE was pretrained on the first 50 classes of *CIFAR100* following the training protocol described in Section 3.5. We observe that binary features tend to cluster by class, with some features showing higher flip rates. The $+1$'s frequency for each feature may constitute a representative prototype of a class' embedding distribution. Thus the class distributions can be generated by realization of the Bernoulli processes as in Figure 4.1b).

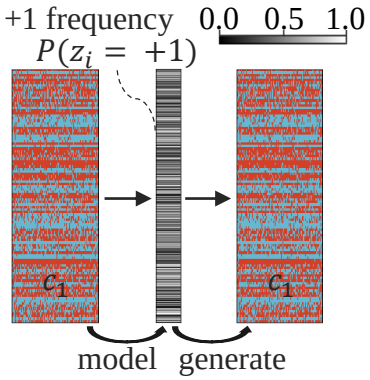
Moreover, we observed that multiple modes appear when applying hierarchical clustering [155] to class embedding (Figure 4.1c)), suggesting that a single prototype may not capture the full diversity of a class. To address this, the Bernoulli Mixture Model (BMM) offers a statistical framework to model binary vectors with multi-modal characteristics.

The observations presented here open several scientific opportunities. First, to the best of our knowledge, no pseudo-replay methods have been specifically designed for BNNs. *Can BMMs provide a robust and efficient replay mechanism tailored for BNNs?*.

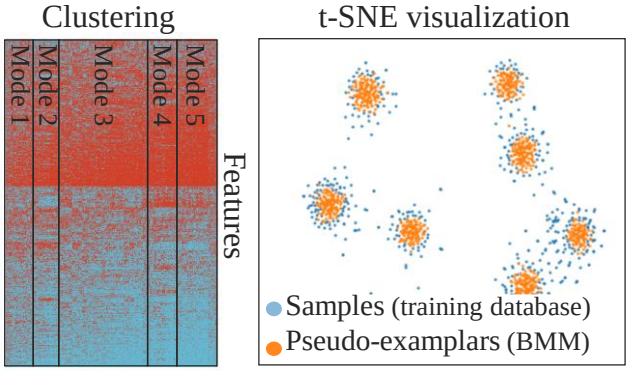
Secondly, BMMs naturally support multi-prototype representations without requiring



a) Latent space as binarized embeddings



b) Prototypes as Bernoulli process



c) c_1 class samples as Bernoulli Mixture Model

Figure 4.1: Motivations of our pseudo-replay CIL approach based on Bernoulli Mixture Model (BMM). a) Illustration of per-class features correlation on binary embedding. b) Illustration of a single prototype encoding and pseudo-examplars generation. Prototype features is computed as the +1's frequency in a class embedding. Pseudo-examplars are generated from the prototype as a Bernoulli process. c) Qualitative investigation of the binary distribution: (left) apparition of modes when reorganising samples and features with hierarchical clustering, (right) t-SNE visualization [134] of training samples and pseudo-exemplars generated from a BMM.

memory-intensive second-moment statistics, such as covariance matrices used in Gaussian Mixture Models (*e.g.*, [241]). By Combining BMMs with specifically designed binary quantizers enables building a multi-prototype pseudo-replay approach compatible with any DNN. This approach can then be evaluated against state-of-the-art methods in CIL.

Thirdly, the memory cost of storing prototypes poses an important trade-off. Each feature in the prototype represents a continuous probability value, which requires q bits for storage. This storage is equivalent to saving q latent binary samples per feature in a latent replay setting. This observation raises the question of optimal memory buffer usage: under which conditions does storing statistical prototypes achieve better performance than storing randomly selected latent embeddings? *For a given memory budget, when is it preferable to store statistical prototypes rather than latent binary exemplars?*

Finally, the Expectation-Maximization (EM) algorithm [25], commonly used to optimize mixture models, can be adapted for constrained applications. Variants exist that support online updates or a reduced arithmetic precision, making them suitable for hardware-friendly implementations. These adaptations provide an opportunity for deploying BMM-based methods on resource-limited platforms.

Building on these opportunities, this chapter brings the following contributions:

- Generative Binary Memory (GBM): A pseudo-replay continual incremental learning (CIL) method operating on binarized embeddings, built upon Binary Mixture Models (BMMs).
- Embedding binarizers: Custom-designed binarization approaches that make GBM compatible with any feature extractor and deep neural network (DNN).
- Improved state-of-the-art performance: Experimental results on ResNet-18 [74] show GBM outperforms other prototype-based methods, achieving a $+1.5\%$ improvement over FeTril [164] on the challenging *TinyImageNet* dataset with 20 incremental tasks.
- Efficient CIL with BNNs: Experimental validation of GBM with binary neural networks (BNNs) demonstrates superior performance under strict memory constraints. On the CORE50 benchmark, GBM achieves $+3.1\%$ higher final accuracy and a $\times 4.7$ memory reduction compared to recent methods [17, 214].
- Hardware-friendly online EM algorithm: An online fixed-point implementation of the EM algorithm, enabling GBM deployment on always-on, on-demand hardware platforms. While still in its early stages, this approach shows promising results, paving the way for real-world applications.

This chapter begins with a review of prototype-based methods and the use of Mixture Models in DNNs. We then introduce our GBM method for continual incremental learning (CIL) scenarios, presenting two embedding binarization approaches that generalize the method to any DNN architecture.

We then investigate the method's behavior using a ResNet-18 feature extractor to test hyperparameter choices and benchmark it against state-of-the-art methods. Next, we validate our approach on hybrid BNNs—networks that retain some full-precision layers—by comparing it to the latent replay method LR+CWR* [214] under varying memory buffer

sizes. Finally, we conduct iso-memory comparisons on our compact *3Mb-BNN*, assessing the benefits of prototype quantization for reducing memory while preserving performance.

Finally, we propose a hardware-friendly implementation preliminary insights of GBM using an online fixed-point EM algorithm, that would ease a future deployment on ASIC platforms for real-time, memory-efficient online CIL settings.

4.2 Related work

Prototype-based method

Among the continual learning problems in the literature [218], our focus is on Class Incremental Learning, CIL [179], whose goal is to learn a unified classifier without relying on task labels. CIL inherently involves a plasticity-stability trade-off, where the model must remain flexible to learn new tasks and stable enough to preserve previously acquired knowledge. Various methods exist, including architectural-based approaches [226], regularization-based techniques like Elastic Weight Consolidation (EWC) [99] and Learning without Forgetting (LwF) [120], and replay-based strategies [179]. Replay methods, such as iCaRL [179], EEIL [35], and LUCIR [80], are effective but introduce the additional memory overhead related to past tasks exemplars. LR methods [161] address the memory issues, storing latent features instead of raw data.

Furthermore, prototype-based methods rely on class centroids and distributions modeling to generate synthetic data. For instance, PASS [242] uses isotropic augmentation based on the trace of the covariance matrix, while IL2A [241] accounts for feature interdependencies estimations using a multivariate Gaussian approximation. More recently, FeTril [164] augments the distribution of old classes based on new classes statistics. Knowledge distillation [66] can also cap drift from previously learned distributions [241, 242].

Like FeTril, we address a system with a fixed feature extractor to ensure a stable representations, while retraining only the classifier. Our proposed method further differentiates from other prototype-based approaches by (1) working on categorical binary features rather than continuous (2) handling the multi-modality of class distributions using a mixture model, and (3) applying prototype quantization.

Mixture model in DNNs

BMMs have been widely studied to model multidimensional categorical data [25, 94, 119, 109], with applications in clustering [152], classification [4, 63], and dimensionality reduction [186]. More recently, BMMs are combined with deep neural networks, particularly for parameterizing priors in variational auto-encoders [222, 127], image retrieval [11, 10] and multi-instance learning [229]. BMMs have not been yet applied to CIL, unlike Gaussian Mixture Models (GMMs) [25]. For example, Gaussian Mixture Replay [165] models and generates input images using GMMs, however it limits to simple datasets such as MNIST [113]. Furthermore, MIX [100] learns GMMs and uses them as classifiers, but performs well only when the model is pre-trained on the whole dataset.

Hybrid- and fully-binary neural network in CIL

BNNs have binary weights and activations (*e.g.*, $\{-1, +1\}$ [232]) and thus all scalar products are replaced with more efficient XNOR logic gates and bit-count operations [85]. As a result, BNNs drastically reduce memory footprint, and energy consumption, making them highly efficient in terms of hardware implementation. Despite their non-differentiable nature, BNNs training through back-propagation is made possible by the Straight-Through Estimator (STE) [20] in a Quantization-Aware Training (QAT) approach [105]. Nevertheless, BNNs' performance does not yet match the one of the corresponding conventional, real-valued networks. A way to narrow this gap is to keep some layers in floating-point precision [16, 124, 126], leading to **hybrid BNNs**. Another branch of research focuses on optimizing BNNs that entirely rely on binary arithmetic [132, 17], *i.e.*, **Fully BNNs**, to enable disruptive hardware implementations.

Recent work also explores CIL for BNNs. Synaptic metaplasticity [107] introduces a regularization function that computes weights' importance based on weights' magnitude during QAT, however is limited to simple datasets. LR on binary embedding is proposed in [214], taking advantage of BNNs' compression. Finally, [17] explores replay mechanisms in fully-binary networks, showing the suitability of both experience and LR for ultra-low power devices. This work goes beyond replay methods by generating binary pseudo-exemplars based on intra-class correlations, to increase performance and further reduce CIL memory needs.

4.3 Methods

4.3.1 CIL problem re-formulation

The CIL problem involves learning a sequence of classification tasks, starting from an initial task, with training dataset $\mathcal{D}_0$, followed by T incremental tasks, with training datasets $\mathcal{D}_1$,...,$\mathcal{D}_T$. $\mathcal{D}_t = \{(\boldsymbol{x_{t,i}}, y_{t,i})\}_{i=1}^{N_t}$ is the training dataset that the model receives at task t. $\mathcal{D}_t$ consists in N_t labeled samples, $\boldsymbol{x_{t,i}}$, $y_{t,i} \in \mathcal{C}_t$ denotes the class labels, and $\mathcal{C}_t$ is the set of classes for task t. There is no intersection between the classes of different tasks, meaning $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset$ for $i \neq j$. Typically, at task t, the model is updated only on the training examples from the current task, $\mathcal{D}_t$ and evaluated on the classification problem containing all seen classes $\bigcup_{i=1}^{t} \mathcal{C}_i$.

4.3.2 Generative binary memory

System overview

We consider a neural network composed of a feature extractor $\mathcal{F}$ and a classifier $\mathcal{G}$, extended with our Generative Binary Memory (GBM), as depicted in Figure 4.2. The GBM incorporates a prototype memory buffer that handles prototype storage and pseudo-exemplar generation through two algorithmic blocks:

1. **Updater algorithm block**: When new classes appear, the buffer is updated using this block, which employs the Expectation-Maximization (EM) algorithm. The EM

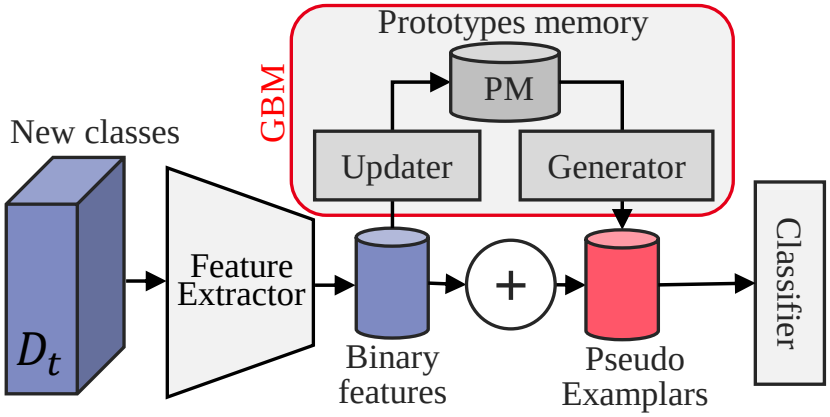


Figure 4.2: GBM system overview. A model feature extractor and classifier are augmented with a prototype memory buffer. The buffer is updated with BMM parameters computed from the updater block and generates pseudo-examplars from the generator block.

algorithm, an iterative optimization method, estimates the parameters of a Binary Mixture Model (BMM) from the training data.

2. **Generator algorithm block**: This block generates synthetic pseudo-exemplars using class-specific statistics encapsulated in the prototypes.

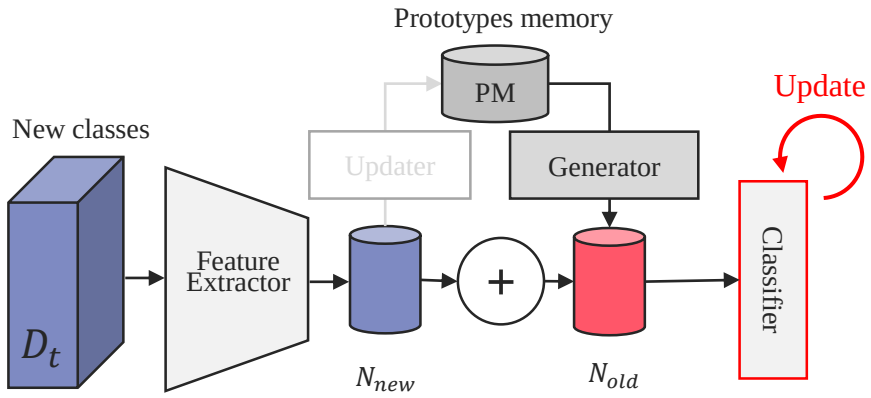
$\mathcal{F}$ is trained during the initial task and then frozen. For each incremental task, the system is updated in two steps. First, $\mathcal{G}$ is re-trained on a mixture of new training samples and generated pseudo-exemplars representing old classes (Figure 4.3). Second, the GBM is updated with prototypes computed from the new classes' training set. The prototypes and their mixing coefficients are estimated per class as a BMM using the EM algorithm.

In what follows, we detail how prototypes are computed with an EM algorithm, given the binary features. The next section describes the prototype memory update and the process of data generation from prototypes. Finally, while our method is motivated by CIL on BNNs, we extend its applicability to any neural network; for this two embedding binarizer blocks are proposed.

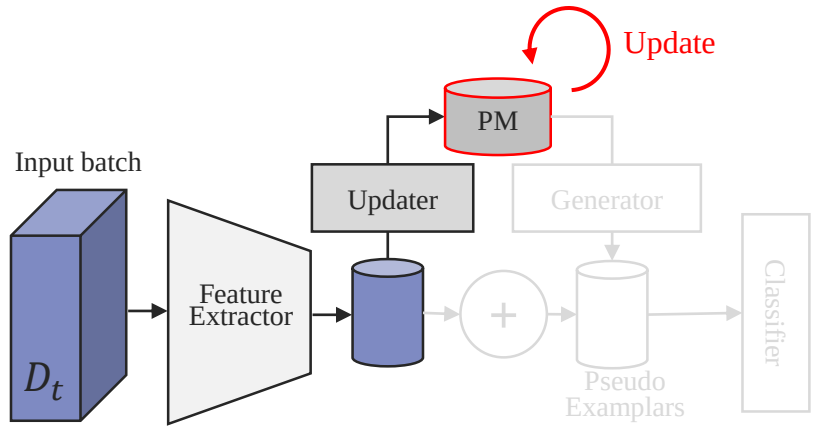
Multi-prototype encoding with BMM

At a given task t training phase, the GBM updater computes a multi-prototype representation of the N training samples, $\boldsymbol{X} = \{x_{t,i} \mid y_{t,i} = c\}$ of a given class $c \in C_t$. Let $\boldsymbol{Z} = \mathcal{F}(\boldsymbol{X})$ be the N latent binary embedding set with dimension $D \in \mathbb{N}$, *i.e.*, $\boldsymbol{Z} = \{\boldsymbol{z_i}\}_{i=1}^{N} \in \{0,1\}^{N \times D}$. Under the assumption that $\boldsymbol{Z}$ is generated by $K \in \mathbb{N}$ underlying Bernoulli processes, we model the distribution with a BMM parameterized by $\Phi = (\boldsymbol{\mu}, \boldsymbol{\pi})$, where $\boldsymbol{\mu} = \{\boldsymbol{\mu_k}\}_{i=1}^{K} \in [0,1]^{K \times D}$ denotes the K Bernoulli probability vectors, namely prototypes, and $\boldsymbol{\pi} = \{\pi_k\}_{i=1}^{K} \in [0,1]^{K \times 1}$ denotes the corresponding mixing coefficients, subject to $\sum_{k=1}^{K} \pi_k = 1$.

The EM algorithm [25] estimates the parameters Φ from the set of observations $\boldsymbol{Z}$. As detailed in illustration pseudo-code 2, the EM algorithm iteratively refines Φ through two



Step 1: Classifier update (per mini batch)



Step 2: Prototype Memory update (per task dataset)

Figure 4.3: Network-Memory sytem update in two-steps.

main steps: (1) an expectation step (E-step), and (2) a maximization step (M-step). In the E-step (Equation 4.1), the responsibilities $\gamma_{i,k}$, *i.e.*, the probability that a binary embedding $\boldsymbol{z_i}$ was generated by component k, are defined as:

$$\gamma_{i,k} = \frac{\pi_k \prod_{j=1}^{D} \mu_{k,j}^{z_{i,j}} \left(1 - \mu_{k,j}\right)^{1-z_{i,j}}}{\sum_{l=1}^{K} \pi_l \prod_{j=1}^{D} \mu_{l,j}^{z_{i,j}} \left(1 - \mu_{l,j}\right)^{1-z_{i,j}}} \tag{4.1}$$

In the M-step (Equation 4.2), the model parameters $\boldsymbol{\mu}$ and $\boldsymbol{\pi}$ are updated based on the computed responsibilities:

$$\pi_k = \frac{1}{N} \sum_{i=1}^{N} \gamma_{i,k} \qquad\qquad \mu_{k,j} = \frac{\sum_{i=1}^{N} \gamma_{i,k} z_{i,j}}{\sum_{i=1}^{N} \gamma_{i,k}} \tag{4.2}$$

Algorithm 2 EM Algorithm for BMM.

Require: Latent binary embeddings $\boldsymbol{Z}$, number of components K, threshold ϵ
Ensure: Parameters $\Phi = (\boldsymbol{\mu}, \boldsymbol{\pi})$
Initialize $\boldsymbol{\mu}$ and $\boldsymbol{\pi}$ around the centroid.
repeat
    E-step:
    for each embedding $\boldsymbol{z}_i$ and component k **do**
        Compute responsibilities $\gamma_{i,k}$ using Equation 4.1.
    M-step:
    for each component k **do**
        Update mixing coefficients π_k using Equation 4.2.
        Update prototypes parameters $\mu_{k,j}$ using Equation 4.2.
    Compute log-likelihood $\ell(\Phi^{(s)})$ using Equation 4.3.
    Check convergence: If relative change in log-likelihood $< \epsilon$, stop.
until Convergence
if prototype quantization is required **then**
    Quantize uniformly $\boldsymbol{\mu}$ to q-bit unsigned integers.
return Parameters Φ

The BMM log-likelihood at iteration s of the EM-algo $\ell(\Phi^{(s)})$ is further defined in Equation 4.3 as:

$$\ell(\Phi^{(s)}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_{i,k} \log \left(\pi_k \prod_{j=1}^{D} \mu_{k,j}^{z_{i,j}} \left(1 - \mu_{k,j}\right)^{1-z_{i,j}} \right) \tag{4.3}$$

We choose to stop the algorithm when the relative change $|\ell(\Phi^{(s)}) - \ell(\Phi^{(s-1)})| / |\ell(\Phi^{(s)})|$, falls below a predefined threshold ϵ.

Initialization. Initialization is critical for the EM algorithm to effectively converge and prevent "pathological cases" [94] such as prototype degeneration, *i.e.*, when a prototype overfits on one single training point. We therefore initialize each prototype parameter $\boldsymbol{\mu}$ by calculating the centroid of $\boldsymbol{Z}$ and then perturbing each μ_k around this centroid with a per-feature standard deviation. The mixing coefficients $\boldsymbol{\pi}$ are fixed to $\pi_k = \frac{1}{K}$, making it not trainable, unlike in Equation 4.3. This prevents prototype degeneration by encouraging a balanced prototype attribution during the E-step. Section 4.4 includes an experimental investigation of these BMM initialization choices. To further improve the initialization, we perform N_{init} warm-up rounds with different initializations, running the EM algorithm for N_{iter} iterations each. The initialization that yields the highest log-likelihood is selected, and then fully optimized up to the stopping criterion (the L1 relative change of the log-likelihood).

Post-EM prototype quantization. Although prototype-based methods are often described as memory-free [242], storing prototypes still incurs memory costs, as highlighted in [164]. To improve efficiency over conventional methods that store latent binary samples, GBM prototypes can be uniformly quantized to q bits, reducing their memory footprint from the standard 32-bit floating-point precision. In Section 4.5.2, we analyze the impact

of post-EM quantization on incremental classification performance, exploring quantization levels that reduce prototype memory usage to match that of a small set of binary exemplars. We then compare performance under "iso-memory" conditions against latent replay methods.

Prototype Memory Update

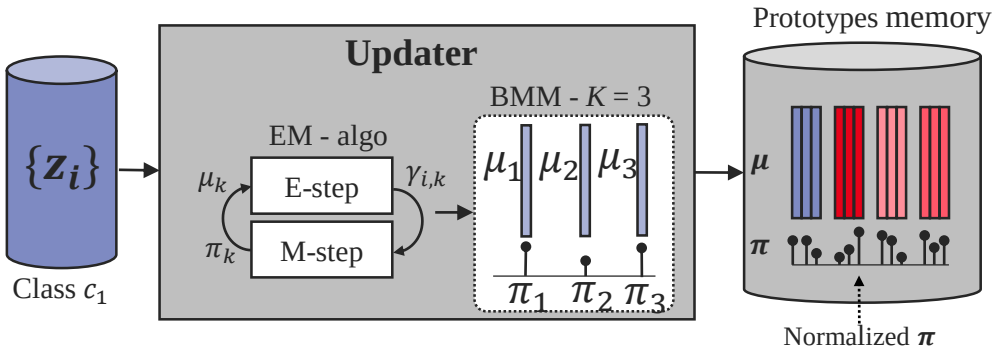


Figure 4.4: The Updater algorithm with K=3 prototypes. Prototypes are computed by class with an EM algorithm and added to the prototype memory buffer.

A BMM, Φ_{new}=($\boldsymbol{\mu_{new}}$, $\boldsymbol{\pi_{new}}$), is computed for each new encountered class. The PM is updated by appending the newly computed prototypes $\boldsymbol{\mu_{new}}$ to the existing set of prototypes $\boldsymbol{\mu}$, and the mixing coefficients $\boldsymbol{\pi_{new}}$ to $\boldsymbol{\pi}$. To ensure that the concatenated $\boldsymbol{\pi}$ accurately represents the drawing probability across all seen classes, $\boldsymbol{\pi}$ is normalized based on the effective number of training samples per class.

Pseudo-examplars generation

During the classifier update on a new class, GBM generates pseudo-exemplars at each training batch. A pseudo-exemplar $(\boldsymbol{z_{pe}}, \boldsymbol{y_{pe}})$ is generated by first selecting a prototype $\boldsymbol{\mu_i}$ in $\boldsymbol{\mu}$ with selection probability $\boldsymbol{\pi}$. Then a realization is drawn from the Bernoulli process parameterized by $\boldsymbol{\mu_i}$ (Figure 4.5).

$$z_{pe} \sim \mathcal{B}(\mu_i) \text{ where } i \sim X \text{ and } P(X = i) = \pi_i \tag{4.4}$$

Balanced Training Batch

Within the fixed training batch size B, the number of new training data N_{new} and pseudo-exemplars N_{old} is set proportionally to the number of new n_{new} and previous classes n_{old}. This ensures equal representation of all seen classes in batches (Equation 4.5), circumventing possible dataset imbalance. This batch filling procedure is equivalent to the loss weighting method described in Section 3.4, eliminating the need for additional loss balancing.

$$N_{new} = \frac{B \times n_{new}}{n_{old} + n_{new}} \qquad N_{old} = \frac{B \times n_{old}}{n_{old} + n_{new}} \tag{4.5}$$

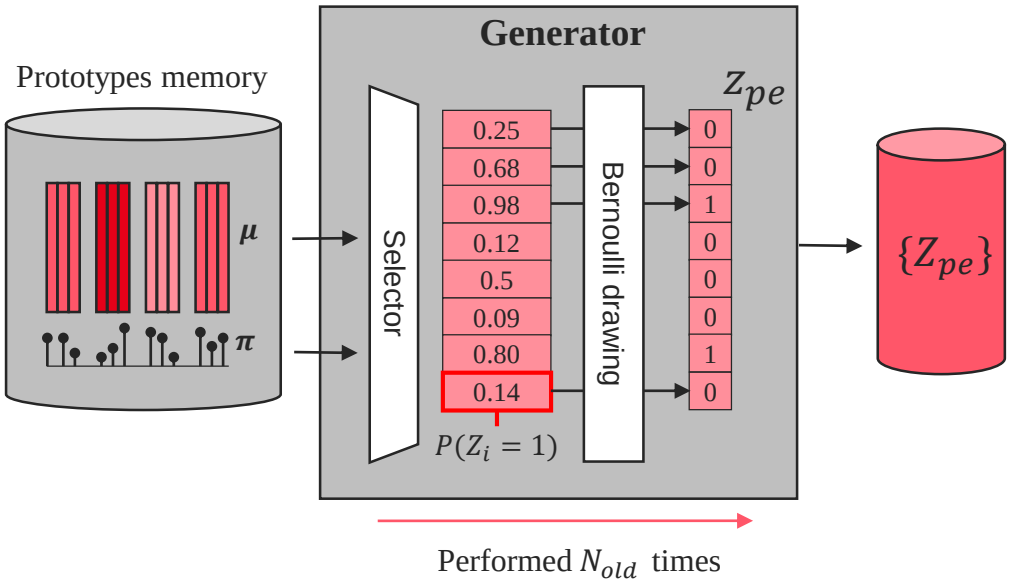


Figure 4.5: The Generator algorithm. Prototype are selected based on their mixing coefficient and Pseudo-examplars are obtained as realization of the associated Bernoulli process.

4.3.3 Embedding binarization

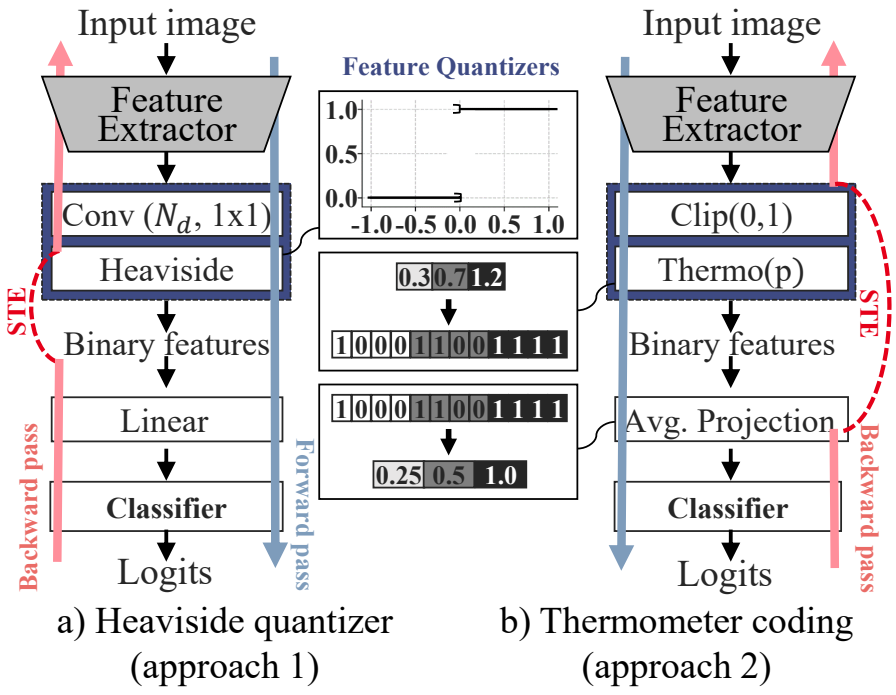


Figure 4.6: The two proposed approaches for embedding binarization.

GBM can be included in non-binary, conventional neural networks by adding an embedding binarizer and slightly adapting the training protocol of the initial task (Figure 4.6).

Two approaches are possible: (1) enforcing $\mathcal{F}$ to output directly a binary vector on its last layer; (2) coding learned full precision features into a quantized value shaped as a vector in a thermometric (binary) representation.

Heaviside binarizer. For the first approach we propose to extend $\mathcal{F}$ with a point-wise convolution with $f \times D$ filters, followed by an Heaviside step activation for binary quantization [154]. The point-wise convolution goal is to counterbalance the loss in expressiveness due to binarization. We train the model using the Straight-Through Estimator (STE) [20] to compute the gradients in the Heaviside's backward computation, to learn a binary feature representation. The binary embedding size is controlled by a factor f, as investigated in Section 4.4.

Thermometer binarizer. For the second approach, a thermometer representation [27] encodes the real-valued D-dimensional embedding. A thermometer code represents an integer number on p bits, where the number of consecutive ones corresponds to the encoded value. Assuming that features values are within $[0, 1]$, we can uniformly quantize the interval $[0, 1]$ on p values and concatenate the associated thermocodes in one unique $p \times D$-dimensional binary vector. Equation 4.6 describes the thermocode transformation $\hat{\boldsymbol{z}}$ of a embedding vector $\boldsymbol{z}$. z_i refers to the i-th feature.

$$\hat{z}_{p\times i+j} = \begin{cases} 1 & \text{if } j \times \lfloor clip(z_i) \times p \rfloor \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

Features need to be converted back in the real-valued domain, before being fed to the classifier. This conversion is done by averaging each thermocode segment (Equation 4.7). It is worth noticing that that reconstructed features undergo quantization errors.

$$z_i = \frac{1}{p} \sum_{j=0}^{p-1} \hat{z}_{p\times i+j} \tag{4.7}$$

Since feature values may not be between 0 and 1, the output of $\mathcal{F}$ needs to be clipped in $[0, 1]$. However, to avoid too much activation saturation, a second training phase is performed with an STE from $\mathcal{F}$'s outputs to $\mathcal{G}$'s inputs. This way, $\mathcal{F}$ can learn to adjust its output dynamic range to the saturation and thermometer coding.

4.4 Evaluation on a ResNet architecture

In this section, we evaluate the approach on a conventional ResNet, enhanced by our proposed embedding binarizers. This allows us to compare with state-of-the-art methods in CIL, assess the generalizability of our technique, and test its robustness across various hyper-parameters. Specifically, ResNet-18 [74] is used as $\mathcal{F}$, adapted from the implementation provided in [92], and trained from scratch. The network consists of a stack of residual blocks organized into four stages, with the number of filters doubling at each stage (64, 128, 256, and 512). Each residual block applies two 3×3 convolutions, with batch normalization and ReLU activation, followed by an identity skip connection. Downsampling is performed using a stride of 2 at the first block of each stage, starting from the second stage.

The feature extractor $\mathcal{F}$ terminates at the output of the average-pooling layer, yielding an output dimension of $D = 512$, with a total of approximately 11.2 million parameters. The classifier $\mathcal{G}$ consists of a linear projection as generally proposed in the literature [160].

4.4.1 Evaluation set-up

Datasets and incremental setting. GBM is evaluated on two common datasets in CIL: *CIFAR100* [103] with 100 classes and *TinyImageNet* [111] with 200 classes, using a fixed random order for classes. Typically, the initial task dataset $\mathcal{D}_0$ consists of half of the classes and the other half of the classes are divided across the incremental tasks. We study CIL in three settings, with T=5, 10, and 20 classes. For T=20 on *CIFAR100*, the initial task includes 40 classes instead of the usual 50.

Compared Methods. We compare GBM with the two proposed binarizers, *p-bits thermometer*, GBM_p^T, and *f-heaviside*, GBM_f^H, to examplars-free CIL methods: LwF-MC [120], EWC [99] which are memory-free and PASS [242], IL2A [241], FeTRIL [164] which store class-prototypes. We also compare to exemplars-based method iCaRL-CNN [179], EEIL [35] and LUCIR [80]. Note that our method bridges the gap between exemplars and non-exemplars as we can control the number of prototype stored by classes.

Metrics. We report the average accuracy. Average accuracy is the average of the $T + 1$ task accuracy (test set) on all the classes that have already been learned so far. We also consider final train and test accuracy to better understand the influence of embedding binarizer blocks on incremental performance.

Training and hyper-parameters. Following the training protocol in [164], we train ResNet-18 from scratch using a batch size of 128 and the SGD optimizer with a momentum of 0.9. The initial learning rate is 0.1, exponentially decaying by a factor of 0.1 every 50 epochs, for a total of 160 epochs. Data augmentation includes random horizontal flips, 4-pixel translations, and random contrast adjustments. Unlike Chapter 3, where we carefully selected hyper-parameters, particularly the learning rate scheduler, here we follow the standard training protocol commonly used in the literature. This approach ensures a fair comparison under consistent conditions. For BMM computation, we use N_{init}=5 and N_{iter}=3 for warm-up, with ϵ=10^{-3} and a maximum of n_{max}=10 steps for stopping criteria. Results are averaged over 3 runs.

4.4.2 Results and discussion

Comparison with the State-of-the-art

Table 4.1 reports the experimental results on *CIFAR100* and *TinyImageNet* for the three incremental settings. The results indicate that our 8-prototypes GBM_1^T has consistently better performance than state-of-the-art, on nearly every configuration. Compared to the second-best method, FeTril, GBM's accuracy is $+1.5\%$ higher, on the most challenging case of *TinyImageNet* with T=20. Moreover our method maintains high final performance when

Category	Method	CIFAR-100			*TinyImageNet*		
	T=	5	10	20	5	10	20
Replay	iCaRL	51.1	48.7	44.4	34.6	31.2	27.9
E=20	EEIL	60.4	56.1	52.3	47.1	45.0	40.5
	LUCIR	63.8	62.4	59.1	49.1	48.5	42.8
Regularization	LwF-MC	45.9	27.4	20.1	29.1	23.1	17.4
K=0	EWC	24.5	21.2	15.9	18.8	15.8	12.4
Prototype	PASS	63.5	61.8	58.1	49.6	47.3	42.1
K=1	IL2A	66.0	60.3	57.9	47.3	44.7	40.0
	FeTril	<u>66.3</u>	<u>65.2</u>	61.5	**54.8**	53.1	52.2
	GBM_2^H	64.5	64.3	61.1	53.4	52.7	52.2
	GBM_1^T	66.0	65.8	64.3	53.9	53.2	52.8
Multiprototype	GBM_2^H	65.2	65.0	<u>61.7</u>	54.0	<u>53.3</u>	<u>52.9</u>
K=8	GBM_1^T	**67.1**	**67.0**	**64.4**	<u>54.6</u>	**54.0**	**53.7**

Table 4.1: GBM average incremental accuracy compared to replay methods (E=20 exemplars per class), memory-free regularization methods and prototype methods. Best GBM^T and GBM^H configurations from Table 4.2 are reported. **Best results**, <u>second best results</u>.

the number of tasks increases, with a relative decrease of 1.6% on *TinyImageNet* between T=5 and T=20, compared to a relative 4.7% for FeTril.

Furthermore our 1-prototype GBM_1^T exhibits the best performance among single prototype approaches on T=10 and T=20. For T=5, GBM_1^T remained at -0.9% of SOTA FeTril on *TinyImageNet* while having features precision on 1-bit compared 32-bits floating-point. Finally, Figure 4.7 shows that the 8-prototype GBM_1^T outperforms other methods on every task.

Sensitivity to quantization

D	p or f	method	Task Init (0)		Task Final (10)	
			Train	Test	Train	Test
512	1	GBM^T	100.0	81.1	76.1	57.9
		GBM^H	100.0	81.1	73.5	54.9
1024	2	GBM^T	100.0	81.3	75.1	57.7
		GBM^H	100.0	81.0	73.6	54.6
2048	4	GBM^T	100.0	81.5	74.5	56.7
		GBM^H	100.0	80.7	73.6	54.3

Table 4.2: Investigation on the binarization method applied to Resnet-18. Results are on *CIFAR100*, T=10, K=8 and for both Heaviside (GBM^H) and Thermometer (GBM^T) binarizations.

Table 4.2 reports train and test accuracy on the initial task and the final incremental task of *CIFAR100*, T=10 for different sizes of binary embedding, with K=8 prototypes. For GBM^T, increasing the number of bits marginally improves test performance of the initial task by making $\mathcal{F}$ more expressive. On the contrary, surprisingly, a higher precision results

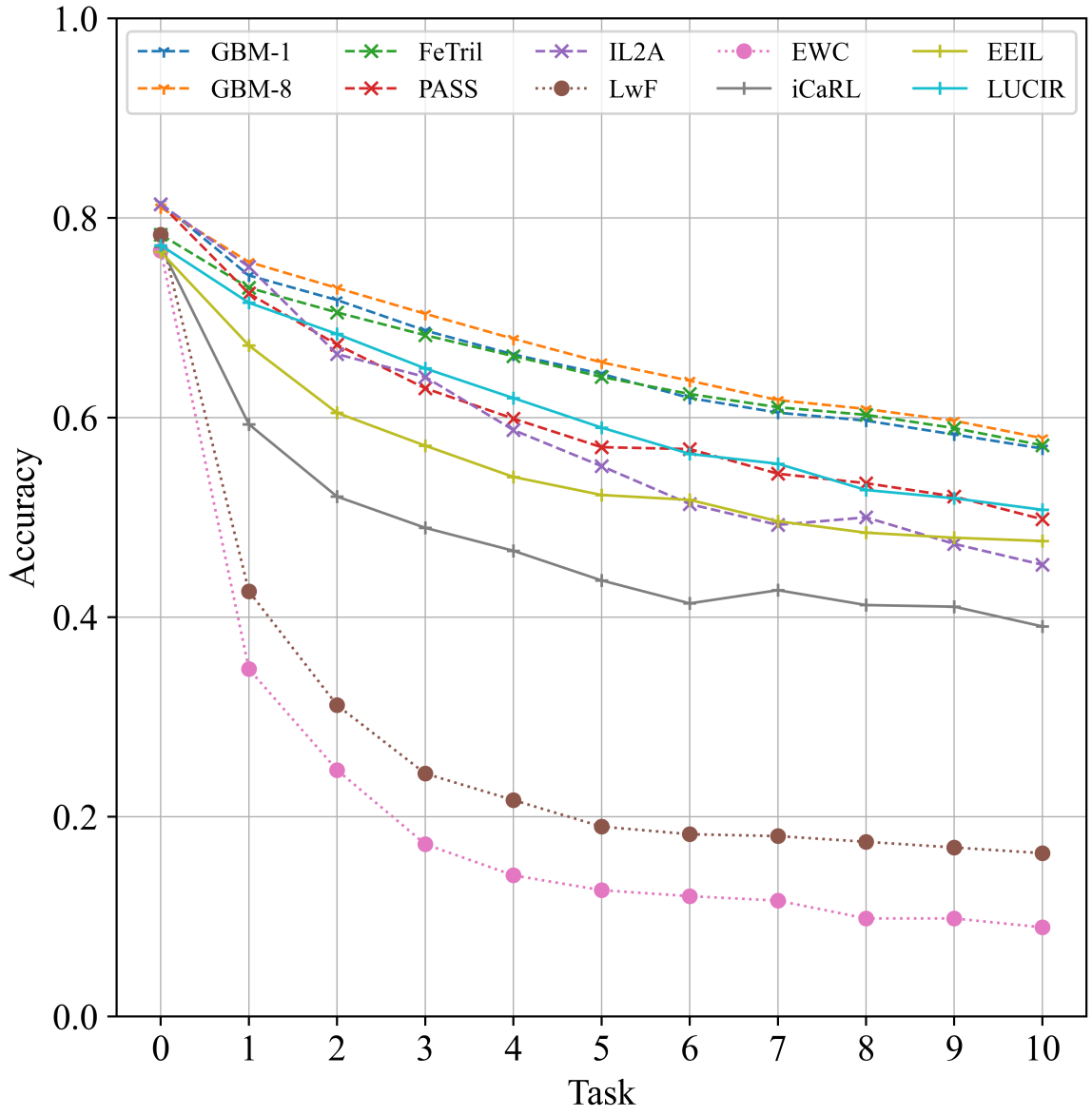


Figure 4.7: Comparison of GBM_1^T and GBM_8^T against state-of-the-art on *CIFAR100* T=10.

in lower train and test accuracy on the final task. One reason can be that a lower-precision embedding regularizes the training and offers more transferable features to future incremental tasks, thus easing the optimization (train set) and enhancing generalization (test set). For GBM^H, the initial and final performance is not affected by the embedding size, with less than 1% difference in the performance. We report results for GBM_2^H, GBM_1^T with K=1 and K=8 on *CIFAR100* and *TinyImageNet* benchmark Table 4.1. GBM^T always yields better performance than GBM^H. Adding a projection to learn binary features is not helpful. Figure 4.8b) presents the training curves during all incremental retrainings for GBM_1^T, K=8, our best configuration. Train accuracy, computed on training batches (samples and pseudo-examplars), nearly always reaches 100%. This suggests that the embedding binarizer does not prevent $\mathcal{F}$ and $\mathcal{G}$ from being correctly optimized despite quantization.

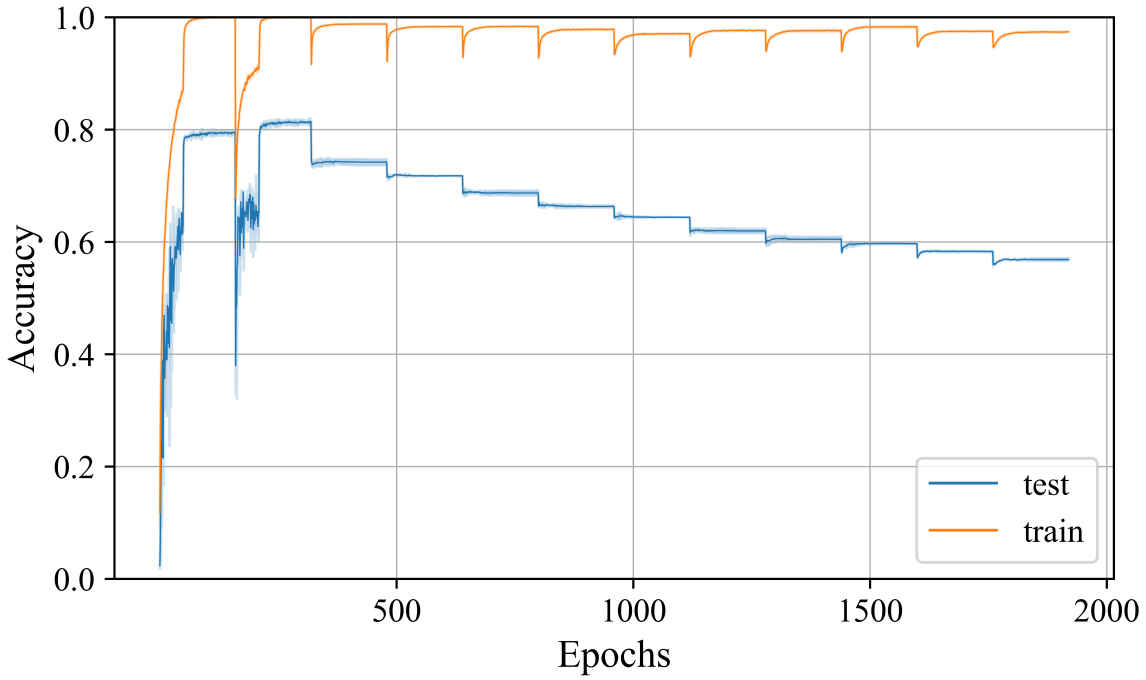


Figure 4.8: Training curves on test and training sets for GBM_8^T on *CIFAR100* T=10.

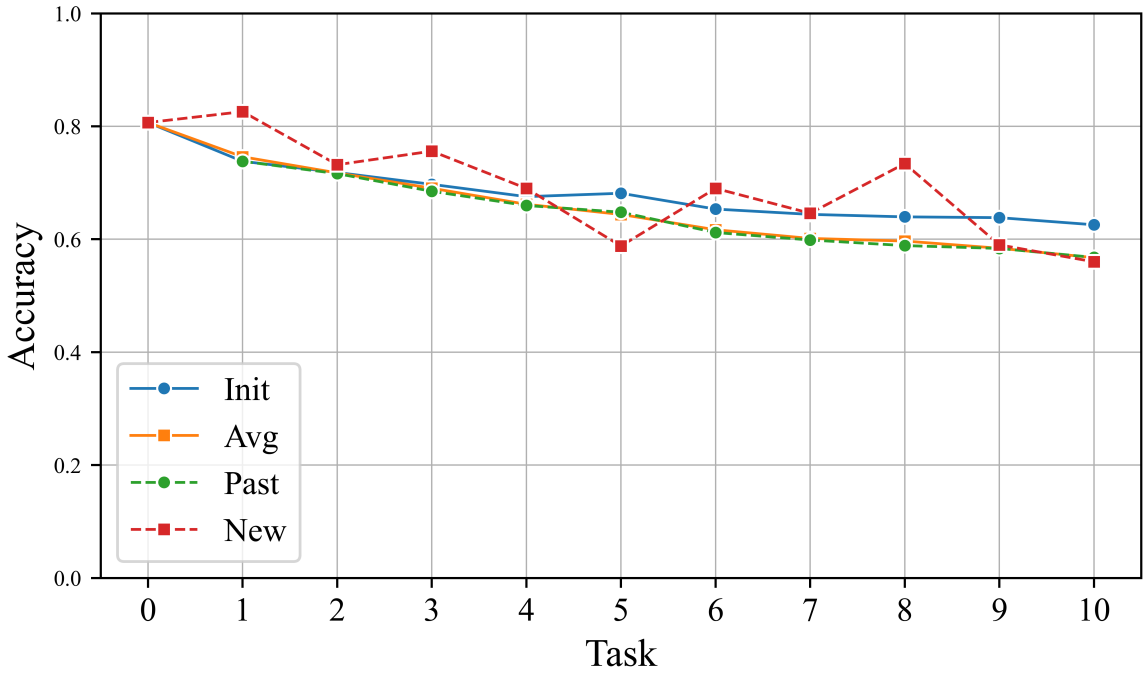


Figure 4.9: Stability-plasticity trade-off, comparing seen class accuracy (Avg) on a subset of New, Past, Initial (Init) classes. GBM_8^T on *CIFAR100* T=10.

Influence of training with STE

p	test acc. w/o STE phase	test acc. w/ STE phase
1	69.9	81.47
2	75.0	81.24
4	76.4	81.18
ResNet-18		81.54

Table 4.3: Ablation study on the STE training phase during the initial training with Thermometer (GBM^T) on *CIFAR100* (T=10).

Table 4.3 reports the initial test accuracy for p=1,2,4 with and without a second training phase with STE, in GBM^T. The results indicate that this second training phase is necessary so that a ResNet-18 with binarization reaches the same performance as without binarization, even for low-precision thermocodes on 1-bit (p=1). It confirms that $\mathcal{F}$ can effectively be trained to mitigate quantization error and learn feature clipping.

Influence of the BMM initialization

$\boldsymbol{\mu}$ init.	$\boldsymbol{\pi}$ update	K=1	K=2	K=4
random	fixed	65.7	66.0	66.9
random	trainable	66.2	67.1	67.3
centroid	fixed	66.1	66.7	67.0
centroid	trainable	66.0	67.4	67.5

Table 4.4: Influence of the BMM's hyper-parameters on the final average accuracy. Results are reported for GBM_1^T on *CIFAR100* (T=5).

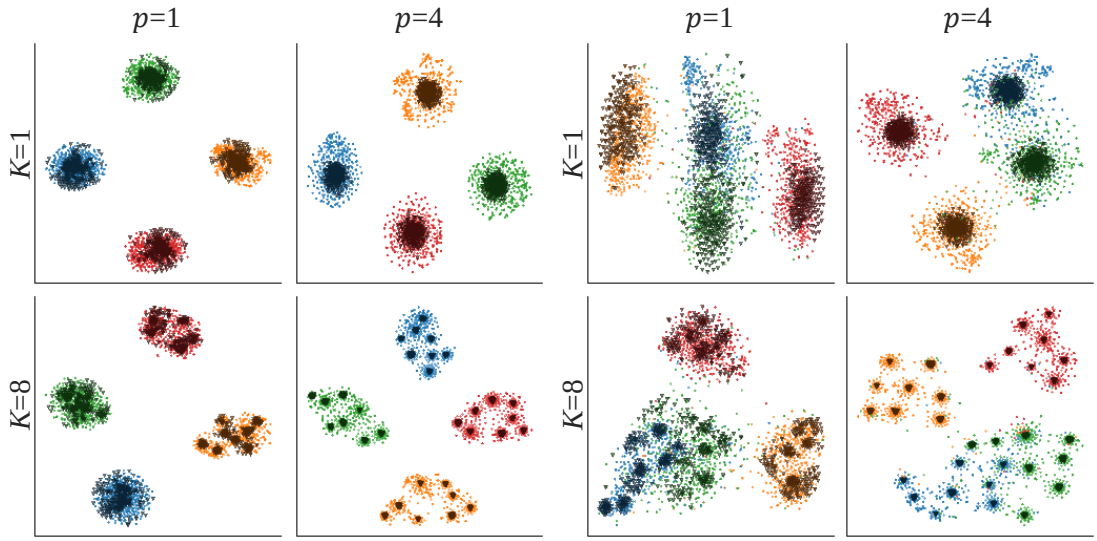
Table 4.4 presents the results for the combinations of the EM-algorithm's $\boldsymbol{\mu}$ initialization and $\boldsymbol{\pi}$ update, for K=1,2,4. When the number of prototype augments, *centroid* initialization is preferable to a random one. Even if intra-class clusters are noticeable (Figure 4.10) they spread closely to the class centroid. Making $\boldsymbol{\pi}$ trainable improves final average accuracy over having them fixed, meaning that accurate mixing coefficients boost performance. For each K, the relative difference among all cases never exceeds 1.5%. Table 4.4 therefore demonstrates that for such a CIL scenario, our method remains robust to a trainable $\boldsymbol{\pi}$ with a random $\boldsymbol{\mu}$ initialization.

Stability-Plasticity trade-off

CIL methods should preferably ensure similar classification performance for every class, new and past. To reach this goal, the number of pseudo-exemplars generated at each iteration is chosen in order that each batch contains an equal number of elements per class. We empirically explore plasticity and stability by evaluating the accuracy on the initial, new, past, and all seen classes in Figure 4.9. Several observations can be made. New classes are better classified than past classes (except on task 5). Even with a fixed $\mathcal{F}$, the classifier can adapt to new classes. Nevertheless, the average relative difference between past and new

accuracy is 4.5%, indicating a good trade-off between stability and plasticity. Finally, initial classes are better classified than past classes, hence freezing the encoder slightly prioritize ($< 6\%$) the retention of initial classes.

Feature Visualization



a) t-SNE on initial task classes b) t-SNE on incremental task classes

Figure 4.10: Embedding visualization on *CIFAR100* with GBM_1^T. a-b) t-SNE [134] on 4 classes, samples (light shade) and generated pseudo-exemplars (dark shade). c) Absolute difference between the correlation matrices of exemplars (dataset samples) and pseudo-exemplars.

Figure 4.10 illustrates the generation of pseudo-exemplars for GBM^T, with t-SNE [134] transformations of real exemplars and pseudo-exemplars for K=1,8 and p=1,4. For a single prototype, K=1, on both the initial and incremental classes, pseudo-exemplars effectively approximate the sample distribution close to the centroid; however, farther away from the centroid, pseudo-exemplars do not completely cover the sample distribution, particularly when the thermocode precision is higher, *i.e.*, p=4. This may be due to the fact that the thermometer coding introduces feature correlations, and a BMM with a single prototype fails to represent this correlation, as illustrated in Figure 4.11. Note that the Frobenius distance between the correlation matrices of samples end pseudo-examplars remains large. In contrast, with K=8, pseudo-exemplars more precisely capture sub-modalities within the class distribution, even when p=4. This finding is further supported by a decrease in Frobenius distance of 33%. Moreover, when comparing in Figure 4.10 initial classes to incremental classes, we observe that the inter-class distance among incremental task classes is qualitatively smaller than that of the initial task classes. Consequently, employing multiple prototypes is advantageous, as it eases the generation of pseudo-exemplars closer to the decision boundaries.

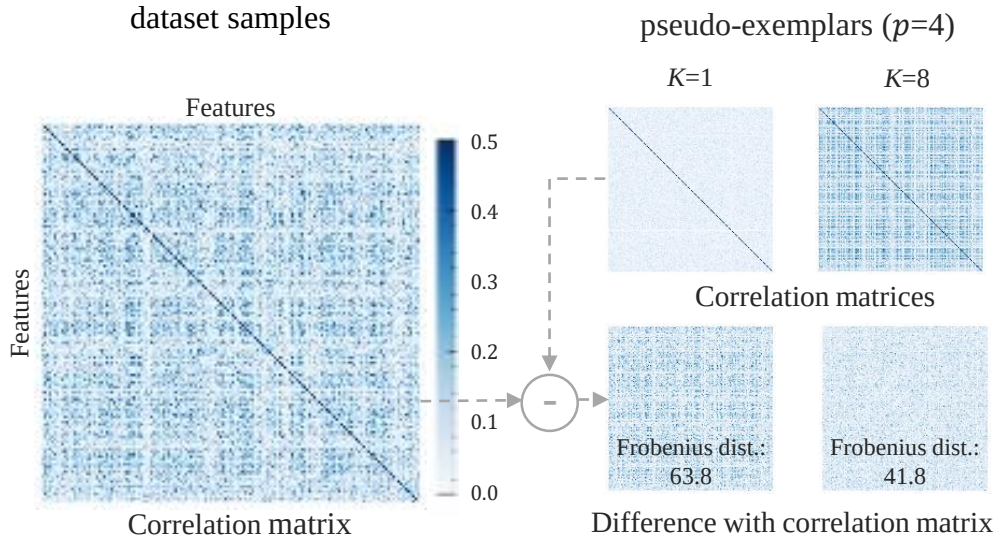


Figure 4.11: Absolute difference between the correlation matrices of exemplars (dataset samples) and pseudo-exemplars for GBM_4^T on classe 0

Method		Memory size	Final accuracy
LR+CWR*	E=75	9.4 Mb	69.5%
[214]	E=100	12.5 Mb	70.6%
	E=150	18.8 Mb	70.8%
GBM	K=1	4.0 Mb	73.9%
	K=4	16.0 Mb	74.2%
	K=8	32.0 Mb	74.6%

Table 4.5: Comparison to [214] on CORE50 NC benchmark.

4.5 Evaluation on BNN architectures

As a reminder, the literature on CIL for BNN addresses two types of networks: (1) hybrid BNN, for TinyML applications, where the model size is $\sim$ 10Mb and full-precision operations are still authorized [16]; (2) Fully BNN, suited to specific hardware technologies and architecture design, where the model size is $\sim$ 1Mb and binary-only arithmetic is permitted during inference. Here we show the applicability of our GBM in both distinct model types. The accuracy and buffer memory requirements, $\mathcal{M}$, are compared only against LR approaches that scale to large datasets: $LR + CWR^*$[214], and our previous work in Chapter 3 [17]. The buffer memory size of the GBM prototypes depends on the number of prototypes K, prototype precision q, embedding size D, and total number of classes n_c, with $\mathcal{M}(\text{GBM}) = K \times D \times n_c \times q$ bits. For binary LR, the memory size depends on the number of latent examplars E, with $\mathcal{M}(\text{LR}) = E \times D \times n_c \times 1$ bits.

4.5.1 Evaluation on hybrid-BNN

We compare GBM to the LR+CWR* method, adhering to their network and dataset settings [214]. As in LR+CWR*, in GBM, the QuickNetLarge backbone [16] is utilized as the feature extractor and is pre-trained on the ImageNet [50] dataset. QuickNet combines binarized 3×3 convolutional blocks with residual connections, batch normalization, global average pooling, and dense layers. The model has 23 million parameters, of which 700,000 are in 32-bit floating point. These floating-point operations occur in the first convolutional layer, the 1×1 convolutions in transition blocks, and the final dense layer, balancing efficiency and accuracy.

Following LR+CWR*, the feature extractor, $\mathcal{F}$, ends in the middle of the convolutional part of QuickNet at the *quant_conv2d_30* layer, producing a $D = 12544$-dimensional feature vector. The classifier, $\mathcal{G}$, consists of the subsequent layers, including convolutional blocks, global average pooling, and dense layers.

For evaluation, still following LR+CWR*, we adopt the CORE50 benchmark in the *NC scenario* [128]. On the contrary to Section 3.7, classes are regrouped in 10 super-classes of domestic objects. The goal is to learn to classify 10 domestic objects incrementally across 9 tasks. The scenario starts with 2 object classes in the first task, and subsequent tasks progressively add more classes. Model accuracy is measured on a fixed test set comprising sessions $\#3$, $\#7$, and $\#10$ after each incremental learning experience.

Table 4.5 reports final accuracy for GBM and LR+CWR* for different sizes of memory. With K=1, GBM gives $+3.1\%$ higher final accuracy than LR+CWR* with the largest buffer, and requires $\times 4.7$ less memory space. Increasing the number of prototypes slightly increases the final accuracy by an absolute $+0.5\%$.

4.5.2 Evaluation on 3Mb-BNN

We compare GBM with latent replay on our specifically tailored *3Mb-BNN* using the *CIFAR100* dataset. A key objective is to minimize memory requirements for CIL with BNNs, particularly in highly constrained applications.

While native and latent replay explore whether it is more efficient to store input data or latent activations in a fixed-size memory buffer, a similar question arises when comparing the storage of latent samples versus latent class statistics, *i.e.*, prototypes. To address this, we plot the final accuracy as a function of memory size for latent replay and different versions of our pseudo-replay method, in a manner similar to Figure 3.21. Figure 4.12 illustrates the memory size required by CIL and the average incremental accuracy on *CIFAR100* for $T = 10$.

Three cases are compared: (1) LR with E ranging from 1 to 500, (2) GBM K=1 with q ranging from 1 to 32 to assess prototype quantization, and (3) GBM q=8 with K ranging from 1 to 16.

GBM prototypes with $q \leq 2$ cannot properly model the embeddings distribution, therefore performing worse than LR. Nevertheless, for $q \geq 4$, GBM outperforms LR with a peak at q=8. To achieve the same accuracy (50.4%), LR requires a $\times 10$ larger replay buffer. With q=8 performance can still be improved up to an additional $+2\%$ by increasing the number of prototypes from 1 to 16. Prototypes quantization thus increases performance while limiting the memory required ($\leq$ 10Mb).

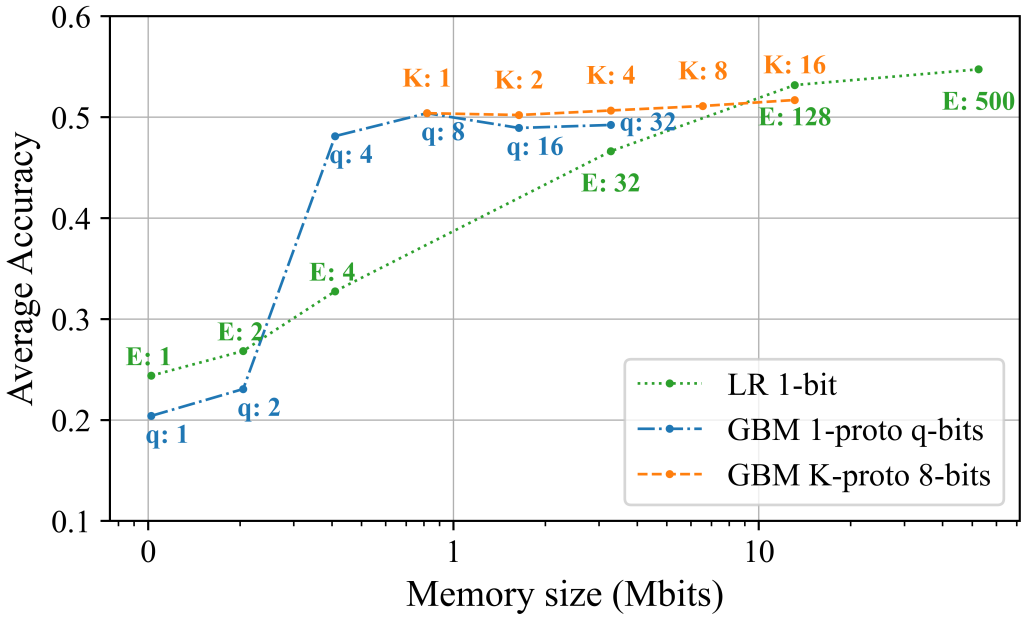


Figure 4.12: Fully-BNN on CIFAR-100, LR and GBM: average incremental accuracy versus required memory size.

4.6 GBM limitations and possible extensions

As Fetril [164], the final accuracy of our approach relies on a fixed $\mathcal{F}$ and a linear classifier, making CIL performance dependent on the initial training on $\mathcal{D}_0$. GBM can be extended to a trainable $\mathcal{F}$, constrained with knowledge distillation [242, 241].

Furthermore, unlike standard CIL scenario benchmarks, in real-world applications, new classes might not emerge in large, discrete task datasets but as a new knowledge acquired sample-by-sample in an online manner [72]. Nevertheless, online versions of the EM-algorithm [33] could extend the GBM to Online-CIL [8] (Figure 4.13 and Few-Shot CIL [204]. In Section 4.7, we present a preliminary study on extending GBM to an Online-CIL setting.

On ResNet-18, we employ a linear classifier, which constrains the classes embedding to be linearly separable. Our multi-prototype approach offers the potential to approximate non-linearly separable classes distribution. Thus GBM could further benefit from more complex classifiers.

4.7 Early attempts for GBM hardware mapping

The GBM method introduced in this chapter has proven effective in CIL scenarios using a conventional offline Expectation-Maximization (EM) algorithm, which assumes unrestricted access to all training samples in a task $\mathcal{D}^t$ and relies on 32-bit floating-point arithmetic. However, real-world embedded deployments—such as smart sensors and near-sensor processing units—operate under much stricter constraints. These platforms typically offer limited data access, low memory, and restricted computational power, and often cannot store large batches of input data for later reuse. To adapt GBM to such environments,

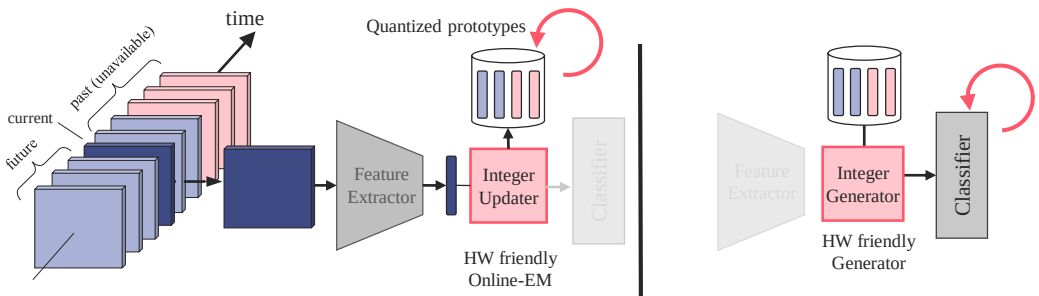


Figure 4.13: An adapted GBM method for the online CIL scenario using fixed-point arithmetic. The prototype memory is updated first, enabling the method to handle a continuous stream of data.

we move toward an online learning setting, where prototypes and the classifier are updated as new data points arrive, using fixed-point integer arithmetic.

Motivation: always-on / on-demand paradigm. The move to online GBM is driven by the always-on/on-demand paradigm, which structures the computation between two hardware components:

- An "always-on" module continuously updates the prototype memory in real time. This module must use ultra-low-power logic and therefore relies on simple integer operations and limited storage.

- An "on-demand" module retrains the classifier when energy is more available (*e.g.*, when a solar-powered system is fully charged). This can be done less frequently and with more computational flexibility.

Such a paradigm imposes a reordering of GBM's two main steps. Instead of first retraining the classifier on the real data and then computing class prototypes, we invert the steps: the system first updates the prototypes using the incoming data, then generates pseudo-exemplars to retrain the classifier (see Figure 4.13). This inversion is critical for online learning since it enables a single-pass update: the system can discard the training data after prototype update, avoiding the need to store it in memory.

Online learning with fixed-point integer arithmetic. To make GBM compatible with constrained hardware platforms, we adapt the EM algorithm to operate in an online fashion using only integer arithmetic. This version supports continuous updates of the prototype memory as new data arrives, without relying on floating-point operations or storing past data.. This design fits fixed-point hardware pipelines and prepares the method for future Register Transfer Level (RTL) implementation in ASICs.

This section introduces the online and hardware-oriented version of GBM in three parts. First, we describe the design of a fully integer-based EM algorithm for fixed-point prototype updates and binary sample generation, using two Pseudo-Random Number Generators (PRNGs) for component selection and Bernoulli sampling. Second, we present a preliminary evaluation in an online Class-Incremental Learning setting on *CIFAR100*, simulating

incremental class arrival without storing inputs. Finally, we outline deployment perspectives for embedded systems, emphasizing the potential for integrating novelty detection and building a fully autonomous, on-device incremental learning pipeline.

4.7.1 Online-fixed-point EM for GBM update

This section presents an online version of the Expectation-Maximization (EM) algorithm tailored for the GBM updater block using unsigned fixed-point arithmetic. As in Section 4.3.2, we focus on learning a Bernoulli Mixture Model (BMM) for a single class c, using a stream of binary latent training samples $\{\boldsymbol{z_i}\}$.

The procedure optimizes the K BMM prototypes $\{\boldsymbol{\mu_k}\}$ to match the distribution of training samples $\{\boldsymbol{z_i}\}$, accessing each z_i only once. It updates the prototypes using q-bit unsigned integer fixed-point arithmetic, where each feature value of $\boldsymbol{\mu_k}$ belongs to $N_q = \{0, 1, \ldots, 2^q - 1\}$. All updates occur directly within this quantized space.

The core procedure consists of three steps applied to each incoming sample. First, in the initialization step, prototypes are set near the midpoint of the quantization range with added random offsets to ensure diversity. Then, for each new sample, the estimation-block assigns it to the closest prototype based on the L1 distance. Finally, the update-block adjusts the selected prototype using an IIR-style recursive rule. These steps are summarized in Algorithm 3 and detailed in the following subsections.

Algorithm 3 Online integer-only EM for GBM update.

Require: Streamed samples $\{\boldsymbol{z_i}\}$, number of prototypes K, Precision q, step size Δ
Ensure: Updated prototypes $\{\boldsymbol{\mu_k}\}$

Initialization:
$\boldsymbol{\mu_k} \leftarrow 2^{q-1} - 1 + \text{Uniform}(-2^{q-2}, 2^{q-2}), \quad k \in \{1, \ldots, K\}$
Procedure:
for each $\boldsymbol{z_i} \in \{\boldsymbol{z_i}\}$ **do**
    Estimation-block:
    $\boldsymbol{z_i^*} \leftarrow \boldsymbol{z_i} \times (2^q - 1)$
    $r_{i,k} \leftarrow D \cdot (2^q - 1) - \|\boldsymbol{z_i^*} - \boldsymbol{\mu_k}\|_1, \quad k \in \{1, \ldots, K\}$
Update-block:
    $ke \leftarrow \arg\max_k \boldsymbol{r_i}$
    if $\boldsymbol{\mu_{ke}} \leq 2^{q-1}$ **then**
        $\eta \leftarrow \Delta \cdot (2z_i - 1) + \Delta$
    else
        $\eta \leftarrow \Delta \cdot (2z_i - 1)$
    $\boldsymbol{\mu_{ke}} \leftarrow \max(0, \min(2^q - 1, \boldsymbol{\mu_{ke}} + \eta))$

Initialization

The initialization of the Binary Mixture Model (BMM) parameters is designed to provide a balanced starting point for the learning process. The prototypes ($\{\boldsymbol{\mu_k}\}$) are initialized to values centered around the midpoint of the quantization range $[0, 2^q - 1]$. Specifically, each prototype is first set to $2^{q-1} - 1$, ensuring that the initial values are neutral within

the range. To introduce diversity among the prototypes, a random offset sampled uniformly from $[-2^{q-2}, 2^{q-2}]$ is added to each component. This variability prevents identical initialization of the prototypes, avoiding symmetry or degeneracy issues that could hinder learning.

Estimation-block

The implementation of the estimation block is illustrated in Figure 4.14. Its purpose is to compute a proxy for the responsibility vector. This block does not use information from $\{\pi_k\}$, assuming balanced weights for all prototypes. This assumption aligns with the GBM approach, which has been shown to be compatible with fixed balanced $\boldsymbol{\pi}$. It simplifies the hardware implementation without introducing significant limitations.

The proxy responsibility vector $\boldsymbol{r_i}$ for a data sample $\boldsymbol{z_i}$ is determined by the L1 distance between the q-bit unsigned fixed-point representation of each prototype $\{\pi_k\}$ and the q-bit unsigned fixed-point representation of z_i, denoted $\boldsymbol{z_i^*}$. To obtain $\boldsymbol{z_i^*}$, the 1-bit binary vector $\boldsymbol{z_i}$ is expanded q-times by duplication (see Figure 4.14). The responsibility component $r_{i,k}$ for prototype $\boldsymbol{\mu_k}$ is computed as:

$$r_{i,k} = D \times (2^q - 1) - \|z_i^* - \mu_k\|_1 \tag{4.8}$$

where D is the embedding dimension, and $D \times (2^q - 1)$ represents the maximum possible value of the L1 distance with q-bit fixed-point arithmetic.

This results in a proxy responsibility vector $\boldsymbol{r_i}$, equivalent to $\boldsymbol{\gamma_i}$, with larger components indicating closer proximity to the corresponding prototype. To classify $\boldsymbol{z_i}$ to its closest prototype, the index of the maximum component in $\boldsymbol{r_i}$ is selected using an ARGMAX operation, yielding the index ke of the closest estimated prototype.

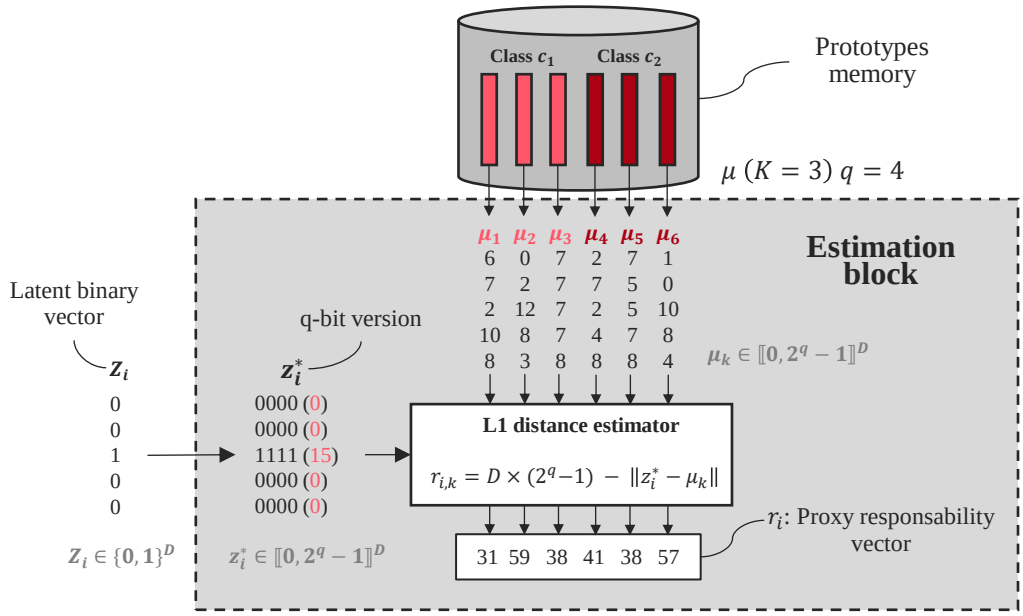


Figure 4.14: Schematic representation of the estimation-block.

Update-block

The update block refines the prototypes $\{\boldsymbol{\mu_k}\}$ by estimating the expected value of a Bernoulli distribution using a recurrent low-pass filtering mechanism. This process ensures efficient updates based on $\boldsymbol{z_i}$ (the input sample) and $\boldsymbol{r_i}$ (the responsibility vector) computed in the estimation block. To avoid the need for buffer storage, an Infinite Impulse Response (IIR) filter is employed. The update procedure consists of the three following steps:

1. **Step 1**: Identify the prototype to update.

$$ke = argmax(r_i)$$

   This selects the prototype ke most responsible for the input sample $\boldsymbol{z_i}$, based on the highest responsibility score in $\boldsymbol{r_i}$.

2. **Step 2**: Compute the update increment.

$$\eta = \Delta \cdot (2\boldsymbol{z_i} - 1)$$

   This calculates the direction and magnitude of the update, bringing the prototype closer to the input sample $\boldsymbol{z_i}$ in binary space. Here, Δ represents the step size for the update. We set Δ to the minimal value, *i.e.*$\Delta = 1$, to prevent rapid saturation of the prototypes.

3. **Step 3**: Apply the update with clipping.

$$\boldsymbol{\mu_{ke}} = \text{Clip}(\boldsymbol{\mu_{ke}} + \eta, 0, 2^q - 1)$$

   The $\text{Clip}(x, a, b)$ function ensures the updated prototype remains within the valid range $[0, 2^q - 1]$, defined as $\text{Clip}(x, a, b) = \max(a, \min(x, b))$. This applies the update while preventing the prototype values from exceeding the valid range through

Challenges and solutions

- **Unsigned arithmetic bias:** When prototype components approach or exceed half the dynamic range ($2^{q-1} - 1$), updates can introduce a bias due to the asymmetric behavior of unsigned arithmetic. To address this, a corrective term is added to η, which is computed as:

$$\eta = \begin{cases} \Delta \cdot (2\boldsymbol{z_i} - 1) + \Delta, & \text{if } \boldsymbol{\mu_{ke}} \leq 2^{q-1}, \\ \Delta \cdot (2\boldsymbol{z_i} - 1), & \text{otherwise.} \end{cases} \tag{4.9}$$

  This ensures balanced updates across the entire range, regardless of whether prototype values are above or below the midpoint.

- **Boundary saturation:** Input samples are binary values ($\{0, 1\}$), which means prototypes are updated in discrete steps of $+1$ or -1. Starting updates from the midpoint (2^{q-1}), prototypes have a usable range of 2^{q-1} in each direction. An IIR filter is typically designed to estimate the mean across a range of values, but with binary inputs,

the updates primarily operate near the range boundaries. This behavior can lead to dramatic saturation issues at 0 or $2^q - 1$, as the finite precision prevents smaller, more gradual updates. To address this, saturation is mitigated by ensuring the precision q is sufficient relative to the number of samples considered for BMM online update. For CIFAR-100, with 500 samples per class, $q = 10$ is chosen based on the heuristic $q \geq \log_2(2 \cdot 500) \approx 10$, where the factor 2 accounts for the symmetric range available for updates starting from the midpoint, allowing for both positive and negative adjustments.

Generation-block

To address the hardware implementation of our GBM methods, we also propose a schematic representation of the generation block in scenarios where mixing coefficients are not required. Figure 4.15 illustrates this schematic, focusing on the generation of a single pseudo-exemplar $\boldsymbol{Z_{pe}}$. The block takes as input a prototype memory buffer, represented here with 4 classes, each characterized by $K = 3$ prototypes quantized to $q = 4$ bits (i.e., values ranging from $[0, \ldots, 15]$), with each prototype having $D = 5$ components.

Two Pseudo-Random Number Generators (PRNGs) perform prototype selection and Bernoulli sampling. The first, *s-PRNG*, generates binary vectors for selecting mixture components. The second, μ-PRNG, draws samples from Bernoulli distributions. Both generators produce decorrelated bits, assuming a Bernoulli distribution with $p = 0.5$.

The process is detailed as follows: First, a prototype $\boldsymbol{\mu_s}$ is randomly selected. The *s-PRNG* generates a vector $\boldsymbol{v_s}$ consisting of $\log_2(K \times N_c)$ bits, where N_c represents the maximum number of classes. This vector specifies the address of the prototype to be selected. The output controls the *Selector*, a $K \times N_c$-input multiplexer comprising a bus of $M \times q$ bits that encodes the binary representation of the selected prototype $\boldsymbol{\mu_s}$. Note that if $\boldsymbol{v_s}$ exceeds the range of valid prototype indices, a redrawing mechanism can be applied, although this is not depicted here.

Next, the μ-PRNG generates a bit vector $\boldsymbol{v_\mu}$, consisting of $D \times q$ bits. This vector is compared to $\boldsymbol{\mu_s}$ in the "Bernoulli drawing" block. The output corresponds to the pseudo-exemplar $\boldsymbol{Z_{pe}}$, where the i-th component of $\boldsymbol{Z_{pe}}$ is 0 if $\mu_{s,i} \leq v_{\mu,i}$, and 1 otherwise. This process is illustrated with numerical examples in Figure 4.15.

4.7.2 Discussion on preliminary results

Online-CIL scenario: To preliminarily evaluate the fixed-point online implementation of GBM, we adapt the CIFAR-100 Class-Incremental Learning (CIL) scenario from Section 4.3.1, using $T = 5$ tasks and a *3Mb-BNN*. These experiments are exploratory in nature and do not represent consolidated results. During the initial task, corresponding to the pre-training phase, both the encoder and classifier are trained offline with unrestricted access to the training data. In the incremental tasks, the online-CIL scenario introduces the constraint that each training data point is seen only once. To address this, during task t, the prototype memory is updated using Algorithm 3 on $\mathcal{D}_t$. The classifier is subsequently trained exclusively on synthetic pseudo-exemplars, following the same optimization process outlined in Section 4.5.2.

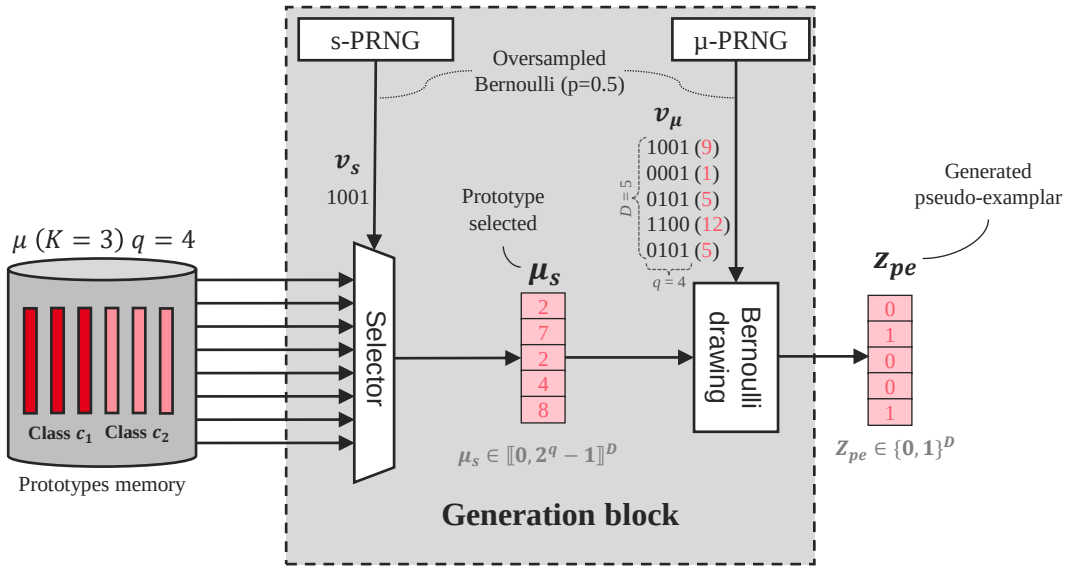


Figure 4.15: Schematic representation of the generation-block.

Compared methods: To assess the impact of the added constraints, we consider three key factors: the EM algorithm, the prototype representation, and the training data used for the classifier. The EM algorithm is evaluated in three configurations: offline, offline post-training quantization (PTQ), and online. For the prototype representation, we compare 32-bit floating-point precision with 10-bit fixed-point precision. Finally, the classifier is trained either on synthetic samples generated from the GBM or on a combination of new training data and synthetic samples ($\mathcal{D}^t$ + GBM). These considerations result in four configurations, with the corresponding CIL performance reported in Table 4.6.

Plasticity-stability investigation: Figure 4.16 presents the per-class accuracy for different configurations on new classes, old classes, and the initial classes used for pre-training. Metrics for the last task are summarized in Table 4.6. Retraining solely on synthetic data improves retention of the initial classes but significantly reduces adaptation to new classes for both offline and online EM configurations. The inability of GBM methods to adapt to new classes in both configurations suggests that this limitation is not due to our proposed update rule for online EM.

This behavior may be explained by two potential causes. First, the regularization on the initial task might be too strong, preventing adaptation to new training data. Second, the generated samples might lack sufficient diversity, failing to generalize to the full distribution of new classes. The Figure 4.17 supports this explanation. It reports the accuracy for offline PTQ trained on GBM-generated data in a FPT scenario, where the initial classes are not retained. In this setup, the network is able to adapt to new classes while also retaining knowledge of old ones. However, current experiments do not provide conclusive evidence as to why retaining initial classes prevents effective learning of new ones when training the classifier exclusively on generated pseudo-exemplars. Further experiments are necessary to better understand the underlying causes of this phenomenon.

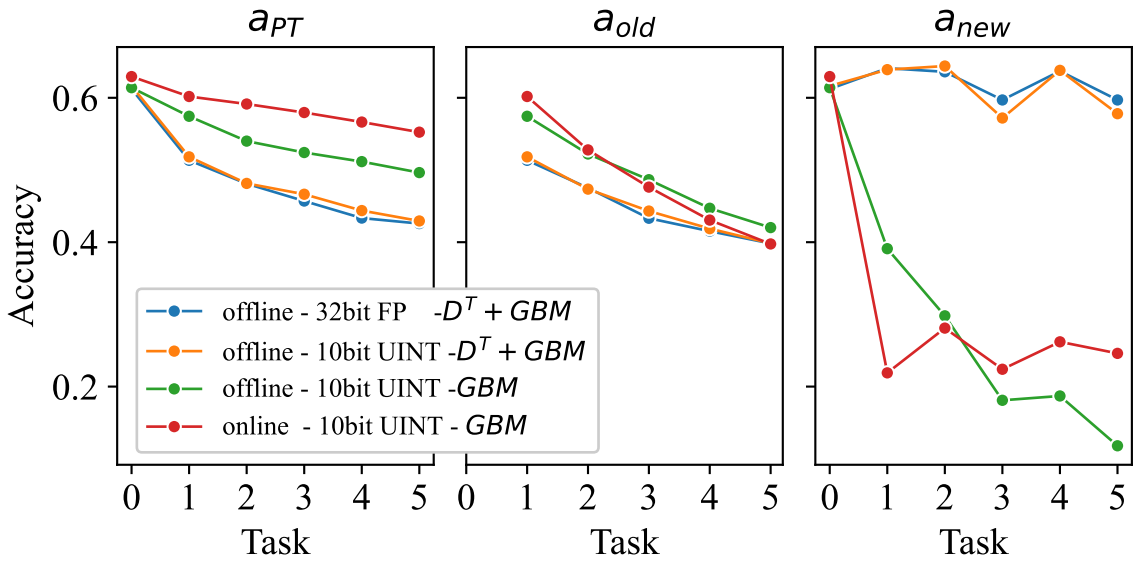


Figure 4.16: Evaluation of initial, old, and new classes accuracies from different variant of the GBM update-block and classifier update-database.

EM algorithm	Classifier update	Prototype precision	**Final Accuracy (%)**			
			Avg	New	Old	Init
Offline	$\mathcal{D}^t + GBM$	32-bit FP	41.6%	59.4%	39.1%	42.9%
Offline PTQ	$\mathcal{D}^t + GBM$	10-bit UINT	41.8%	59.7%	39.8%	42.6%
Offline PTQ	GBM only	10-bit UINT	39.0%	11.8%	42.0%	49.6%
Online QAT	GBM only	10-bit UINT	38.2%	24.6%	39.8%	55.2%

Table 4.6: Investigation of GBM and classifier update strategies on the plasticity-stability trade-off. The Prototype Precision column indicates whether prototypes are represented using 32-bit floating-point (FP) or 10-bit unsigned integer (UINT) formats. Metrics are reported for the last task (task 4) and include the average accuracy (Avg), accuracy on new classes (New), accuracy on old classes (Old), and accuracy on the initial pre-training classes (Init).

Prototype saturation investigation: This analysis evaluates the effect of the number of training samples on the performance of the online EM algorithm and examines whether the anticipated feature saturation phenomenon occurs. With $K = 1$, the optimized prototype $\boldsymbol{\mu}$ provides an approximation of the mean of the latent embedding, assuming the embedding are represented in $\{0, 1\}$ instead of their original $\{-1, +1\}$ representation. To assess the quality of this approximation, we compute the mean absolute error (MAE) between $\boldsymbol{\mu}$ and the mean of the latent embedding for class 0, transformed into the $\{0, 1\}$ range. Results are shown in Figure 4.18. To highlight the saturation effect, we artificially increase the number of training samples by duplicating the training set. We also track the number of features in $\boldsymbol{\mu}$ that saturate to 0 or $2^q - 1$ as a function of the number of seen samples. The results indicate that features begin to saturate around 400 samples, and the MAE reaches

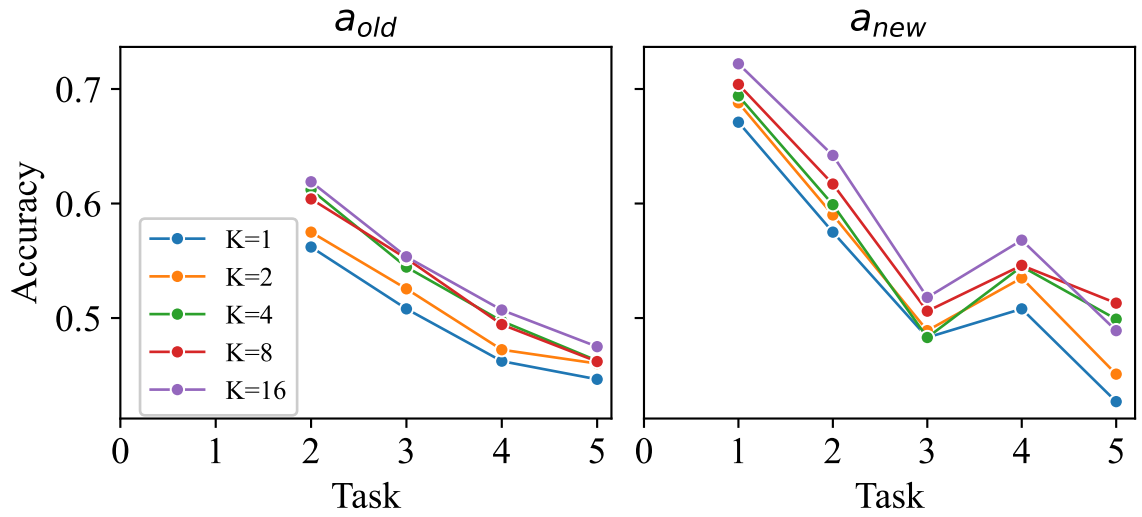


Figure 4.17: Early results on classifier updated only on pseudo-examplars in an FPT scenario. Prototypes are computed with an offline version of GBM with q=8 bits.

its minimum after approximately 600 samples. This supports our heuristic for determining the required precision q based on the number of training samples. However, beyond this point, the MAE increases, likely due to the growing number of saturated features in $\boldsymbol{\mu}$.

Main takeaways: These exploratory results provide an initial evaluation of the fixed-point online implementation of GBM in incremental learning scenarios. While retraining solely on synthetic pseudo-exemplars demonstrates the feasibility of adapting without accessing real task data, a critical limitation is observed: strong regularization from the initial task prevents effective adaptation to new tasks. This observation sees to be true for both offline EM and online EM. Thus it is not a limitation linked to the hardware deployment of the online EM. Solutions have to be investigated to bypass this regularization issue. Additionally, the prototype saturation analysis reveals that the method's reliance on fixed precision ties its performance to the anticipated number of seen samples, making it not entirely task-data agnostic. Other approaches could be investigated to bypass the saturation issue related to number of samples. In the presented approach we use the full range of the integer arithmetic for representing the interval $[0, 1]$. A possible improvement investigation would be to see what happens if we allow values beyond 0 or 1, for example by adding 2 extra bits. This could lead to better saturation control.

These findings are preliminary and do not represent consolidated conclusions. Notably, the multi prototype approach, already implemented in the method, has not been tested in these experiments. Future work should investigate the potential of multi prototype to improve representation capacity and adaptability, alongside efforts to enhance the diversity of generated samples and conduct more detailed evaluations under diverse online incremental learning scenarios.

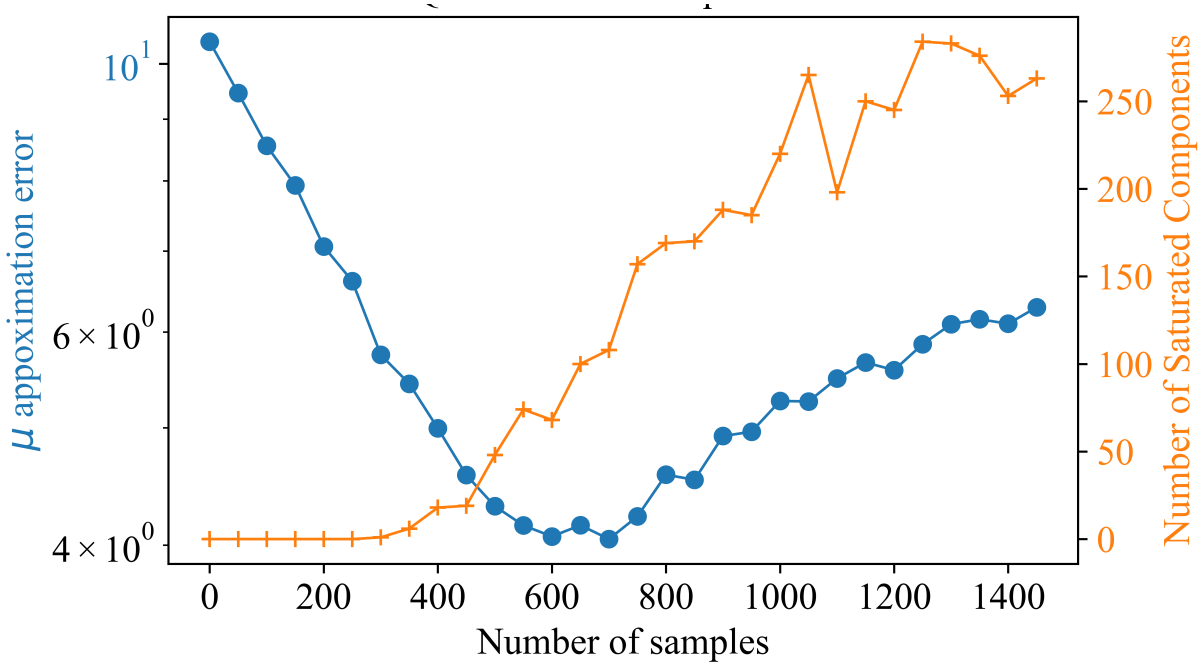


Figure 4.18: Effect of number of training samples of expectation approximation and feature saturation with our online fixed-point EM algorithm on class 0 of *CIFAR100*.

4.7.3 Future direction for hardware mapping

The exploratory results presented in this work demonstrate the feasibility of implementing a fixed-point online version of our GBM methods. While the current evaluation focuses on the case of a single prototype per class and does not retain pre-training classes during incremental tasks, these early results highlight promising directions for further development and refinement. Below, we outline key avenues for future research and implementation.

Multi-prototype implementation on hardware platforms: If further investigations validate the adaptation capabilities of the proposed method, extending it to support multiple prototypes per class will be a natural next step. Both the estimation-block and update-block can be adapted to handle multiple prototypes by incorporating mixing coefficients, allowing for a more granular representation of class distributions.

Novelty detection with the estimation-block: The estimation-block could be leveraged to identify the emergence of new, unseen classes in the data stream. Integrating this capability paves the way to a fully autonomous incremental learning pipeline that does not rely on external oracles for detecting task boundaries or labels. By enabling automatic detection and adaptation to new classes, such a system would align closely with the requirements of real-world, open-ended learning scenarios. Note that even with novelty detection an operator is needed to labeled the newly encountered classes. Promising avenue are found in the junction of autonomous incremental learning and active learning systems [118].

Toward a fully autonomous incremental learning system: The integration of a multi-prototype approach and novelty detection could lead to a continual learning pipeline

that operates without the need for explicit task boundaries or predefined labels. This would allow for the seamless handling of dynamic and evolving data streams. Further experimentation is required to explore the robustness of these extensions and their compatibility with real-world hardware constraints, such as precision, memory, and computational efficiency.

By addressing these challenges and opportunities, future work can aim to consolidate the GBM strategy into a versatile, hardware-optimized framework for handling diverse continual learning scenarios while maintaining computational and memory efficiency.

4.8 Conclusion

In this chapter, we introduced a pseudo-replay approach that models the correlations between latent embedding features using Bernoulli Mixture Models to compactly represent past class statistics. We evaluated the proposed GBM method in a CIL setting on a Fully-Binary Neural Network (*3Mb-BNN*), showing that it maintains competitive performance under tight memory constraints. We further extended the method to larger BNN architectures, such as QuickNet, on the CORE50 dataset. There, GBM outperformed existing BNN-specific CIL methods, achieving a $+3.1\%$ improvement in final accuracy and a $4.7\times$ reduction in memory usage, combining high accuracy with strong memory efficiency.

In addition our work extends the GBM approach in two key directions:

- First, the GBM strategy was generalized to non-binary deep neural networks (DNNs) by introducing a binarizer to map real-valued feature representations to binary embedding. This allowed us to evaluate GBM on a ResNet18 feature extractor, achieving state-of-the-art performance compared to other prototype-based methods. The multiprototype extension, in particular, demonstrated its scalability and robustness to the increase of number of tasks, maintaining high accuracy even with an increasing numbers of tasks and classes. This highlights its potential for application in large-scale continual learning problems where representation granularity is crucial.

- Second, we introduced a hardware-friendly variant of GBM designed for online learning scenarios. This version relies on fixed-point integer arithmetic, laying the groundwork for a future RTL implementation and potential ASIC integration. The envisioned ASIC would embed both the GBM mechanism and a hardware accelerator for BNN inference. Preliminary experiments demonstrated the feasibility of performing online updates using a single prototype per class, confirming the method's suitability for deployment in memory- and energy-constrained environments such as edge devices and smart sensors. Additionally, this implementation naturally supports extensions toward novelty detection, paving the way for a fully autonomous incremental learning system capable of adapting to new classes without predefined task boundaries or external supervision.

While these contributions establish the GBM framework as a promising avenue for incremental learning, several challenges remain. Specifically, further investigations are required to consolidate the multi-prototype approach in online setting and to better understand its behavior under varying constraints, such as data sparsity (few-shot learning) and numerical precision scaling.

Highlights of the Chapter

- The Bernoulli Mixture Model (BMM) effectively encodes and generates binary embedding distributions, enabling efficient pseudo-replay for binary neural networks (BNNs).
- GBM-based pseudo-replay outperforms latent replay on our *3Mb-BNN*, particularly when memory constraints are below 1MB.
- On ResNet18, the multiprototype GBM achieves state-of-the-art performance compared to other prototype-based methods on CIFAR-100 and *TinyImageNet*, demonstrating robustness in high-class-count scenarios.
- The GBM framework is compatible with a fixed-point only arithmetic for compact RTL design aiming a being implemented in ASICs, supporting efficient online updates and real-time data streams, paving the way for highly resource-constrained (*i.e.*, µW's, Mb's) continual learning systems.

Chapter 5

Conclusion and Perspectives

This thesis improves incremental learning in resource-constrained environments by developing novel methodologies for Fully Binary Neural Networks. It presents a comprehensive investigation, from efficient model design and replay strategies toward hardware-aware deployment, ensuring low-power inference in dynamic settings. Our work contributes to the optimization of binary neural networks for class-incremental learning, introduces memory-efficient generative models, and explores the co-design of learning algorithms with hardware constraints. This chapter summarizes these contributions and outlines future research directions, categorized into short-, mid-, and long-term perspectives.

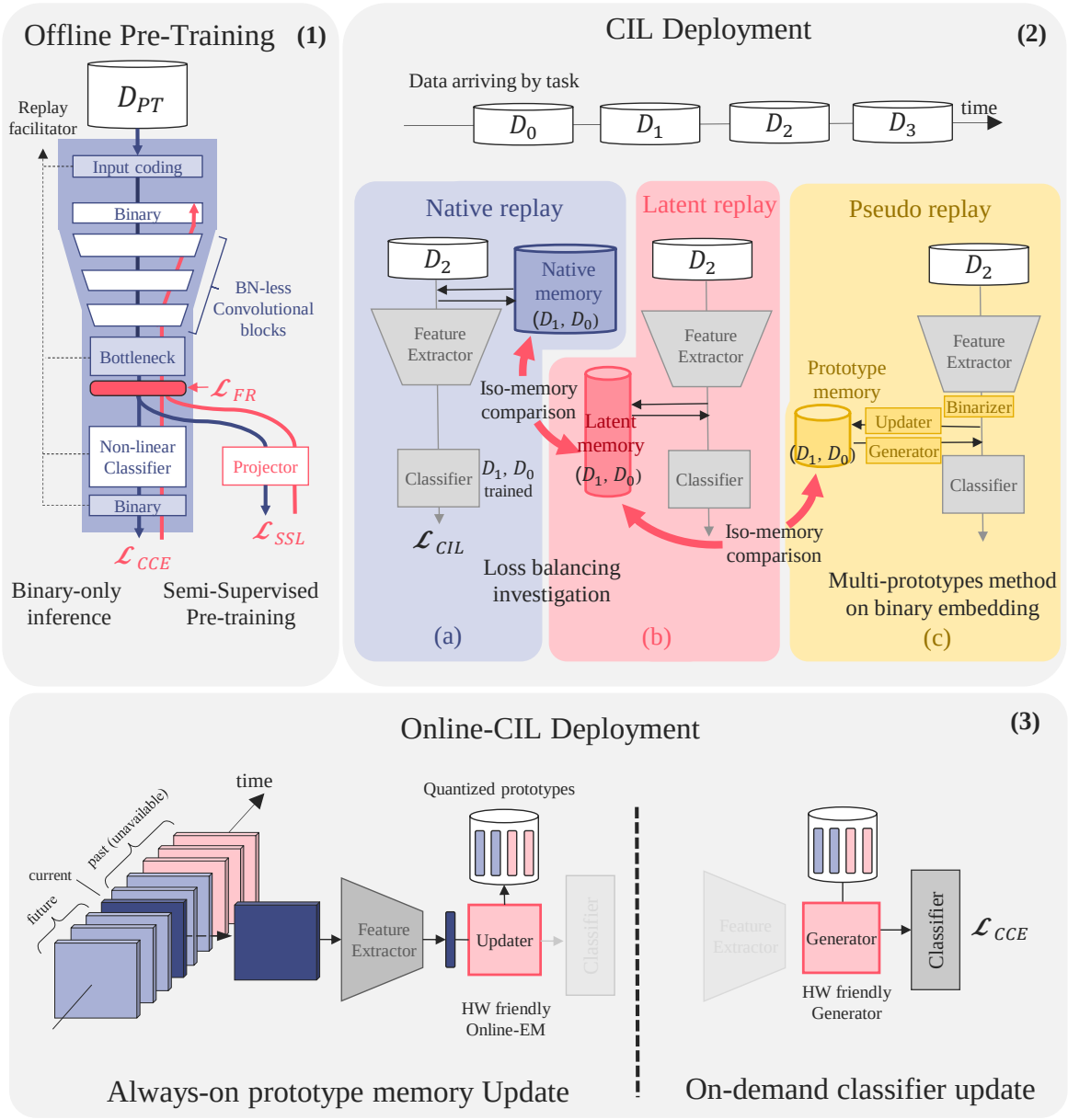
5.1 Contribution overview

The work presented in this thesis has led to the publication of the following contributions:

- **Conference paper:**
  - Y. Basso-Bert, W. Guicquero, A. Molnos, R. Lemaire, A. Dupret, "On Class-Incremental Learning for Fully Binarized Convolutional Neural Networks" in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Singapore, 2024, pp. 1-5, doi: 10.1109/ISCAS58744.2024.10558661.
- **Patent:**
  - Y. Basso-Bert, W. Guicquero, A. Molnos, R. Lemaire, "Dispositif électronique comprenant un réseau de neurones artificiels configuré pour un apprentissage initial et un apprentissage complémentaire et procédés d'apprentissage et d'utilisation correspondant" Patent submitted, FR2501670, 2025.
- **Journal articles (under review):**
  - Y. Basso-Bert, A. Molnos, R. Lemaire, W. Guicquero, A. Dupret, "Toward Experience Replay for Class-Incremental Learning in Fully-Binary Networks," submitted to *ACM Transactions on Embedded Computing Systems*, 2025. Preprint available at https://arxiv.org/abs/2503.07107.
  - Y. Basso-Bert, W. Guicquero, A. Molnos, R. Lemaire, A. Dupret, "Generative Binary Memory: Pseudo-Replay Class-Incremental Learning on Binarized Embeddings" submitted to *Elsevier Neural Networks*, 2025. Preprint available at https://arxiv.org/abs/2503.10333.

The core contributions of this thesis address key challenges in deploying Binary Neural Networks for class-incremental learning. Our work focuses on three major research axes: (1) optimizing binary inference for incremental learning, (2) developing memory-efficient replay methods, and (3) enabling real-world deployment in dynamic environments.

First, we introduce a fully binary inference pipeline tailored for near-sensor deployment, ensuring competitive performance with state-of-the-art BNNs while adhering to continual learning constraints. Second, we systematically explore replay strategies, transitioning from native replay to latent and pseudo replay, all the way to a Generative Binary Memory (GBM) method for binary pseudo-exemplar generation. Lastly, we move toward



Big Picture Contributions

(1) : Advancements toward fully-binary neural network inference in CIL

(1) →(3): Advancements toward real-life deployment in dynamic environment

(a) →(c): Advancements toward generative model for binary memory

Figure 5.1: Overview of the key contributions of this PhD thesis. Advancements toward fully binary neural network inference in Continual Incremental Learning (CIL). Progress toward real-world deployment in dynamic environments. Developments toward generative models for binary memory.

practical deployment by designing an always-on memory prototype for real-time learning and implementing a fixed-point version of GBM for hardware-constrained platforms. Figure 5.1 summarizes these contributions, which are detailed in the following sections.

5.1.1 Toward FBNN inference in CIL

- **Fully Binary inference:** This thesis introduces good practices to design a BNN with fully binary inference while taking into account the requirements of incremental learning with a memory buffer. This method offers a baseline model achieving performance comparable to state-of-the-art BNNs on the *CIFAR100* dataset while staying within the constraints of near-sensor inference. Notably, it reaches 58% test accuracy on *CIFAR100* with a $3Mb$ model, outperforming BNN-BN's 55% accuracy with an $11Mb$ model [39].

- **Semi-supervised pretraining:** A semi-supervised pretraining approach was developed to enhance the transferability of features in binary networks. By combining supervised learning with Barlow Twins regularization and custom activation regularization, the proposed framework produced more distributed and transferable features. This approach demonstrated a $+1.17\%$ improvement in final accuracy on *CIFAR100* compared to conventional supervised pretraining, particularly benefiting Latent replay scenarios where the feature extractor remains fixed during incremental learning.

5.1.2 Toward generative model for binary memory

- **From Native to Latent replay:** Extensive experimentations have compared Native replay, which stores raw input data, and Latent replay, which stores intermediate feature embedding. Results revealed that Latent replay achieves higher retention with smaller memory footprints (<25Mb), while Native replay performs better with larger buffers. This analysis highlights the critical importance of selecting replay strategies based on memory and computational constraints.

- **Study of loss design:** The trade-off between adaptability (performance on current tasks) and retention (performance on past tasks) was addressed through a systematic investigation of loss functions for binary networks. Among Categorical Cross-Entropy (CCE), squared-hinge loss, and focal loss, CCE emerged as the most effective for both Latent and Native replay, delivering higher retention with minimal loss in adaptability.

- **From latent to pseudo replay:** To further enhance memory efficiency, this thesis proposed to go beyond latent replay with a specific pseudo replay, the Generative Binary Memory (GBM) method, for generation of binary pseudo-examplars in BNNs. We also demonstrate that GBM can be extended to any kind of DNN achieving state-of-the-art performance with its multi-prototype approach.

5.1.3 Toward real-life deployment in dynamic environments

- **Early attempts to deal with online streams of data:** This thesis laid the foundation for real-life deployment of binary neural networks in dynamic environments

by studying an always-on memory prototype system. This system processes continuously arriving data streams incrementally, integrating efficient replay mechanisms to adapt to new tasks while retaining past knowledge.

- **Fixed-point implementation of our GBM:** A hardware-friendly, integer implementation of the Expectation-Maximization (EM) algorithm for GBM enables deployment on memory- and computation-constrained devices. This implementation ensured efficient pseudo-examplars generation and memory updates, demonstrating promising results in on-demand hardware platforms. This approach represents a step toward bridging the gap between theoretical models and practical real-world applications.

5.2 Perspectives

Building on the contributions of this thesis, several research directions emerge for improving FBNNs in continual learning, extending their capabilities, and facilitating their deployment on hardware. Future work spans three interconnected areas: advancing FBNN architectures and training strategies, refining incremental learning techniques for real-world applications, and integrating learning algorithms with hardware accelerators.

To structure these research opportunities, we outline short-term directions that can be explored immediately, mid-term objectives requiring further experimental validation, and long-term visions that could shape future advancements in scalable and energy-efficient incremental learning.

5.2.1 FBNN architecture and pre-training

This section outlines future research directions for advancing the FBNN developments in this thesis, focusing on network architecture design and training strategies.

Short-term: Improving computational efficiency and accuracy in our topology starts with optimized residual connections. In *Res-BNN*, concatenation-based feature fusion eliminates 32-bit precision needs, enabling deeper networks for fine-grained tasks. Adaptation of a **ternary-specific residual connection** proposed in [154] with MUX-OR-based connections, replacing 32-bit additions with logic gates while mitigating distribution imbalances between -1 and +1 values. Advanced gating mechanisms could further improve accuracy, depth scalability, and train-test consistency in FBNNs.

Another promising avenue, directly applicable to our research framework, is teacher-student guided learning for pre-training. Knowledge distillation has proven effective in enhancing low-precision neural networks [145]. Leveraging large DNNs, which are easier to train on extensive datasets, to guide the pre-training of the feature extractor in our compact FBNN could lead to substantial improvements. Moreover, this approach aligns with our pre-training/deployment segmentation strategy, where we advocate maximizing computational resources during pre-training to embed as much knowledge as possible within the network. Comparisons with our proposed semi-supervised pre-training protocol could provide valuable insights. This approach could advance both offline training and pre-training

strategies, facilitating future incremental retraining.

Mid-term: Beyond immediate improvements, the mid-term perspectives focus on integrating recent advances in neural architectures to enhance FBNNs. One promising direction is adapting FBNNs to recurrent neural networks, expanding their applicability to video processing and time-series analysis. Incremental gesture recognition and drifting time-series modeling are particularly relevant applications where incremental learning and always-on inference play a crucial role.

NMCNET [68], a 1Mb binary model for hand gesture classification, has demonstrated the feasibility of low-precision recurrent models. However, its five-step training protocol highlights the challenges of training such architectures. Adapting FBNNs to recurrent structures presents a compelling research avenue, with BillNet serving as a reference point for future implementations in both FBNNs and incremental learning deployments.

Another promising research direction is adapting Vision Transformers (ViTs) to FBNNs. ViTs have emerged as a competitive alternative to CNNs for computer vision tasks, benefiting from ongoing research and improvements in state-of-the-art performance. However, direct binarization of ViTs has shown poor results [110].

BinaryViT [110] highlights the importance of careful architectural design, incorporating best practices from CNNs to mitigate performance degradation. BiViT [76] addresses the large binarization errors in the attention mechanism by making the softmax function trainable, allowing it to adapt to binarization constraints. However, these advances have primarily been developed for hybrid BNNs, where only part of the network is binarized. Their adaptation to fully binary architectures remains an open challenge. Investigating these modifications for FBNNs aligns with the broader objective of making foundation models more compatible with binarization and on-device inference.

Recent progress in ternarizing large language models (LLMs), such as the 1.58-bit LLM [133], reinforces the feasibility of binarizing parameter-heavy architectures. Additionally, ongoing advancements in hardware accelerators [148] are improving computational support for low-precision models, paving the way for efficient deployment of edge AI systems.

Long-term: A key limitation of current FBNN development is the reliance on manually designed pre-training protocols. A promising long-term direction is the integration of meta-learning for pre-training, where the optimization process itself determines the best strategy for initializing and adapting binary networks. Instead of relying on heuristics or fixed pre-training pipelines, meta-learning enables a model to learn how to learn, optimizing hyperparameters, loss functions, and data augmentation strategies in a way that generalizes across tasks [59]. Beyond potentially improving BNN optimizations, it could allowed to take into the incremental learning procedure as demonstrated in meta-learning-enhanced continual learning [91] or reduce training energy cost with few-shot adaptation [178]. It offers a promising direction for lifelong adaptation of binarized models, ensuring efficient knowledge transfer across tasks while mitigating catastrophic forgetting.

In addition to pre-training, the automatic design of network architectures is another critical challenge. Throughout this thesis, BNN architectures were designed based on established best practices for CNNs and empirical trial-and-error, often leading to suboptimal architectures. A promising paradigm shift is the adoption of Neural Architecture Search (NAS) to replace manually crafted designs. As discussed in Chapter 2, recent research has

explored NAS for binarized architectures, demonstrating that NAS-designed BNNs can surpass hand-crafted ones while reducing computational costs per inference [98][240].

An especially relevant direction is Hardware-Aware Neural Architecture Search (HW-NAS) [41][191][123], particularly for ASIC and CIM-based accelerators [101]. HW-NAS integrates hardware constraints into the optimization process, producing network topologies specifically tailored to target applications. Additionally, Accelerator-Architecture Co-Search has been developed to jointly optimize both the neural network and its hardware implementation [228]. A promising avenue for future research is to incorporate binary arithmetic and replay-memory constraints into NAS to enable co-search of accelerators and architectures for application-specific deployment.

A novel direction is the development of Differentiable Logic Gate Networks (DLGNs) [163], which offer an alternative computational paradigm to standard neural networks by constructing architectures from logic gates rather than weighted connections. Unlike traditional BNNs, DLGNs are trained via gradient-based optimization by relaxing binary gate operations into differentiable forms, then discretized post-training for deployment. In the long term, DLGNs could open new pathways for building low-power binary logic networks.

This research direction aligns with the broader AutoML initiative, which seeks to automate deep learning model design, reducing the need for manual intervention. Advances in meta-learning for pre-training and architecture search could accelerate the democratization of efficient, customizable AI systems that seamlessly integrate hardware constraints and real-world requirements.

5.2.2 Incremental learning deployment

This section outlines future research directions for advancing the incremental learning algorithms developed in this thesis.

Short-term: Deploying incremental learning at the edge is viable if re-training can be performed locally with minimal computational overhead. While this thesis primarily focuses on inference constraints, a natural next step is to account for the computational and memory costs of backpropagation. In particular, investigating the impact of reduced gradient precision and computationally lighter optimizers, such as SGD with momentum, on incremental learning performance is a promising direction.

Another promising avenue is the implementation of the pseudo-replay method DreamNet [198], developed within our LIIM laboratory. Unlike previously explored pseudo-replay strategies that store explicit data statistics, DreamNet encodes past knowledge directly within the classifier by leveraging associative memory through reinjection mechanisms. This approach could further reduce memory overhead associated with past knowledge retention. The insights gained in this thesis on replay mechanisms in FBNNs provide a strong foundation for advancing and implementing this method.

Mid-term: Future research should move beyond fixed feature extractors. This thesis focused on them because they require fewer trainable parameters, mitigate the aging effect, and allow for studying the learning of transferable features. The next step is to explore learnable binary representations, which can either specialize for each new task or evolve continuously to accommodate a unified classification problem.

A key research direction involves representation regularization, including knowledge distillation [242], slow-update mechanisms [161], and feature adaptation [89]. Investigating these techniques provides an opportunity to challenge Fertil's claim that fixed representations outperform adaptive ones, particularly in the context of BNNs. One critical question is how deep feature extractor adaptation should extend, as full backpropagation is computationally expensive [161]. Determining when and where adaptation is necessary could be guided by specific indicators such as weight momentum and the number of weight flips, which may provide valuable insights into selecting layers for updates.

Beyond architectural considerations, it is essential to expand our approach to domain-incremental learning and larger datasets like ImageNet. While CIL is useful for studying catastrophic forgetting, it does not address domain drift, which is a more realistic challenge for smart sensors and time-series data. Scaling to complex datasets has become standard practice in deep learning, and this thesis has already made significant efforts to transition compact FBNNs from *CIFAR10* to *CIFAR100* and *CORE50*. Further scaling will require architectural innovations, as discussed in Section 5.2.1, along with adjustments to replay strategies to handle severe class imbalance [223].

Long-term: The long-term vision is TinyAI for fully autonomous sensor-based decision-making. Achieving this goal requires AI systems with human-like adaptability, which presents three major research challenges. First, from a data perspective, autonomous systems must learn unsupervised, adapting continuously with minimal labeled examples in an online fashion, aligning with the vision in [180]. Second, from an energy efficiency standpoint, local update rules are necessary to ensure compatibility with edge computing constraints. Replacing backpropagation in favor of alternative training methods, such as those introduced in Section 2.1, could significantly reduce energy consumption by limiting data movement.

Finally, from a memory perspective, the challenge is to develop efficient knowledge retention mechanisms. Beyond integrating DreamNet to merge buffer memory with the classifier, promising directions lie in associative memory research, particularly with Hopfield networks. Ramsauer *et al.* [173] introduced a modern version of Hopfield networks capable of storing an exponential number of patterns (relative to latent space dimensionality), retrieving them in a single update with exponentially low error rates. Adapting such techniques to binary arithmetic presents new challenges but aligns with the vision of drawing inspiration from biological systems while leveraging numerical hardware strengths.

5.2.3 Hardware integration

This section outlines research directions for integrating incremental learning on FBNNs with hardware, building on the preliminary work presented in Section 4.7.

Short-term: Immediate priorities include consolidating experimental studies on online GBM in both FPT and RPT scenarios to better understand their limitations in adaptation capacity. Since our proposed online GBM supports multiple prototypes, characterizing prototype assignment within the algorithm and monitoring performance gains are essential next steps. Additionally, the Estimation Block can serve as a novelty detection mechanism by leveraging distance-based methods [227]. Its responsibility scores could indicate

out-of-distribution data points, triggering retraining when exceeding a certain threshold. This approach can be evaluated directly within CIL settings by applying anomaly detection methodologies [1] to unseen classes.

Mid-term: If responsibility scores prove effective for novelty detection and online GBM successfully handles multi-prototype learning, a promising direction is to explore Binary Mixture Models (BMMs) as classifiers. As a generative model, BMMs can function as non-linear classifiers [25] while eliminating the need for training a residual MLP, which is energy-intensive due to backpropagation.

A BMM-based approach would fully leverage multi-prototype learning, providing a non-linear decision boundary while maintaining a fixed feature extractor. This setup enables integer-only update rules for edge adaptation, significantly reducing computational complexity. Furthermore, this approach remains compatible with the perspectives outlined in Section 5.2.2, particularly in scenarios where feature extractors are learnable.

Long-term: The long-term goal is to implement our GBM-based methodology in a dedicated near-sensor ASIC, enabling real-time, autonomous adaptation for smart imagers in embedded systems. Achieving this requires integrating the online GBM algorithm into hardware, including the estimator, updater, and prototype memory blocks, while designing a specialized hardware architecture for fully binary feature extraction and inference.

For memory implementation and low-precision execution of the estimator and updater blocks, non-volatile memory (NVM) accelerators, such as RRAM or PCM, provide an efficient way to store prototypes while minimizing power consumption. PCM has already supported continual learning through hyper-dimensional computing for memory implementation, retrieval, and update [95], offering valuable inspiration for our approach.

From an accelerator perspective, compute-in-memory (CIM) architectures and low-precision digital circuits can further enhance efficiency by performing direct updates within memory arrays, reducing data movement and lowering the energy costs of real-time IL.

Testing this hardware-integrated system in real-world applications is essential, where smart imagers must incrementally learn new classes at the edge. In wildlife monitoring, a smart camera deployed in a remote environment must recognize new animal species without requiring cloud-based retraining. In personalized security systems, face or gesture recognition models must continuously adapt to new users. The system must also handle input distribution shifts, such as lighting variations in surveillance cameras or background changes in industrial defect detection, without costly global retraining.

Handling distribution shifts effectively requires adapting the feature extractor. The feature adaptation strategies proposed in Section 5.2.2, including knowledge distillation, necessitate hardware co-design to balance adaptability with computational efficiency while managing the storage of both proxy and binary weights. Current research explores solutions for BNN on-chip learning using specialized memory [210]. This approach can be extended to deal with knowledge distillation.

By integrating energy-efficient memory, hardware-accelerated processing, and real-time adaptation, this research will enable autonomous, lifelong-learning smart imagers that efficiently adapt to new environments with minimal power consumption. These systems will drive fully embedded, real-time continual learning for low-power vision applications, bridging the gap between deep learning and hardware-efficient intelligence at the edge.

Highlights of the chapter

- This thesis advances Fully Binary Neural Networks for class-incremental learning by optimizing binary inference, designing memory-efficient generative models, and enabling real-world deployment.
- We demonstrate that FBNNs can achieve competitive performance with state-of-the-art models while significantly reducing computational and memory costs, making them viable for edge AI applications.
- Future research should focus on refining FBNN architectures, improving continual learning strategies, and integrating learning directly into hardware to enable fully autonomous, low-power AI systems.

Bibliography

[1] Supriya Agrahari, Sakshi Srivastava, and Anil Kumar Singh. "Review on novelty detection in the non-stationary environment". In: *Knowledge and Information Systems* 66.3 (2024), pp. 1549–1574.

[2] Hongjoon Ahn et al. "Uncertainty-based continual learning with adaptive regularization". In: *Advances in neural information processing systems* 32 (2019).

[3] Thalaiyasingam Ajanthan et al. "Mirror descent view for neural network quantization". In: *International conference on artificial intelligence and statistics*. PMLR. 2021, pp. 2809–2817.

[4] Fahdah Alalyan, Nuha Zamzami, and Nizar Bouguila. "A hybrid approach based on SVM and Bernoulli mixture model for binary vectors classification". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE. 2020, pp. 1155–1160.

[5] Khaled Alhaj Ali et al. "MOL-Based In-Memory Computing of Binary Neural Networks". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 30.7 (2022), pp. 869–880.

[6] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. "Expert gate: Lifelong learning with a network of experts". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3366–3375.

[7] Rahaf Aljundi, Marcus Rohrbach, and Tinne Tuytelaars. "Selfless sequential learning". In: *arXiv preprint arXiv:1806.05421* (2018).

[8] Rahaf Aljundi et al. "Gradient based sample selection for online continual learning". In: *Advances in neural information processing systems* 32 (2019).

[9] Rahaf Aljundi et al. "Memory aware synapses: Learning what (not) to forget". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 139–154.

[10] Giuseppe Amato, Fabrizio Falchi, and Lucia Vadicamo. "Aggregating binary local descriptors for image retrieval". In: *Multimedia Tools and Applications* 77 (2018), pp. 5385–5415.

[11] Giuseppe Amato et al. "Combining Fisher Vector and Convolutional Neural Networks for Image Retrieval." In: *IIR*. 2016.

[12] Renzo Andri et al. "YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights". In: *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2016, pp. 236–241.

[13] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (2010), pp. 2787–2805. ISSN: 1389-1286.

[14] Yu Bai, Yu-Xiang Wang, and Edo Liberty. "Proxquant: Quantized neural networks via proximal operators". In: *arXiv preprint arXiv:1810.00861* (2018).

[15] Daniel Bankman et al. "An Always-On 3.8 μJ86% CIFAR-10 Mixed-Signal Binary CNN Processor With All Memory on Chip in 28-nm CMOS". In: *IEEE Journal of Solid-State Circuits* 54.1 (2019), pp. 158–172.

[16] Tom Bannink et al. “Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks”. In: *Proceedings of Machine Learning and Systems* 3 (2021), pp. 680–695.

[17] Yanis Basso-Bert et al. “On Class-Incremental Learning for Fully Binarized Convolutional Neural Networks”. In: *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2024, pp. 1–5.

[18] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. “A comprehensive study of class incremental learning algorithms for visual tasks”. In: *Neural Networks* 135 (2021), pp. 38–54.

[19] Eden Belouadah and Adrian Daniel Popescu. “DeeSIL: Deep-Shallow Incremental Learning”. In: *ECCV Workshops*. 2018.

[20] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. “Estimating or propagating gradients through stochastic neurons for conditional computation”. In: *arXiv preprint arXiv:1308.3432* (2013).

[21] Joseph Bethge, Haojin Yang, and Christoph Meinel. “Training accurate binary neural networks from scratch”. In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, pp. 899–903.

[22] Joseph Bethge et al. “Back to simplicity: How to train accurate bnns from scratch?” In: *arXiv preprint arXiv:1906.08637* (2019).

[23] Joseph Bethge et al. “BinaryDenseNet: Developing an architecture for binary neural networks”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 2019, pp. 0–0.

[24] Joseph Bethge et al. “Meliusnet: Can binary neural networks achieve mobilenet-level accuracy?” In: *arXiv preprint arXiv:2001.05936* (2020).

[25] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[26] Andy Brock et al. “High-performance large-scale image recognition without normalization”. In: *International conference on machine learning*. PMLR. 2021, pp. 1059–1071.

[27] Jacob Buckman et al. “Thermometer encoding: One hot way to resist adversarial examples”. In: *International conference on learning representations*. 2018.

[28] Adrian Bulat and Georgios Tzimiropoulos. “Xnor-net++: Improved binary neural networks”. In: *arXiv preprint arXiv:1909.13863* (2019).

[29] Pietro Buzzega et al. “Dark experience for general continual learning: a strong, simple baseline”. In: *Advances in neural information processing systems* 33 (2020), pp. 15920–15930.

[30] Pietro Buzzega et al. “Rethinking Experience Replay: a Bag of Tricks for Continual Learning”. In: *2020 25th International Conference on Pattern Recognition (ICPR)* (2020), pp. 2180–2187.

[31] Lucas Caccia et al. “Online learned continual compression with adaptive quantization modules”. In: *International conference on machine learning*. PMLR. 2020, pp. 1240–1250.

[32] Hao Cai et al. “A 28nm 2Mb STT-MRAM Computing-in-Memory Macro with a Refined Bit-Cell and 22.4 - 41.5TOPS/W for AI Inference”. In: *2023 IEEE International Solid- State Circuits Conference (ISSCC)*. 2023, pp. 500–502.

[33] Olivier Cappé and Eric Moulines. “On-line expectation–maximization algorithm for latent data models”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 71.3 (2009), pp. 593–613.

[34] Mathilde Caron et al. “Unsupervised Learning of Visual Features by Contrasting Cluster Assignments”. In: (2020).

[35] Francisco M Castro et al. “End-to-end incremental learning”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 233–248.

[36] Sungmin Cha et al. “Rebalancing batch normalization for exemplar-based class-incremental learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 20127–20136.

[37] Arslan Chaudhry et al. *On Tiny Episodic Memories in Continual Learning*. 2019.

[38] Arslan Chaudhry et al. “Riemannian walk for incremental learning: Understanding forgetting and intransigence”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 532–547.

[39] Tianlong Chen et al. “"BNN - BN = ?": Training Binary Neural Networks Without Batch Normalization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2021, pp. 4619–4629.

[40] Yu-Hsin Chen et al. “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks”. In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138.

[41] Krishna Teja Chitty-Venkata and Arun K Somani. “Neural architecture search survey: A hardware perspective”. In: *ACM Computing Surveys* 55.4 (2022), pp. 1–36.

[42] Karam Cho, Akul Malhotra, and Sumeet Kumar Gupta. “XNOR-VSH: A Valley-Spin Hall Effect-based Compact and Energy-Efficient Synaptic Crossbar Array for Binary Neural Networks”. In: *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits (JXCDC)* (2023), pp. 1–1.

[43] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[44] Adam Coates, Andrew Ng, and Honglak Lee. “An analysis of single-layer networks in unsupervised feature learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 215–223.

[45] Adam Coates et al. “Deep learning with COTS HPC systems”. In: *International conference on machine learning*. PMLR. 2013, pp. 1337–1345.

[46] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “Binaryconnect: Training deep neural networks with binary weights during propagations”. In: *Advances in neural information processing systems* 28 (2015).

[47] Sajad Darabi et al. “Regularized binary network training”. In: *arXiv preprint arXiv:1812.11800* (2018).

[48] Waltenegus Dargie and Christian Poellabauer. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley & Sons, 2010.

[49] Matthias De Lange et al. "A continual learning survey: Defying forgetting in classification tasks". In: *IEEE transactions on pattern analysis and machine intelligence* 44.7 (2021), pp. 3366–3385.

[50] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[51] Ruizhou Ding et al. "Regularizing activation distribution for training binarized deep networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 11408–11417.

[52] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[53] Arthur Douillard et al. "Dytox: Transformers for continual learning with dynamic token expansion". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 9285–9295.

[54] Timothy J Draelos et al. "Neurogenesis deep learning: Extending deep networks to accommodate new classes". In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 526–533.

[55] Robert Dürichen et al. "Binary Input Layer: Training of CNN models with binary input data". In: *arXiv preprint arXiv:1812.03410* (2018).

[56] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural architecture search: A survey". In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21.

[57] Steve K Esser et al. "Backpropagation for energy-efficient neuromorphic computing". In: *Advances in neural information processing systems* 28 (2015).

[58] Chrisantha Fernando et al. "Pathnet: Evolution channels gradient descent in super neural networks". In: *arXiv preprint arXiv:1701.08734* (2017).

[59] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks". In: *International conference on machine learning*. PMLR. 2017, pp. 1126–1135.

[60] Robert M French. "Catastrophic forgetting in connectionist networks". In: *Trends in cognitive sciences* 3.4 (1999), pp. 128–135.

[61] Robert M French. "Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks". In: *Proceedings of the 13th annual cognitive science society conference*. Vol. 1. 1991, pp. 173–178.

[62] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations". In: *ArXiv* abs/1803.07728 (2018).

[63] Adria Gimenez, Jesús Andrés-Ferrer, and Alfons Juan. "Discriminative Bernoulli HMMs for isolated handwritten word recognition". In: *Pattern Recognition Letters* 35 (2014), pp. 157–168.

[64] Ian Goodfellow. *Deep learning*. 2016.

[65] Dipam Goswami et al. “Fecam: Exploiting the heterogeneity of class distributions in exemplar-free continual learning”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 6582–6595.

[66] Jianping Gou et al. “Knowledge distillation: A survey”. In: *International Journal of Computer Vision* 129.6 (2021), pp. 1789–1819.

[67] Jayavardhana Gubbi et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. In: *Future generation computer systems* 29.7 (2013), pp. 1645–1660.

[68] William Guicquero et al. “SmartNMC: A 1Mb-200µW-20fps near-imager spatio-temporal inference hardware module”. In: *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2025, pp. 1–5.

[69] N Guo et al. “Join the high accuracy club on ImageNet with a binary neural network ticket. arXiv 2022”. In: *arXiv preprint arXiv:2211.12933* (2022).

[70] Raia Hadsell et al. “Embracing change: Continual learning in deep neural networks”. In: *Trends in cognitive sciences* 24.12 (2020), pp. 1028–1040.

[71] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding”. In: *arXiv preprint arXiv:1510.00149* (2015).

[72] Tyler L. Hayes and Christopher Kanan. “Online Continual Learning for Embedded Devices”. In: *Conference on Lifelong Learning Agents (CoLLAs)*. 2022.

[73] Tyler L Hayes et al. “Replay in deep learning: Current approaches and missing biological elements”. In: *Neural computation* 33.11 (2021), pp. 2908–2950.

[74] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[75] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.

[76] Yefei He et al. “Bivit: Extremely compressed binary vision transformers”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 5651–5663.

[77] Koen Helwegen et al. “Latent weights do not exist: Rethinking binarized neural network optimization”. In: *Advances in neural information processing systems* 32 (2019).

[78] Geoffrey Hinton. “The forward-forward algorithm: Some preliminary investigations”. In: *arXiv preprint arXiv:2212.13345* (2022).

[79] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* (2015).

[80] Saihui Hou et al. “Learning a unified classifier incrementally via rebalancing”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 831–839.

[81] Andrew G Howard. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).

[82] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[83] Yuhuang Hu, Tobi Delbruck, and Shih-Chii Liu. "Incremental learning meets reduced precision networks". In: *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2019, pp. 1–5.

[84] Baichuan Huang and Amir Aminifar. "Binary Forward-Only Algorithms". In: *IEEE Design & Test* (2025).

[85] Itay Hubara et al. "Binarized neural networks". In: *Advances in Neural Information Processing Systems (NIPS)* 29 (2016).

[86] Rakib Hyder et al. "Incremental task learning with incremental rank updates". In: *European Conference on Computer Vision*. Springer. 2022, pp. 566–582.

[87] Dmitry Ignatov and Andrey Ignatov. "Controlling information capacity of binary neural network". In: *Pattern Recognition Letters* 138 (2020), pp. 276–281.

[88] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. Vol. 37. 2015, pp. 448–456.

[89] Ahmet Iscen et al. "Memory-efficient incremental learning through feature adaptation". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*. Springer. 2020, pp. 699–715.

[90] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2704–2713.

[91] Khurram Javed and Martha White. "Meta-learning representations for continual learning". In: *Advances in neural information processing systems* 32 (2019).

[92] Jiang. *Keras-CIFAR10*. URL: https://github.com/jerett/Keras-CIFAR10.

[93] Hyundong Jin and Eunwoo Kim. "Helpful or harmful: Inter-task association in continual learning". In: *European Conference on Computer Vision*. Springer. 2022, pp. 519–535.

[94] Alfons Juan, José García-Hernández, and Enrique Vidal. "EM initialisation for Bernoulli mixture learning". In: *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer. 2004, pp. 635–643.

[95] Geethan Karunaratne et al. "In-memory realization of in-situ few-shot continual learning with a dynamically evolving explicit memory". In: *ESSCIRC 2022-IEEE 48th European Solid State Circuits Conference (ESSCIRC)*. IEEE. 2022, pp. 105–108.

[96] Ronald Kemker and Christopher Kanan. "Fearnet: Brain-inspired model for incremental learning". In: *arXiv preprint arXiv:1711.10563* (2017).

[97] Latif U Khan et al. "Edge-computing-enabled smart cities: A comprehensive survey". In: *IEEE Internet of Things journal* 7.10 (2020), pp. 10200–10232.

[98] Dahyun Kim, Kunal Pratap Singh, and Jonghyun Choi. "Learning architectures for binary networks". In: *European conference on computer vision*. Springer. 2020, pp. 575–591.

[99] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the national academy of sciences* 114.13 (2017), pp. 3521–3526.

[100] Lukasz Korycki and Bartosz Krawczyk. "Class-Incremental Mixture of Gaussians for Deep Continual Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4097–4106.

[101] Olga Krestinskaya et al. "Neural architecture search for in-memory computing-based deep learning accelerators". In: *Nature Reviews Electrical Engineering* 1.6 (2024), pp. 374–390.

[102] Raghuraman Krishnamoorthi. "Quantizing deep convolutional networks for efficient inference: A whitepaper". In: *arXiv preprint arXiv:1806.08342* (2018).

[103] Alex Krizhevsky and Geoffrey Hinton. *Learning multiple layers of features from tiny images*. Tech. rep. 0. Toronto, Ontario: University of Toronto, 2009.

[104] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[105] Uday Kulkarni et al. "A survey on quantization methods for optimization of deep neural networks". In: *2022 international conference on automation, computing and renewable systems (ICACRS)*. IEEE. 2022, pp. 827–834.

[106] Dharshan Kumaran, Demis Hassabis, and James L McClelland. "What learning systems do intelligent agents need? Complementary learning systems theory updated". In: *Trends in cognitive sciences* 20.7 (2016), pp. 512–534.

[107] Axel Laborieux et al. "Synaptic metaplasticity in binarized neural networks". In: *Nature communications* 12.1 (2021), p. 2549.

[108] Jérémie Laydevant et al. "Training dynamical binary neural networks with equilibrium propagation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 4640–4649.

[109] P.F. Lazarsfeld and N.W. Henry. *Latent Structure Analysis*. Houghton, Mifflin, 1968. ISBN: 9780395047682.

[110] Phuoc-Hoan Charles Le and Xinlin Li. "BinaryViT: Pushing Binary Vision Transformers Towards Convolutional Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2023, pp. 4664–4673.

[111] Ya Le and Xuan Yang. "Tiny imagenet visual recognition challenge". In: *CS 231N* 7.7 (2015), p. 3.

[112] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[113] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[114] Yann LeCun et al. “Handwritten digit recognition with a back-propagation network”. In: *Advances in neural information processing systems* 2 (1989).

[115] Eunhae Lee. “The impact of model size on catastrophic forgetting in Online Continual Learning”. In: *arXiv preprint arXiv:2407.00176* (2024).

[116] Jay Lee, Behrad Bagheri, and Hung-An Kao. “A cyber-physical systems architecture for industry 4.0-based manufacturing systems”. In: *Manufacturing letters* 3 (2015), pp. 18–23.

[117] Sam Leroux et al. “Training binary neural networks with knowledge transfer”. In: *Neurocomputing* 396 (2020), pp. 534–541.

[118] Timothée Lesort et al. “Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges”. In: *Information fusion* 58 (2020), pp. 52–68.

[119] Cheng Li et al. “Conditional bernoulli mixtures for multi-label classification”. In: *International conference on machine learning*. PMLR. 2016, pp. 2482–2491.

[120] Zhizhong Li and Derek Hoiem. “Learning without forgetting”. In: *IEEE transactions on pattern analysis and machine intelligence* 40.12 (2017), pp. 2935–2947.

[121] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[122] Xiaofan Lin, Cong Zhao, and Wei Pan. “Towards accurate binary convolutional neural network”. In: *Advances in neural information processing systems* 30 (2017).

[123] Yujun Lin, Mengtian Yang, and Song Han. “NAAS: Neural Accelerator Architecture Search”. In: *2021 58th ACM/ESDA/IEEE Design Automation Conference (DAC)*. 2021.

[124] Zechun Liu et al. “Bi-Real Net: Binarizing Deep Network Towards Real-Network Performance”. In: *International Journal of Computer Vision* 128 (2018), pp. 202 –219.

[125] Zechun Liu et al. “How do adam and training strategies help bnns optimization”. In: *International conference on machine learning*. PMLR. 2021, pp. 6936–6946.

[126] Zechun Liu et al. “Reactnet: Towards precise binary neural network with generalized activation functions”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer. 2020, pp. 143–159.

[127] Gabriel Loaiza-Ganem and John P Cunningham. “The continuous Bernoulli: fixing a pervasive error in variational autoencoders”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[128] Vincenzo Lomonaco and Davide Maltoni. “Core50: a new dataset and benchmark for continuous object recognition”. In: *Conference on robot learning*. PMLR. 2017, pp. 17–26.

[129] Vincenzo Lomonaco, Davide Maltoni, Lorenzo Pellegrini, et al. “Rehearsal-Free Continual Learning over Small Non-IID Batches.” In: *CVPR Workshops*. Vol. 1. 2. 2020, p. 3.

[130] David Lopez-Paz and Marc'Aurelio Ranzato. "Gradient episodic memory for continual learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Curran Associates Inc., 2017, 6470–6479. ISBN: 9781510860964.

[131] Ningning Ma et al. "Shufflenet v2: Practical guidelines for efficient cnn architecture design". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 116–131.

[132] Ruichen Ma et al. "A&B BNN: Add&Bit-Operation-Only Hardware-Friendly Binary Neural Network". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 5704–5713.

[133] Shuming Ma et al. "The era of 1-bit llms: All large language models are in 1.58 bits". In: *arXiv preprint arXiv:2402.17764* 1 (2024).

[134] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of machine learning research* 9.11 (2008).

[135] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. "Piggyback: Adapting a single network to multiple tasks by learning to mask weights". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 67–82.

[136] Arun Mallya and Svetlana Lazebnik. "Packnet: Adding multiple tasks to a single network by iterative pruning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2018, pp. 7765–7773.

[137] Brais Martinez et al. "Training binary neural networks with real-to-binary convolutions". In: *arXiv preprint arXiv:2003.11535* (2020).

[138] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." In: *Psychological review* 102.3 (1995), p. 419.

[139] James L McClelland and David E Rumelhart. "Distributed memory and the representation of general and specific information". In: *Connectionist Psychology*. Psychology Press, 2020, pp. 75–106.

[140] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[141] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[142] Adnan Mehonic and Anthony J Kenyon. "Brain-inspired computing needs a master plan". In: *Nature* 604.7905 (2022), pp. 255–260.

[143] Thomas Mensink et al. "Metric learning for large scale image classification: Generalizing to new classes at near-zero cost". In: *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II 12*. Springer. 2012, pp. 488–501.

[144] Marvin Minsky and Seymour Papert. "An introduction to computational geometry". In: *Cambridge tiass., HIT* 479.480 (1969), p. 104.

[145] Asit Mishra and Debbie Marr. "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy". In: *arXiv preprint arXiv:1711.05852* (2017).

[146] Asit Mishra et al. "WRPN: Wide reduced-precision networks". In: *International Conference on Learning Representations* (2018).

[147] Sudhanshu Mittal, Silvio Galesso, and Thomas Brox. "Essentials for Class Incremental Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2021, pp. 3513–3522.

[148] Tamador Mohaidat and Kasem Khalil. "A survey on neural network hardware accelerators". In: *IEEE Transactions on Artificial Intelligence* (2024).

[149] Mehrdad Morsali et al. "XOR-CiM: An Efficient Computing-in-SOT-MRAM Design for Binary Neural Network Acceleration". In: *2023 24th International Symposium on Quality Electronic Design (ISQED)*. 2023, pp. 1–5.

[150] Martin Mundt et al. "Unified probabilistic deep continual learning through generative replay and open set recognition". In: *Journal of Imaging* 8.4 (2022), p. 93.

[151] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[152] Amir Najafi, Abolfazl Motahari, and Hamid R. Rabiee. *Reliable Clustering of Bernoulli Mixture Models*. 2019.

[153] Van Thien Nguyen, William Guicquero, and Gilles Sicard. "A 1Mb Mixed-Precision Quantized Encoder for Image Classification and Patch-Based Compression". In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.8 (2022), pp. 5581–5594.

[154] Van Thien Nguyen, William Guicquero, and Gilles Sicard. "Histogram-Equalized Quantization for logic-gated Residual Neural Networks". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. May 2022, pp. 1289–1293. (Visited on 08/09/2024).

[155] Frank Nielsen. *Introduction to HPC with MPI for Data Science*. Springer, 2016.

[156] Arild Nøkland. "Direct feedback alignment provides learning in deep neural networks". In: *Advances in neural information processing systems* 29 (2016).

[157] Hyunmyung Oh et al. "Energy-Efficient In-Memory Binary Neural Network Accelerator Design Based on 8T2C SRAM Cell". In: *IEEE Solid-State Circuits Letters (SSCL)* 5 (2022), pp. 70–73.

[158] Francesco Paissan, Alberto Ancilotto, and Elisabetta Farella. "Phinets: A scalable backbone for low-power ai at the edge". In: *ACM Transactions on Embedded Computing Systems* 21.5 (2022), pp. 1–18.

[159] Alexandros Pantelopoulos and Nikolaos G Bourbakis. "Prognosis—a wearable health-monitoring system for people at risk: Methodology and modeling". In: *IEEE Transactions on Information Technology in Biomedicine* 14.3 (2010), pp. 613–621.

[160] German I Parisi et al. "Continual lifelong learning with neural networks: A review". In: *Neural networks* 113 (2019), pp. 54–71.

[161] Lorenzo Pellegrini et al. “Latent replay for real-time continual learning”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10203–10209.

[162] Bogdan Penkovsky et al. “In-memory resistive ram implementation of binarized neural networks for medical applications”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2020, pp. 690–695.

[163] Felix Petersen et al. “Convolutional differentiable logic gate networks”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 121185–121203.

[164] Grégoire Petit et al. “Fetril: Feature translation for exemplar-free class-incremental learning”. In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2023, pp. 3911–3920.

[165] Benedikt Pfülb and Alexander Gepperth. “Overcoming catastrophic forgetting with Gaussian mixture replay”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2021, pp. 1–9.

[166] Quang Pham, Chenghao Liu, and Steven Hoi. “DualNet: Continual Learning, Fast and Slow”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[167] Quang Pham et al. “Contextual transformation networks for online continual learning”. In: *International Conference on Learning Representations*. 2021.

[168] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. “Gdumb: A simple approach that questions our progress in continual learning”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 524–540.

[169] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, 2002, pp. 55–69.

[170] Haotong Qin et al. “Binary neural networks: A survey”. In: *Pattern Recognition* 105 (2020), p. 107281.

[171] Jathushan Rajasegaran et al. “Random path selection for continual learning”. In: *Advances in neural information processing systems* 32 (2019).

[172] Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. “Effect of scale on catastrophic forgetting in neural networks”. In: *International conference on learning representations*. 2021.

[173] Hubert Ramsauer et al. “Hopfield networks is all you need”. In: *arXiv preprint:2008.02217* (2020).

[174] Mohammad Rastegari et al. “Xnor-net: Imagenet classification using binary convolutional neural networks”. In: *European conference on computer vision*. Springer. 2016, pp. 525–542.

[175] Roger Ratcliff. “Connectionist models of recognition memory: constraints imposed by learning and forgetting functions.” In: *Psychological review* 97 2 (1990), pp. 285–308.

[176] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. “Energy efficiency in wireless sensor networks: A top-down survey”. In: *Computer networks* 67 (2014), pp. 104–122.

[177] Leonardo Ravaglia et al. "A TinyML Platform for On-Device Continual Learning With Quantized Latent Replays". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.4 (2021), pp. 789–802.

[178] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning". In: *International conference on learning representations*. 2017.

[179] Sylvestre-Alvise Rebuffi et al. "icarl: Incremental classifier and representation learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[180] Mengye Ren et al. "Wandering within a world: Online contextualized few-shot learning". In: (2021).

[181] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[182] T-YLPG Ross and GKHP Dollár. "Focal loss for dense object detection". In: *proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2980–2988.

[183] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[184] Andrei A Rusu et al. "Progressive neural networks". In: *arXiv preprint arXiv:1606.04671* (2016).

[185] Grzegorz Rypeść et al. "Divide and not forget: Ensemble of selectively trained experts in continual learning". In: *arXiv preprint arXiv:2401.10191* (2024).

[186] Mehreen Saeed, Kashif Javed, and Haroon Atique Babri. "Machine learning using Bernoulli mixture models: Clustering, rule extraction and dimensionality reduction". In: *Neurocomputing* 119 (2013), pp. 366–374.

[187] Charbel Sakr et al. "True gradient-based training of deep binary activated neural networks via continuous binarization". In: *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2018, pp. 2346–2350.

[188] Eyyüb Sari, Mouloud Belbahri, and Vahid Partovi Nia. "How does batch normalization help binary training?" In: *arXiv preprint arXiv:1909.09139* (2019).

[189] Ratshih Sayed et al. "A systematic literature review on binary neural networks". In: *IEEE Access* 11 (2023), pp. 27546–27578.

[190] Abu Sebastian et al. "Memory devices and applications for in-memory computing". In: *Nature nanotechnology* 15.7 (2020), pp. 529–544.

[191] Lukas Sekanina. "Neural architecture search and hardware accelerator co-search: A survey". In: *IEEE access* 9 (2021), pp. 151337–151362.

[192] Joan Serra et al. "Overcoming catastrophic forgetting with hard attention to the task". In: *International conference on machine learning*. PMLR. 2018, pp. 4548–4557.

[193] Taha Shahroodi et al. "Lightspeed Binary Neural Networks using Optical Phase-Change Materials". In: *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2023, pp. 1–2.

[194] Zhiqiang Shen et al. "S2-bnn: Bridging the gap between self-supervised real and 1-bit neural networks via guided distribution calibration". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 2165–2174.

[195] Weisong Shi et al. "Edge computing: Vision and challenges". In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646.

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

[197] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[198] Miguel Solinas et al. "On the Beneficial Effects of Reinjections for Continual Learning". In: *SN Comput. Sci.* 4.1 (2023), p. 37.

[199] Daniel Soudry, Itay Hubara, and Ron Meir. "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights". In: *Advances in neural information processing systems* 27 (2014).

[200] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[201] Thomas Strypsteen and Alexander Bertrand. "Bandwidth-efficient distributed neural network architectures with application to neuro-sensor networks". In: *IEEE Journal of Biomedical and Health Informatics* 27.2 (2022), pp. 933–943.

[202] Febin P. Sunny et al. "ROBIN: A Robust Optical Binary Neural Network Accelerator". In: *ACM Transactions on Embedded Computing Systems (TECS)* 20.5s (2021).

[203] Wei Tang, Gang Hua, and Liang Wang. "How to train a compact binary neural network with high accuracy?" In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[204] Xiaoyu Tao et al. "Few-shot class-incremental learning". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 12183–12192.

[205] Timothy J Teyler and Jerry W Rudy. "The hippocampal indexing theory and episodic memory: updating the index". In: *Hippocampus* 17.12 (2007), pp. 1158–1169.

[206] Matteo Tremonti et al. "An empirical evaluation of tinyML architectures for Class-Incremental Continual Learning". In: *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*. IEEE. 2024, pp. 690–695.

[207] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[208] Sairam Sri Vatsavai, Venkata Sai Praneeth Karempudi, and Ishan Thakkar. "An optical xnor-bitcount based accelerator for efficient inference of binary neural networks". In: *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE. 2023, pp. 1–8.

[209] Gido M Van de Ven, Hava T Siegelmann, and Andreas S Tolias. "Brain-inspired replay for continual learning with artificial neural networks". In: *Nature communications* 11.1 (2020), p. 4069.

[210] Elisa Vianello et al. “Unified Ferroelectric/Memristive Memory for Neural Network Inference and Training”. In: (2024).

[211] Lorenzo Vorabbi, Davide Maltoni, and Stefano Santi. “Input Layer Binarization with Bit-Plane Encoding”. In: *International Conference on Artificial Neural Networks*. Springer. 2023, pp. 395–406.

[212] Lorenzo Vorabbi, Davide Maltoni, and Stefano Santi. “On-Device Learning with Binary Neural Networks”. In: *Image Analysis and Processing - ICIAP 2023 Workshops: Udine, Italy, September 11–15, 2023, Proceedings, Part I*. Springer-Verlag, 2024, 39–50.

[213] Lorenzo Vorabbi et al. “Enabling On-Device Continual Learning with Binary Neural Networks and Latent Replay:” in: *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. Rome, Italy: SCITEPRESS, 2024, pp. 25–36. ISBN: 978-989-758-679-8. (Visited on 08/09/2024).

[214] Lorenzo Vorabbi. et al. “Enabling On-Device Continual Learning with Binary Neural Networks and Latent Replay”. In: *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 2: VISAPP*. INSTICC. SciTePress, 2024, pp. 25–36.

[215] Erwei Wang et al. “Enabling binary neural network training on the edge”. In: *Proceedings of the 5th international workshop on embedded and mobile deep learning*. 2021, pp. 37–38.

[216] Fangxin Wang et al. “Deep Learning for Edge Computing Applications: A State-of-the-Art Survey”. In: *IEEE Access* 8 (2020), pp. 58322–58336.

[217] Lin Wang and Kuk-Jin Yoon. “Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks”. In: *IEEE transactions on pattern analysis and machine intelligence* 44.6 (2021), pp. 3048–3068.

[218] Liyuan Wang et al. “A comprehensive survey of continual learning: theory, method and application”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[219] Liyuan Wang et al. “Memory replay with data compression for continual learning”. In: *arXiv preprint arXiv:2202.06592* (2022).

[220] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1995–2003.

[221] Jiaxiang Wu et al. “Quantized convolutional neural networks for mobile devices”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4820–4828.

[222] Suya Wu et al. “Deep clustering of compressed variational embeddings”. In: *arXiv preprint arXiv:1910.10341* (2019).

[223] Yue Wu et al. “Large Scale Incremental Learning”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 374–382.

[224] Yixing Xu et al. “Learning frequency domain approximation for binary neural networks”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 25553–25565.

[225] Tomoharu Yamauchi et al. “Design and Implementation of Energy-Efficient Binary Neural Networks Using Adiabatic Quantum-Flux-Parametron Logic”. In: *IEEE Transactions on Applied Superconductivity (TAS)* 33.5 (2023), pp. 1–5.

[226] Shipeng Yan, Jiangwei Xie, and Xuming He. “Der: Dynamically expandable representation for class incremental learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 3014–3023.

[227] Jingkang Yang et al. “Generalized out-of-distribution detection: A survey”. In: *International Journal of Computer Vision* 132.12 (2024), pp. 5635–5662.

[228] Lei Yang et al. “Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks”. In: *2020 57th ACM/IEEE design automation conference (DAC)*. IEEE. 2020, pp. 1–6.

[229] Lilin Yang and Wei Wu. “Multiple-instance Learning based on Bernoulli Mixture Model”. In: *Journal of Physics: Conference Series*. Vol. 1650. 3. IOP Publishing. 2020, p. 032071.

[230] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. “Designing energy-efficient convolutional neural networks using energy-aware pruning”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 5687–5695.

[231] Peng Yao et al. “Fully hardware-implemented memristor convolutional neural network”. In: *Nature* 577.7792 (2020), pp. 641–646.

[232] Chunyu Yuan and Sos S Agaian. “A comprehensive review of binary neural network”. In: *Artificial Intelligence Review* 56.11 (2023), pp. 12949–13013.

[233] Jure Zbontar et al. “Barlow twins: Self-supervised learning via redundancy reduction”. In: *International conference on machine learning*. PMLR. 2021, pp. 12310–12320.

[234] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual Learning Through Synaptic Intelligence”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, 2017, pp. 3987–3995.

[235] X. Zhang et al. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6848–6856.

[236] Xueyong Zhang and Arindam Basu. “A 915–1220 TOPS/W, 976–1301 GOPS Hybrid In-Memory Computing Based Always-On Image Processing for Neuromorphic Vision Sensors”. In: *IEEE Journal of Solid-State Circuits (JSSC)* 58.3 (2023), pp. 589–599.

[237] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. “Pokebnn: A binary pursuit of lightweight accuracy”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12475–12485.

[238] Yichi Zhang et al. “FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations”. In: *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 2021, pp. 171–182.

[239] Da-Wei Zhou et al. "Class-Incremental Learning: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024), pp. 1–20. (Visited on 08/09/2024).

[240] Baozhou Zhu, Zaid Al-Ars, and H Peter Hofstee. "Nasb: Neural architecture search for binary convolutional neural networks". In: *2020 International joint conference on neural networks (IJCNN)*. IEEE. 2020, pp. 1–8.

[241] Fei Zhu et al. "Class-incremental learning via dual augmentation". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 14306–14318.

[242] Fei Zhu et al. "Prototype augmentation and self-supervision for incremental learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 5871–5880.

[243] Kai Zhu et al. "Self-sustaining representation expansion for non-exemplar class-incremental learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 9296–9305.

[244] Shilin Zhu, Xin Dong, and Hao Su. "Binary ensemble neural network: More bits per network or more networks per bit?" In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4923–4932.

[245] Bohan Zhuang et al. "Structured binary neural networks for accurate image classification and semantic segmentation". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 413–422.