Towards On-Device Continual Learning with Binary Neural Networks in Industrial Scenarios

Lorenzo Vorabbi[a,b,*], Angelo Carraggi[a], Davide Maltoni[b], Guido Borghi[c] and Stefano Santi[a]

[a]Datalogic Labs, Via San Vitalino, Bologna, 40012, Italy
[b]Department of Computer Science and Engineering, University of Bologna, Cesena, 47521, Italy
[c]Department of Education and Humanities, University of Modena and Reggio Emilia, Reggio Emilia, 42121, Italy

ARTICLE INFO

Keywords:
On-Device Learning
Continual Learning
Latent Replay
Binary Neural Networks
Image Classification

ABSTRACT

This paper addresses the challenges of deploying deep learning models, specifically Binary Neural Networks (BNNs), on resource-constrained embedded devices within the Internet of Things context. As deep learning continues to gain traction in IoT applications, the need for efficient models that can learn continuously from incremental data streams without requiring extensive computational resources has become more pressing. We propose a solution that integrates Continual Learning with BNNs, utilizing replay memory to prevent catastrophic forgetting. Our method focuses on quantized neural networks, introducing the quantization also for the backpropagation step, significantly reducing memory and computational requirements. Furthermore, we enhance the replay memory mechanism by storing intermediate feature maps (*i.e.* latent replay) with 1-bit precision instead of raw data, enabling efficient memory usage. In addition to well-known benchmarks, we introduce the DL-Hazmat dataset, which consists of over 140k high-resolution grayscale images of 64 hazardous material symbols. Experimental results show a significant improvement in model accuracy and a substantial reduction in memory requirements, demonstrating the effectiveness of our method in enabling deep learning applications on embedded devices in real-world scenarios. Our work expands the application of Continual Learning and BNNs for efficient on-device training, offering a promising solution for IoT and other resource-constrained environments.

1. Introduction

Recently, the incorporation of Artificial Intelligence (AI), and in particular deep neural networks, within the Internet of Things (IoT) framework has gained considerable attention in the literature (Mohamed, 2020; Alshehri and Muhammad, 2020). This integration presents several challenges, since it requires, among others, the deployment of intelligent systems capable of learning even within embedded boards, *i.e.* devices with limited resources, in terms of memory storage and computational capability.

Recently, the integration between AI and IoT has been supported by several industries and technological factors.

Firstly, the advancements in microchip technology that have made cost-effective chips widely available in everyday objects, with an ever-increasing memory, computational capacity, and reduced power consumption (Chen, Zheng, Zhang, Wang, Shen and Zhang, 2020). These hardware advancements have concretely translated into the release of a variety of highly efficient embedded devices on the market that, however, often lack a powerful GPU capable of running state-of-the-art deep learning architectures.

Secondly, novel deep learning solutions focused on reduced computational load while maintaining a competitive accuracy, have been introduced in the literature. Among several proposals, one of the most promising solutions is represented by the Binary Neural Networks (BNNs) (Rastegari, Ordonez, Redmon and Farhadi, 2016; Courbariaux, Hubara, Soudry, El-Yaniv and Bengio, 2016; Qin, Gong, Liu, Bai, Song and Sebe, 2020; Liu, Shen, Savvides and Cheng, 2020b; Liu, Shen, Li, Helwegen, Huang and Cheng, 2021; Shi, Qi, Cai, Fu, Zhao, Li, Liu and Liu, 2022; Zhang, Zhang and Lew, 2022), architectures in which single unit information (bit) is used to encode activations and weights. This architectural approach can significantly reduce the computational and memory requirements, making BNNs highly efficient for deployment on resource-constrained devices such as mobile phones and embedded systems.

Another important aspect is the recent exploration of novel learning methodologies tailored for the combined use of IoT technologies and AI, such as Continual Learning (CL) (Parisi, Kemker, Part, Kanan and Wermter, 2019; Masana, Liu, Twardowski, Menta, Bagdanov and Van De Weijer, 2022). These methodologies have paved the way for techniques that enable neural networks to train continuously on incremental, and possibly size-limited, data chunks. Indeed, in contrast to the traditional Machine Learning paradigm, where models are trained on a single, large dataset, the CL approach involves neural networks learning from smaller, sequential portions of data over time. Notably, this shift reduces the storage and computational demands typically associated with training large models, making CL an attractive solution for deploying artificial networks on embedded devices with limited resources. However, a significant challenge to be tackled in Continual Learning is the phenomenon known as catastrophic forgetting (Kirkpatrick, Pascanu, Rabinowitz, Veness, Desjardins, Rusu, Milan, Quan, Ramalho, Grabska-Barwinska et al., 2017),

*Corresponding author
lorenzo.vorabbi@datalogic.com (L. Vorabbi)
ORCID(s): 0000-0002-4634-2044 (L. Vorabbi)
https://www.linkedin.com/profile/view?id=https://www.linkedin.com/in/lorenzo-vorabbi/ (L. Vorabbi)

i.e. the trend of neural networks to forget previously learned information when exposed to new data. To mitigate this issue, various counteracting mechanisms have been proposed, and one of the most effective techniques is the replay memory (Shin, Lee, Kim and Kim, 2017). Specifically, replay memory involves storing and reusing past data to prevent the model from forgetting previously learned tasks as it adapts to new ones.

Despite the considerable interest of the scientific community, there are still several challenges that persist in the application of deep learning models on devices and, specifically, the adoption of the CL strategy in combination with BNNs: indeed, methods incorporating BNNs with CL algorithms are currently lacking.

Therefore, this paper proposes a solution that effectively enables the CL paradigm on embedded devices through BNNs. Specifically, by introducing a back-propagation and input binarization algorithm, we demonstrate the feasibility of continuously tuning a Convolutional Neural Network (CNN) model, including both the classification head and convolutional layers, with minimal memory requirements and high efficiency. Besides, we focus on replay memory, integrating it as a key component of the learning process. Indeed, the use of quantized networks offers significant advantages in terms of memory efficiency, particularly in resource-constrained environments such as embedded devices. By storing the intermediate feature maps – *i.e.* the latent replay (Pellegrini, Graffieti, Lomonaco and Maltoni, 2020) – rather than the raw data in the replay memory, the system is able to maintain a compact memory footprint. This allows for the storage of essential features necessary for preventing forgetting, while further minimizing the memory requirements.

Summarizing, our work advances beyond the traditional quantization approach of BNNs commonly described in the literature, where binarization is typically applied only during the forward pass, and a binary model is trained using latent floating-point weights (Bannink, Hillier, Geiger, de Bruin, Overweel, Neeven and Helwegen, 2021) following the replay memory approach. The primary advances of the proposed solution are outlined as follows:

1. **Reduced replay memory**: the proposed approach leverages on latent replay, *i.e.* involves storing intermediate activations quantized to 1-bit in the replay memory, resulting in significant memory conservation. The necessary trade-offs to maintain model accuracy while decreasing memory usage are investigated.
2. **Enhanced model accuracy**: by enabling continual learning strategies on intermediate convolutional layers, in addition to the final classification head, our solution significantly is able to incrementally learn and outperforms currently available methods in the literature.
3. **Optimized binary weight quantization**: a tailored quantization strategy for binary weights is presented, which produces an outstanding 8× reduction in memory requirements.
4. **Optimized back-propagation framework**: a comprehensive backpropagation framework has been developed that accommodates various quantization levels for both the inference and backpropagation stages.

This paper further extends our initial work (Vorabbi, Maltoni, Borghi and Santi, 2024a), and includes, in addition to the previous ones, the following original contributions:

- **DL-Hazmat**: we introduce a new challening benchmark to test the proposed solution in a real-world industrial world, under the CL paradigm. Indeed, the dataset is acquired in a real operational scenario, in which a set of hazardous material symbols must be continuously classified by an embedded system. This dataset is publicly released[1] to further support future work and comparisons in the investigated setting.
- **New experimental validations**: we apply the proposed solution on a new architecture, namely RepBNN (Shi et al., 2022). Therefore, all the models are tested on several datasets, including two new ones, our DL-Hazmat and Clear-10 (Lin, Shi, Pathak and Ramanan, 2021).

The paper is structured as follows: in Section 3.1, we elucidate the latent replay mechanism and estimate the memory saved when employing it in a binary layer. Section 3.2 introduces the quantization approach adopted for both forward and backward passes. Subsequently, in Section 3.3, we detail the methodology used for quantizing gradient computation. Experimental settings, including analysis of the tested models and datasets, are reported in Section 4. A thorough experimental evaluation in Section 5 focuses on accuracy comparison with the CWR* algorithm (Section 5.1), reduction in replay memory storage requirements (Sections 5.2 and 5.3), and the efficiency of the backpropagation algorithm (Section 5.4).

2. Related Work

2.1. Deep Learning on Device

Designing efficient neural network architectures is crucial for successful deployment on embedded devices, with limited processing power, memory, and battery life.

SqueezeNet (Koonce and Koonce, 2021c) reduces the model size by over 500 times compared to AlexNet using special layers called “fire modules” that combine squeeze and expand operations. ShuffleNet (Hluchyj and Karol, 1991) introduces the shuffle operation for better interaction between convolution groups, and MobileNetV1 (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto and Adam, 2017) employs depthwise separable convolutions to minimize parameters and computational cost. MobileNetV2 (Dong, Zhou, Ruan and Li, 2020) enhances accuracy with inverted residual structures and shortcut connections, while MobileNetV3 (Koonce and Koonce, 2021b) integrates Neural

[1] The full dataset and further information can be downloaded from https://github.com/acarraggi/dl-hazmat.

Architecture Search, HardSwish activation, and squeeze-and-excitation blocks. EfficientNet (Koonce and Koonce, 2021a) introduces compound scaling for balanced optimization of width, depth, and resolution. Additionally, a posteriori optimization techniques like pruning and quantization, analyzed in the following, reduce model size and improve efficiency, though they require careful execution to avoid performance loss.

Many studies in the academic literature have focused on on-device learning tasks that aim to reduce the memory requirements of learning algorithms. Ren, Anicic and Runkler (2021) introduced a method to transfer learning on small devices by incorporating a trainable layer on top of a fixed inference model. Cai, Gan, Zhu and Han (2020) suggested freezing the model weights and solely retraining biases to diminish memory storage during the forward pass. Lin, Zhu, Chen, Wang, Gan and Han (2022) presented a sparse update approach to bypass gradient computation for less critical layers and sub-tensors.

2.2. Continual Learning and Latent Replay for on-Device training

Continual Learning (CL) (Parisi et al., 2019) is a machine learning paradigm focused on enabling models to learn from a continuous stream of data, adapting over time without forgetting previously acquired knowledge. This contrasts with the traditional machine learning setting, where models are typically trained on a fixed, static dataset. CL is particularly valuable in real-world applications, ranging from autonomous systems (Shaheen, Hanif, Hasan and Shafique, 2022) to predictive maintenance (Ding, Qin, Wang, Cheng and Jia, 2023; Ren, Qin, Li, Wang, Yi and Jia, 2024), where data arrives incrementally, and the model needs to adapt to new information without retraining from scratch.

CL strategies can be categorized into three main approaches: architectural, regularization, and rehearsal. Architectural strategies use specific architectures or layers to combat forgetting, such as PNNs (Rusu, Rabinowitz, Desjardins, Soyer, Kirkpatrick, Kavukcuoglu, Pascanu and Hadsell, 2016) and CWR (Maltoni and Lomonaco, 2019). CWR protects the classification head by maintaining temporary and consolidated weights; more details are reported in the following section. Regularization strategies use techniques like weight sparsification or loss functions to prevent overwriting of important weights. Rehearsal strategies, like iCaRL (Rebuffi, Kolesnikov, Sperl and Lampert, 2017), and A-GEM (Chaudhry, Ranzato, Rohrbach and Elhoseiny, 2018), store a subset of past data to replay to the model, usually referred to as replay memory, often combined with other hybrid techniques.

Latent replay (Pellegrini et al., 2020) is a promising technique for overcoming catastrophic forgetting in continual learning. It involves storing and replaying activations from intermediate layers of a neural network instead of raw data. This approach reduces computational cost and storage requirements while potentially enhancing privacy and data efficiency. Choosing the appropriate latent replay layer is crucial, with deeper layers offering more discriminative features but requiring more resources. The latent replay can be effectively combined with existing continual learning techniques like regularization and dynamic architectures. The effectiveness of latent replay is influenced by the choice of the pre-trained model and its relationship to the downstream continual learning tasks. Ostapenko *et al.* (Ostapenko, Lesort, Rodriguez, Arefin, Douillard, Rish and Charlin, 2022) highlighted that pre-trained models on broader datasets tend to produce better representations for continual learning. In particular, a non-parametric classifier trained on latent representations could achieve comparable performance to more complex models with significantly lower computational costs.

Generative replay (Shin et al., 2017) offers a potential solution to the memory limitations of traditional replay methods in continual learning, particularly for embedded devices. Indeed, instead of storing past data, generative models learn to synthesize examples that mimic previously encountered data distributions. This eliminates the need for a large memory buffer, making it attractive for resource-constrained devices. However, generative replay introduces complexities in training the generative model itself, and the quality of the generated data can significantly impact the performance of the continual learning system (Liu, Wu, Menta, Herranz, Raducanu, Bagdanov, Jui and de Weijer, 2020a). If the generative model fails to accurately capture the complexities of past data distributions, the synthesized examples might not be representative enough to effectively prevent catastrophic forgetting. Additionally, the training phase, which relies on on-the-fly generated examples, not only demands hardware resources for the training procedure itself but also for performing inference on the generative model. This inference step often carries a significant computational burden, further emphasizing the need for efficient resource management and optimization in such scenarios.

Some works in the literature investigate the combined use of Continual Learning and model optimization.

QLR-CL, introduced in (Ravaglia, Rusci, Nadalini, Capotondi, Conti and Benini, 2021), utilizes 8-bit quantization to accelerate network execution up to the latent layer while reducing the memory demand of latent replay vectors from 32-bit floating point to 8-bit. In comparison with our work, QLR-CL optimizes the computation pipeline for a specific ultra-low-power CPU based on the RISC-V ISA and employs floating-point precision for backpropagation. In the works of Nadalini et al. (Nadalini, Rusci, Tagliavini, Ravaglia, Benini and Conti, 2022; Nadalini, Rusci, Benini and Conti, 2023) the authors developed a framework for on-device learning on small devices using floating-point computation of 32 and 16 bits.

Our solution deviates significantly in this context since we introduce a quantized fixed-point implementation for both binary and non-binary layers instead of post-training quantization of frozen layers followed by backpropagation using floating-point precision. Notably, our work represents the first endeavor, to the best of the author's knowledge, in

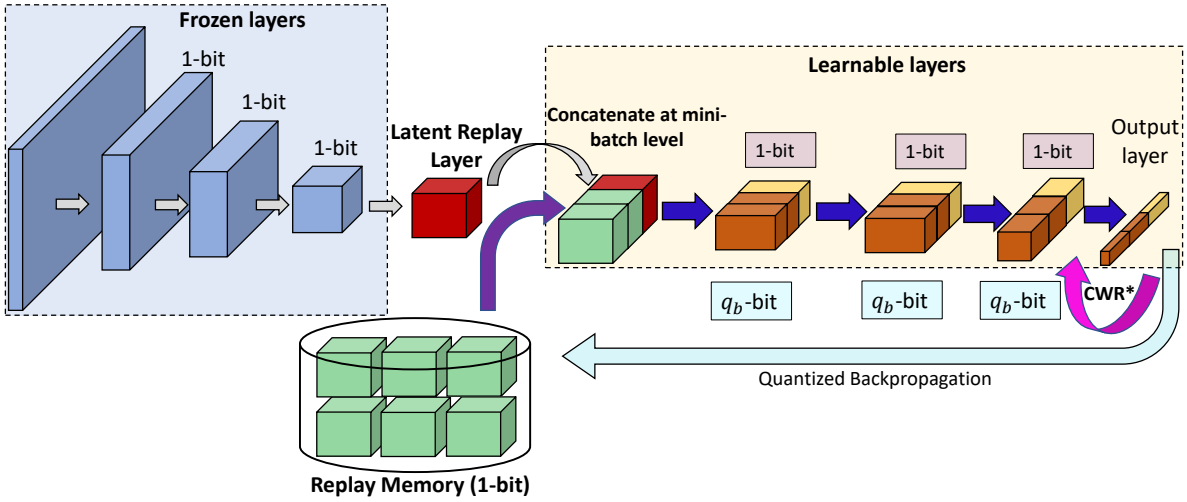


Figure 1: Visual representation of the Continual Learning paradigm with Latent Replay memory. It is worth noting that, if the deep learning architecture is a Binary Neural Network, the activations stored in the replay memory can be quantized to 1-bit, resulting in significant memory reduction with respect to a standard CNN. Image source: Vorabbi et al. (2024a)

implementing on-device learning by quantizing the back-propagation of binary layers using low bitwidths.

2.3. Binary Neural Networks

Quantization is a technique used to compress neural network models by reducing the precision of their weights and activations, which significantly reduces their memory footprint. By mapping continuous values to a smaller set of discrete values, the bit-width for storage and computation is lowered. This approach is particularly useful for deploying models on resource-constrained devices. However, quantization can lead to accuracy loss, though careful design can minimize this impact. Overall, it offers substantial benefits in terms of memory efficiency and computational cost, especially for real-time applications.

Binarization, the most extreme form of quantization, restricts data representation to just two values, typically -1 (0) or $+1$ (1). By utilizing only 1-bit for representing both weights and activations, the model's memory requirements are substantially reduced. In addition, computationally expensive matrix multiplication operations are replaced with more efficient bitwise XNOR and Bitcount operations. According to Bannink et al. (2021), the inference time of binary layers can be up to 9 to 12 times faster than that of 8-bit quantized or floating-point layers on low-power ARM CPUs, providing a significant improvement in computational efficiency.

Binary Neural Networks offer several hardware-friendly characteristics, including memory efficiency, power optimization, and significant acceleration. In certain network configurations, Binary Neural Networks can be executed on-device without the need for floating-point operations, simplifying deployment on ASIC or FPGA hardware, as highlighted in (Vorabbi, Maltoni and Santi, 2024b), which achieved a consistent speed-up compared to LCE Bannink et al. (2021).

A previous work (Vorabbi et al., 2024a) proposed novel solutions to train a BNN model on a tiny device by adapting part of the backbone with an ad-hoc quantization scheme and mitigating the catastrophic forgetting by incorporating the CWR* (Lomonaco, Maltoni and Pellegrini, 2020; Graffieti, Borghi and Maltoni, 2022) in the classification head. Additionally, the usage of latent replay memory in (Vorabbi et al., 2024a) consistently increased the model accuracy compared to similar approaches (Vorabbi, Maltoni and Santi, 2023).

3. Method

This section presents our approach for implementing CL method utilizing Latent Replay and BNNs. Specifically, we leverage the CWR* (Vorabbi et al., 2024a) approach, which addresses class-bias correction in the classification head (Masana et al., 2022). This approach is visually represented in Figure 1.

Specifically, CWR* uses a set of temporary weights tw adjusted during the back-propagation and, the consolidated weights cw, used during inference. tw are reset to 0 at the beginning of each training mini-batch while cw are updated according to formula of line 16. CWR* initializes the weights using the cw at the beginning of a training batch and, the consolidation step computes a per-class weighted sum based on the number of training samples used. For the sake of clarity, the mechanism of CWR* is detailed in Algorithm 1.

3.1. Continual Learning with Latent Replays

In Figure 1, we visually represent the Continual Learning (CL) process with Latent Replay (LR). When new data becomes available, it is input to the neural network, which generates latent activations during the forward pass. These latent activations represent the feature maps corresponding to a specific intermediate layer, denoted as l (where $l \in$

Algorithm 1: Pseudocode of CWR*.

// k = index of the classification layer
$cw_k = 0$;
// # samples for each class i analyzed
$past_i = 0$;
initialize $\overline{\Theta}$ weights ;
// B_j: mini-batch with index j
for *each training batch* B_j **do**
  expand layer k to add new classes in B_j
  $tw_k[i] = \begin{cases} cw_k[i], & \text{if class } i \text{ in } B_j \\ 0, & \text{otherwise} \end{cases}$;
  train the model using SGD optimizer;
  if $B_j = B_1$ **then**
    learn both $\overline{\Theta}$ and tw_k
  else
    learn tw_k while keeping $\overline{\Theta}$ fixed
  end
  // consolidation step
  for *each class* i *in* B_j **do**
    // cur_i = # samples of class i in B_j
    $wpast_i = \sqrt{\frac{past_i}{cur_i}}$
    $cw_k[i] = \frac{cw_k[i] \cdot wpast_i + (tw_k[i] - avg(tw_k))}{wpast_i + 1}$
    $past_i = past_i + cur_i$
  end
  evaluate model's performance with $\overline{\Theta}$ and cw_k
end

Algorithm 2: Procedure used to populate the replay memory (RM). RM is initially pre-populated using training samples of the first experience. The reservoir sampling is used on a class basis to maintain the balance among different classes. This approach prevents a skewed representation of classes within RM.

Input: N = max number of samples per class
Input: C = max number of classes
$RM_{size} = C \cdot N$;
// $C \cdot N$ is the max size of RM populated during the first experience.
for *each on-device experience* **do**
  // T is the number of classes
  // M_t = samples of class t
  // RM_t = samples of class t already in RM
  for $t = 0$ **to** $T - 1$ **do**
    $B_t = RM_t \cup M_t$;
    // # is the cardinality operator
    RM_t^{new} = apply Reservoir sampling to extract $\#RM_t$ samples from B_t;
    remove not selected RM_t samples ;
    update RM with RM_t^{new};
  end
end

$[0, L)$), with L indicating the total number of layers in the model. The activations of the new data are combined at the minibatch level with the replay activations stored from previous experiences. The forward and backward passes are executed on the remaining layers, specifically those with an index ranging from $l + 1$ to $L - 1$. To elaborate, if B_N represents the minibatch size of the newly acquired latent activations, a subset of replay vectors (B_R) is retrieved from the replay memory and merged to create a minibatch with a total size of $B_T = B_N + B_R$. On the other hand, layers with an index lower than l are kept frozen and excluded from the learning process. Upon completion of each training experience, the replay memory is updated by incorporating samples from the most recent experience using class-balanced reservoir sampling (Vitter, 1985), ensuring a balanced distribution of samples across classes and experiences (refer to Algorithm 2).

3.2. Quantization of activations and weights

Quantization techniques have been widely embraced to decrease the data size associated with model parameters and layer activations. By employing quantization strategies, the data bitwidth is reduced from the conventional 32-bit floating-point format to a lower bit-precision representation, typically 8 bits or even fewer. This reduction in precision often results in minimal loss of accuracy during the model's forward pass. For quantizing non-binary layers that require training on-device, we have adopted the approach outlined in the work of Jacob, Kligys, Chen, Zhu, Tang, Howard, Adam and Kalenichenko (2018). This approach provides a quantization method that maintains computational efficiency and minimizes the impact on model performance when transitioning to lower bit-precision formats.

By considering the i-th layer of the network, we can define the dynamic range of the activations as $[a^i_{min}, a^i_{max}]$, and the corresponding quantized activations a_q as:

$$a^i_q = cast\_to\_q\lfloor \frac{a^i}{S^i_a} \rceil, \quad S^i_a = \frac{a^i_{max} - a^i_{min}}{2^q - 1} \tag{1}$$

where q represents the number of quantization bits used (8, 16, 32), a^i denotes the full-precision activations and a^i_{max}, a^i_{min} are computed through a calibration on the training dataset. Weights can be quantized similarly by using the equation 1. As suggested in (Vorabbi et al., 2024a), we adopt two separate quantization settings for both the forward and backward passes.

During the forward pass, binarization is computed by using the following equation:

$$STE(x) = \begin{cases} +1 & x \geq 0 \\ -1 & otherwise \end{cases} \tag{2}$$

as reported in (Courbariaux et al., 2016). During the backward stage, STE approximates the derivative of the sign

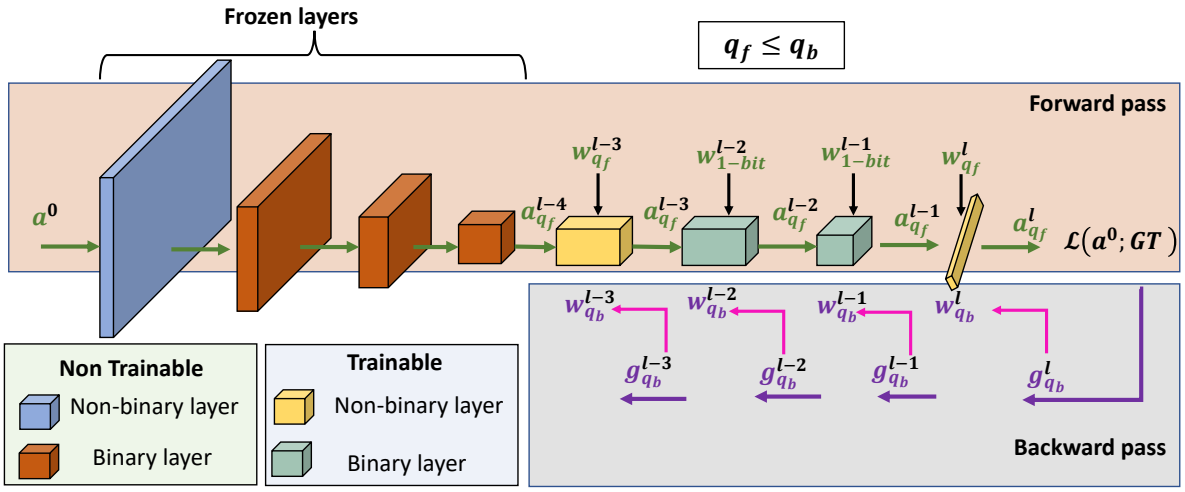


Figure 2: Quantization scheme that uses a different number of bitwidth for forward (q_f) and backward (q_b) pass. Usually, trainable non-binary layers are Batch Normalization (Ioffe and Szegedy, 2015), Addition and Concatenation layers. Image source: Vorabbi et al. (2024a)

	# total weights	LR shape	# B = binary weights after LR	# NB = non-binary weights after LR	$\frac{B}{B+NB}$	MACs
BiReal-18	$11.2M$	$(4, 4, 512)$	$7.0M$	$19K$	99.7%	$37M$
BiReal-18	$11.2M$	$(8, 8, 256)$	$10.1M$	$28K$	99.7%	$37M$
React-18	$28.3M$	$(4, 4, 512)$	$17.8M$	$45K$	99.7%	$98.5M$
React-18	$28.3M$	$(8, 8, 256)$	$21.1M$	$61K$	99.7%	$98.5M$
VGG-Small	$14M$	$(8, 8, 512)$	$7M$	$260K$	96.4%	$617M$
QuickNet	$12.7M$	$(7, 7, 256)$	$9.5M$	$36K$	99.6%	$38.4M$
QuickNet	$12.7M$	$(14, 14, 128)$	$11.9M$	$43K$	99.6%	$38.4M$
QuickNetLarge	$22.8M$	$(7, 7, 256)$	$14.2M$	$40K$	99.7%	$76.1M$
QuickNetLarge	$22.8M$	$(14, 14, 128)$	$21.3M$	$56K$	99.7%	$76.1M$
RepBNN	$23.1M$	$(2, 2, 1024)$	$22.8M$	$154K$	99.3%	$77.6M$

Table 1
The table presents a comparison of memory usage (# parameters) for different BNN models. B denotes the number of binary weights that can be updated during back-propagation, while NB represents the number of non-binary weights. The latent replay (LR) placement is discussed in Section 5. It is worth noting that the largest part of memory weights is used by binary weights. The last column reports the amount of multiply-and-accumulate operations (MACs) considering an input volume of $32 \times 32 \times 3$ for all the tested models.

function with a constant value. More recent works (Liu, Wu, Luo, Yang, Liu and Cheng, 2018; Liu et al., 2020b) further improved the gradient approximation.

3.3. Quantized Backpropagation

Gradient quantization is the primary source of accuracy degradation during the backpropagation stage, as pointed out in several works (Gupta, Agrawal, Gopalakrishnan and Narayanan, 2015; Das, Mellempudi, Mudigere, Kalamkar, Avancha, Banerjee, Sridharan, Vaidyanathan, Kaul, Georganas et al., 2018; Banner, Hubara, Hoffer and Soudry, 2018). To mitigate the poor gradient quality, we propose a similar quantization scheme to that presented in (Vorabbi et al., 2023). In such a design, two sets of bit widths are adopted for forward and backward passes as the forward path necessitates less bit precision to avoid accuracy loss (Liang, Glossner, Wang, Shi and Zhang, 2021).

The back-propagation step computes the gradients of the loss function (indicated as $\mathcal{L}$) iteratively w.r.t. the input of each layer l, indicated as a^{l-1}, by starting from the final layer, as reported below:

$$g^l = \frac{\partial \mathcal{L}}{\partial a^l} \tag{3}$$

For each layer of the model, the Equation 3 is used to propagate backward the gradient up to the input layer, as g^l is also used to calculate the gradients w.r.t. the layer weights. By analyzing the case of a linear layer, where $a^l = W^l \cdot a^{l-1}$ and $\frac{\partial a^l}{\partial a^{l-1}} = W^l$, the gradients can be computed as follows:

$$g^{l-1} = \frac{\partial \mathcal{L}}{\partial a^l} \cdot \frac{\partial a^l}{\partial a^{l-1}} = W^l g^l \tag{4}$$

The gradient with respect to the layer (of index l) weights is reported below:

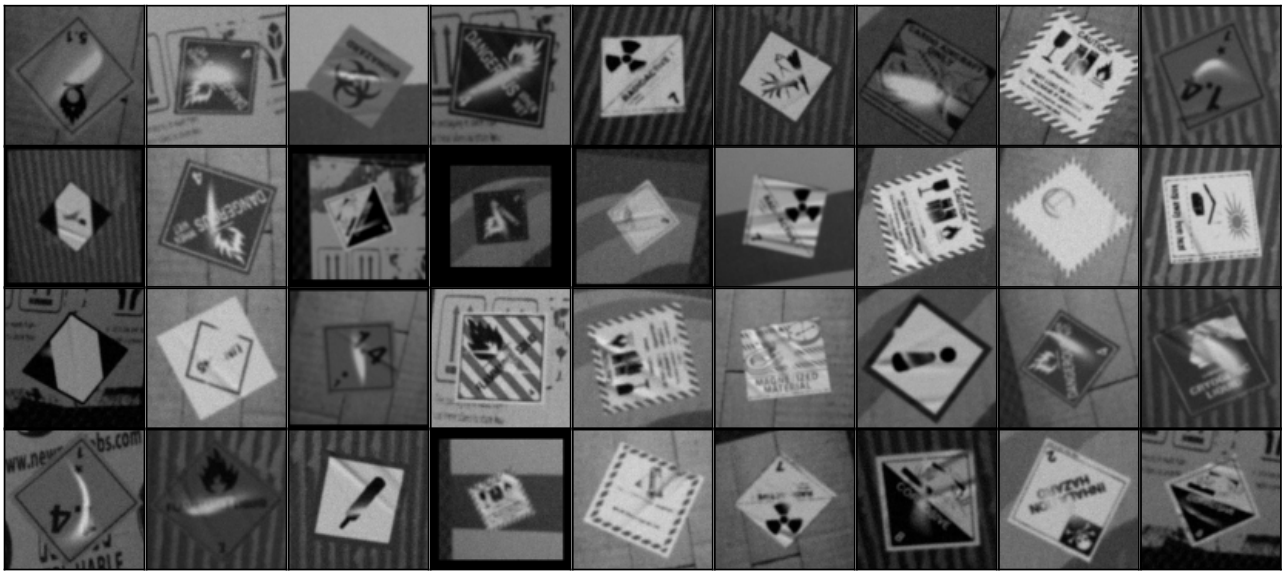


Figure 3: Example images of our *DL-Hazmat* dataset. This dataset consists of 64 Hazmat symbols, fundamental for the logistic operation in industry. Frequently, in real-world industrial scenarios, these symbols can present new symbols and/or new variations of the known one, and therefore a continuous adaptation of the classifier is necessary, as described in Section 4.1.1.

$$g_w^l = \frac{\partial \mathcal{L}}{\partial a^l} \cdot \frac{\partial a^l}{\partial W^l} = a^{l-1} g^l \tag{5}$$

The Equations 4 and 5 highlight the necessity of approximately twice the number of Multiply-And-Accumulate (MAC) operations during the backward compared to the forward pass, pushing the need for an optimized and quantized backpropagation to train a neural network model on-device efficiently. The quantization of gradients and weights (Equations 4 and 5) is implemented through the Equation 1 and it is visually represented in Figure 2. As reported in (Banner et al., 2018; Vorabbi et al., 2023), the necessity of preserving the directionality of the weight tensor usually leads to quantize the gradients using higher bitwidth and, based on this insight, we propose to adopt during forward pass a lower bitwidth (see Figure 2, q_f bits, green path) to reduce latency and more bits for the backward pass, to preserve gradient accuracy (Figure 2, q_b bits, purple path).

Given the limited memory resources typically available on embedded devices, it is essential to accurately estimate the memory requirements of the learning algorithm. Memory can be broadly divided into two categories: the memory utilized by the CL method (for example, the replay memory) and the memory needed to store intermediate tensors during the forward pass, which are later used in the backpropagation process, to compute the model weights gradients. In this context, our primary focus will be on the latter aspect, particularly in the case of binary layers where q_f is kept constant at 1 bit, while q_b may vary based on the desired level of accuracy.

Table 1 outlines the assessment of memory usage for representing binary weights of trainable layers on the device. It is pertinent to note that binary weights make up a significant portion of the total model parameters, resulting in substantial memory savings when reducing q_b to 1-bit, as demonstrated in comparison to the conventional setting of 16 bits. The decline in memory utilization shows a nearly linear correlation with the number of bits employed. Furthermore, we differentiate q_b for binary and non-binary layers to enable the application of diverse quantization bitwidths, as detailed in Section 5 of the manuscript. This section highlights that it is conceivable to uphold accuracy while notably decreasing q_b for binary layers. Referring to q_b^{bin} and $q_b^{non-bin}$ as the quantization configurations for binary and non-binary layers, correspondingly, we illustrate in Section 5 that employing a 1-bit setting for q_b^{bin} yields minimal accuracy degradation in comparison to higher quantization bitwidths.

Finally, in the last column of Table 1 we report the total amount of multiply and accumulate operations (MACs) – binary and non-binary – for each model. The MAC metric serves assesses the computational load since it reflects the total number of arithmetic operations performed during model execution, capturing the interplay between multiplication and accumulation, which are critical for tasks like matrix multiplication and convolution. These operations dominate the computational complexity in deep learning, making MAC count a direct indicator of both processing demand and energy efficiency of a model. In Section 5.2 we provide a latency comparison of a BNN network, both for forward and backward stages, with an 8-bit quantized Mobilenetv2.

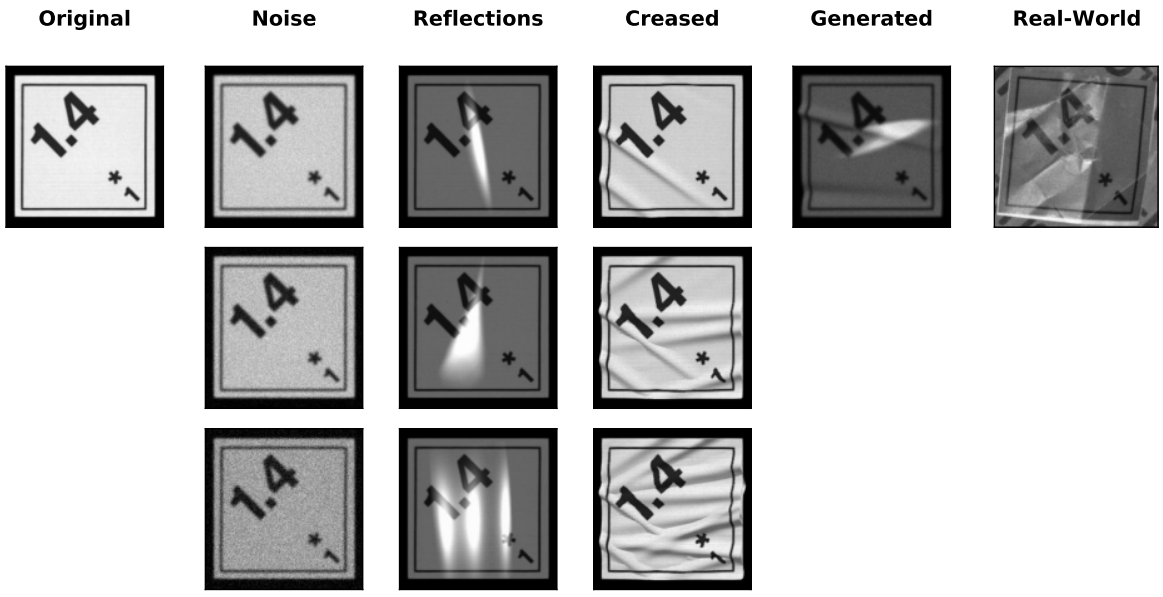


Figure 4: Example of the synthetic generation pipeline adopted for our dataset DL-Hazmat. The original image is augmented with noise, reflections, and creasing effects. In each line, the augmentation is increased in intensity from the top to the bottom. The combination of such pipelines originates the image reported in the fifth column.

4. Experimental Evaluation

4.1. Datasets

CORe50 (Lomonaco and Maltoni, 2017): this dataset is specifically designed for the continuous object recognition task and comprises 50 domestic objects categorized into 10 classes. Data has been collected over 11 distinct sessions (8 indoor and 3 outdoor), each featuring varied backgrounds and lighting conditions. For continuous learning scenarios – *i.e.* New Instances (NI) and New Classes (NC) – the test set includes sessions #3, #7, and #10, with accuracy measured after each learning experience. The remaining 8 sessions are divided into batches and presented sequentially during training, resulting in 9 experiences for the NC scenario and 8 for the NI scenario. No data augmentation procedures have been applied, as the dataset inherently includes sufficient variability in terms of brightness, flips, and rotations variations. The input RGB images are standardized and resized to $128 \times 128 \times 3$ pixels.

CIFAR-10 and CIFAR-100 (Krizhevsky, Hinton et al., 2009): the CIFAR-10 dataset comprises 60k images, each with a resolution of 32×32 pixels, distributed across 10 distinct classes. These classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. There are 50k training images and 10k test images. The CIFAR-100 dataset is similar to CIFAR-10 but contains more classes (100), each with 600 images. Each class includes 500 training images and 100 testing images. In the NC scenario for CIFAR-10, due to the smaller number of classes, there are 5 experiences, with 2 classes added per experience. For CIFAR-100, 10 experiences are utilized. For both datasets, the NI scenario consists of 10 experiences. Similar to CORe50, the test set remains constant throughout the experiences. The RGB images are scaled to the interval $[-1.0, +1.0]$, and the following data augmentation techniques are applied: zero padding of 4 pixels on each side, a random 32×32 crop, and a random horizontal flip. No augmentation is applied during testing.

Clear-10 (Lin et al., 2021): this dataset is developed from the YFCC100M image collection (Thomee, Shamma, Friedland, Elizalde, Ni, Poland, Borth and Li, 2016) using a scalable and cost-effective method based on vision-language models. These models, such as CLIP, are used to construct labeled datasets interactively, which are then validated through crowd-sourcing to eliminate errors and inappropriate images. The key advantage of the CLEAR dataset over previous CL benchmarks is the smooth temporal evolution of visual concepts using real-world imagery, combining high-quality labeled data with many unlabeled samples for continual semi-supervised learning. The dataset contains 10 classes for 10 experiences both for training and testing. Each class in the training dataset includes 300 images of various sizes while the testing dataset comprises 50 samples per class. The training dataset is composed of $30K$ samples, and $5K$ is used for testing. In the NC scenario, due to the smaller number of classes, we reduced the number of experiences to 5, with 2 classes added per experience. As usual, the test set remains constant throughout the experiences. No data augmentation procedures were applied since the dataset includes sufficient variability. The input RGB images are standardized and resized to $224 \times 224 \times 3$ pixels.

4.1.1. DL-Hazmat Dataset

To evaluate the proposed method in a realistic industrial scenario, we introduce DataLogic-Hazardous MATerial

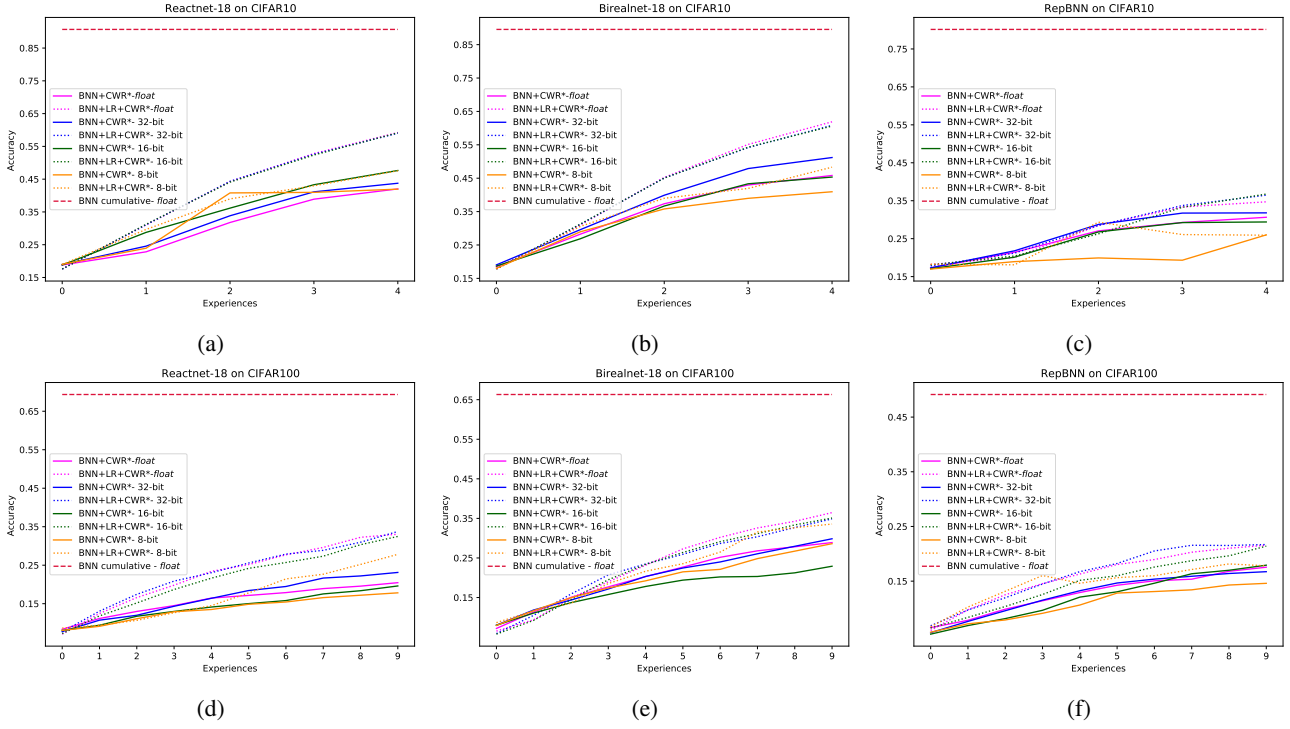


Figure 5: Accuracy comparison of our solution (BNN+LR+CWR*) with previous work BNN+CWR* (Vorabbi et al., 2023) on CIFAR-10 (first row) and CIFAR-100 (second row) datasets using different models, *i.e.* ReactNet, BirealNet and RepBNN.

(DL-Hazmat), a novel dataset specifically designed for hazard symbol detection using embedded devices and computer vision techniques. The dataset features a diverse collection of hazard symbols, subject to variation over time based on client-specific requirements. This dynamic nature requires a method that efficiently retrains and incorporates newly introduced classes directly on the device. DL-Hazmat provides a crucial benchmark for assessing the adaptability and performance of such methods in environments where frequent updates to the classification model are required. The limited accuracy achieved in tests on the DL-Hazmat dataset highlights its challenging nature, making it a particularly valuable resource for future benchmarking in the literature. This complexity underscores the need for robust and adaptive methods capable of addressing the dynamic and varied industrial scenarios represented within the dataset.

Specifically, hazardous material symbols (also known as warning or safety symbols) are visual indicators designed to convey information about potential hazards associated with specific materials, locations, or objects. In the Transportation and Logistics (T&L) market, classifying the Hazard symbols on each parcel is mandatory for the devices used for traceability. The variation of hazard symbols, even within the same class, necessitates an automatic solution that can easily accommodate new customizations. This scenario is well-suited for a continual learning approach, specifically for accommodating new classes. Practically, a deep neural network model is pre-trained on an initial subset of hazard symbols. Before deploying the model, if a customer needs to recognize new symbols, the new hazards are collected (this process can be automated using synthetic generation into Experience 1). The pre-trained model is then trained on the newly collected set of symbols using the method described in Section 3.1. A subset of the symbols used to pre-train the model is also utilized as the initial replay memory content. Whenever a new set of hazard symbols needs to be added (a new continual learning stage is triggered over a new experience), this process is repeated. *DL-Hazmat* contains over 140k high-resolution images with a spatial resolution of 1280×760 divided into 64 classes, as depicted in Figure 3. The classification task can be performed at the symbol level (64 classes) or the category level (24 classes). The former task is more challenging as the symbols of the same category are rather difficult to distinguish, as in many cases only a few small details change. Images in the database have been synthetically generated, starting from several real background textures of packages and conveyor belts, according to the pipeline of Figure 4. Images are then subdivided into 11 sessions. For each session and symbol class, 200 training grayscale images are provided. 10 sessions can be used for training/validation and 1 session as the test set. The input RGB images are standardized and resized to $128 \times 128 \times 3$ pixels.

To generate the variability of a real-world scenario, in addition to the standard effects of noise, Gaussian blur, scale, and pose, we augmented the dataset using other visual effects that frequently happen with real data such as thermal printing effect, ink degradation, specular light reflections and creasing effects. Figure 4 reports an example of such a

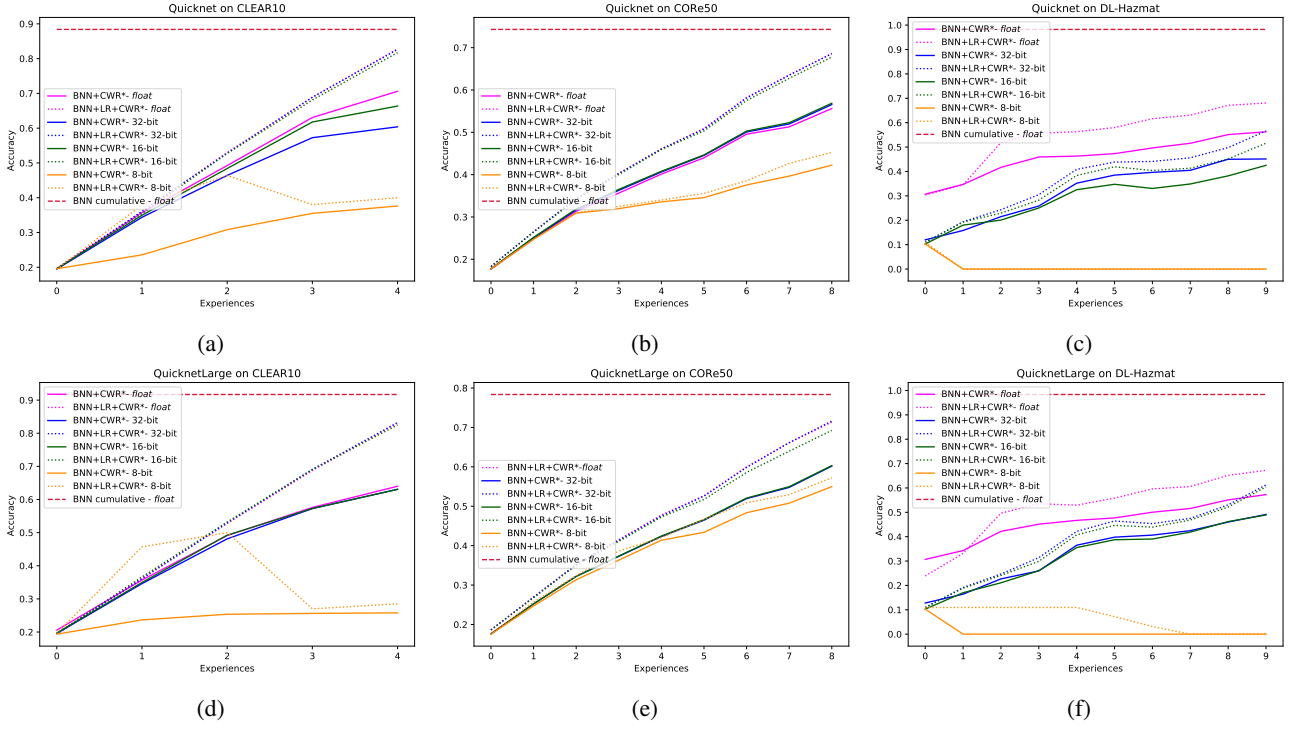


Figure 6: Accuracy evaluation of our solution (BNN+LR+CWR*) with previous work BNN+CWR* (Vorabbi et al., 2023) on several datasets *i.e.* CLEAR-10 (first column), CORe50 (second column), DL-Hazmat (third column), using QuickNet and QuickNetLarge models.

pipeline starting from an input image (Original), and comparing the final synthetic output (Generated) with an original symbol (Real-World). The noise is generated by using two different contributions. The first one samples a Gaussian distribution (kernel size $\in \{7, 11, 15\}$ and $\sigma \in [1, 10]$) to simulate the low-pass filter of the optics. The second one utilizes a Poisson distribution to generate shot noise, which describes the fluctuations of the number of photons detected in a photosensor. The reflections are generated by randomly applying, on the rendered surface, a set of pre-configured effects. The creased effect is generated by applying waves on random portions of the rendered surface. Every wave is modeled through a Gaussian distribution where the standard deviation and amplitude of the distribution determine the level of turbulence.

4.2. Models

QuickNet and QuickNetLarge (Bannink et al., 2021): these networks build on previous works (Liu et al., 2018; Bethge, Yang, Bornstein and Meinel, 2019; Martinez, Yang, Bulat and Tzimiropoulos, 2020), proposing a sequence of blocks with varying numbers of binary (3×3) convolutions and residual connections in each layer. Transition blocks between residual sections halve the spatial resolution and increase the filter count. QuicknetLarge enhances accuracy by employing more blocks and feature maps. For Quicknet, latent replay memory is set to the `quant_conv2d_16` layer, storing 1-bit activations, while for QuicknetLarge, it is set to the `quant_conv2d_30` layer. At this level, both Quicknet and QuicknetLarge have activations of dimensionality (7, 7, 256), leading to significant memory savings by storing 1-bit activations in replay memory. Contrary to the findings in (Vorabbi et al., 2023), our study did not include the *Realtobinary* model (Martinez et al., 2020), as it demonstrated significantly lower accuracy levels, which did not align with our research objectives.

BiRealNet (Liu et al., 2018): this architecture is a modified version of the classical ResNet designed to enhance the representational capability of a 1-bit CNN by preserving the real activations before the sign function through a simple shortcut. This network employs a close approximation to the derivative of the non-differentiable sign function with respect to activation and utilizes a magnitude-aware gradient for updating weight parameters. We used the 18-layer variant of this network, as detailed in the Bi-RealNet official repository[2]. The latent replay layer is set to *add_12*, where the activations have a dimensionality of (4, 4, 512). In our implementation, the model has been pre-trained on Tiny Imagenet (Le and Yang, 2015).

ReactNet (Liu et al., 2020b): to further compress compact networks, this model builds a baseline on MobileNetV1 (Howard et al., 2017) and incorporates a shortcut to bypass every 1-bit convolutional layer with matching input and output channels. The 3×3 depth-wise and 1×1 point-wise convolutional blocks of MobileNet are replaced by 3×3 and 1×1 vanilla convolutions, respectively, in parallel with shortcuts

[2] https://github.com/liuzechun/Bi-Real-net

in ReactNet (details available at the ReactNet repository[3]). Similar to Bi-RealNet, we evaluated the 18-layer version of ReactNet. The latent replay layer is set to `add_12`, where the activations have a dimensionality of (4, 4, 512) and the model has been pre-trained on Tiny Imagenet Le and Yang (2015).

RepBNN (Shi et al., 2022): this network introduces an innovative replaceable convolution, namely RepConv module that enhances feature maps. It achieves this by duplicating inputs or outputs along the channel dimensions, without incurring extra parameter costs or requiring additional convolution computations. A set of RepTran rules for implementing RepConv across BNN modules is also proposed, including binary convolution, fully connected layers, and batch normalization. In the original paper (details are available at the original repository[4]), the authors show that after applying the RepTran transformation, a variety of BNNs exhibit universally improved performance compared to their original versions. The latent replay layer is set to *learnable_bias_48*, where the activations have a dimensionality of (2, 2, 1024). The model, as for BiRealnet and Reactnet, has been pre-trained on Tiny Imagenet (Le and Yang, 2015).

4.3. Experimental Settings

In our experimental setup, we discovered that decreasing the number of epochs in each learning experience had an insignificant impact on model accuracy. Consequently, we empirically set the number of epochs to 5 to reduce training time on the on-device platform. We utilized the Cross Entropy loss function and Stochastic Gradient Descent (SGD) as the optimizer for all classification tasks. Cross Entropy was chosen due to its straightforward derivative computation with the Softmax activation function. At the same time, SGD was preferred for its computational efficiency and lower overhead compared to more complex algorithms like Adam (Kingma and Ba, 2014). In our experiments, the ratio of B_N to the batch size of latent activations sampled from replay memory was set at $\frac{1}{4}$. Both weight and activation binarization were applied during training, including the initial training experience and on-device stages. This required the implementation of a quantized backward pass technique for all non-differentiable functions, particularly the binarization functions (using Equation 1 and 4). To assess model accuracy during on-device training, we developed quantized backward steps for all layers of the previously described models. To simplify and optimize the training phase of the models utilized at the edge, we removed the double training step adopted in BiRealnet, Reactnet, and RepBNN. Additionally, we do not employ a student-teacher approach for training, as it would require a higher computational overhead and memory demand. The simplifications introduced are necessary to execute the models on tiny devices but translate into lower accuracy values than the ones reported in the original works.

[3] https://github.com/liuzechun/ReActNet

[4] https://github.com/imfinethanks/Rep_AdamBNN

Our experiments predominantly focused on the New Classes (NC) scenario. As noted in (Pellegrini et al., 2020), the use of latent replay memory did not notably improve model accuracy in the New Instances (NI) context. Furthermore, the NC scenario more accurately reflects real-world applications, where a model's ability to recognize and adapt to new, previously unseen classes is crucial.

5. Experiments

5.1. Model Classification Accuracy

To evaluate the effectiveness of our proposed solution, we began by comparing it to existing methods, specifically BNN+CWR* (Vorabbi et al., 2023), which only trains the final classification layer on-device without utilizing replay memory. Our comparison involved a series of tests with various quantization bitwidths for both forward and backward passes.

Figures 5 and 6 illustrates the accuracy comparisons between BNN+CWR* and our method, labeled **BNN+LR+CWR***, across various datasets: CIFAR10 (Figures 5a, 5b, 5c), CIFAR100 (Figures 5d, 5e and 5f), CLEAR10 (Figures 6a and 6d), CORe50 (Figures 6b and 6e) and DL-Hazmat (Figures 6c and 6f). Each figure highlights the performance improvements achieved by the new method across all tested quantization settings, which include floating-point arithmetic as well as 32-bit, 16-bit, and 8-bit quantized representations. In all plots, with the red dotted line, we report the results for cumulative training of the model with float precision.

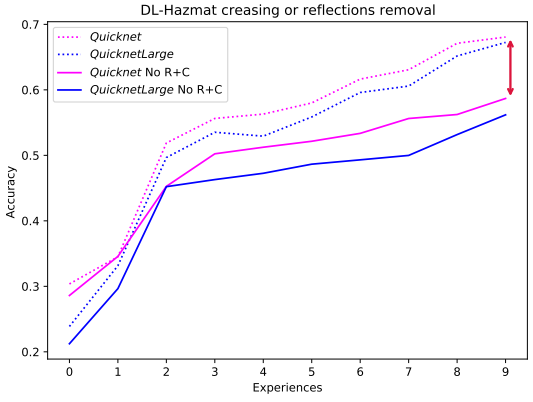


Figure 7: Accuracy comparison of our solution (BNN+LR+CWR*) using Quicknet and QuicknetLarge for DL-Hazmat dataset with/without the effects of reflections and creased, as reported in Section 4.1.1. The results without reflections and creasing effects (No R+C) have been obtained by adding the standard effects of noise, blur, scale and pose.

It is important to note that, for this evaluation, the same quantization bitwidths (q_b) were applied to both binary (q_b^{bin}) and non-binary ($q_b^{non-bin}$) layers during the backward pass, as BNN+CWR* does not differentiate between these cases. The results consistently indicate that our BNN+LR+CWR*

method surpasses the previous approach, demonstrating improvements with floating-point arithmetic and quantized implementations. This highlights the enhanced performance of BNN+LR+CWR* but a consistent accuracy gap persists compared to the cumulative training. As anticipated in Section 4.3, BiRealnet, Reactnet, and RepBNN are trained using a procedure more close to our embedded context, and this reduces the classification accuracy of such models. Additionally, we found that using $q_b = 8$ in BNN+LR+CWR* results in a noticeable accuracy decline compared to higher bitwidth settings, mirroring the findings from BNN+CWR*. This reinforces the significance of employing higher bitwidth representations during the backward pass to maintain model accuracy. For the experiments, we set the replay memory sizes as follows: $LR_{size} = 300$ for CIFAR10 and CLEAR10, $LR_{size} = 3000$ for CIFAR100, $LR_{size} = 1500$ for CORe50 and $LR_{size} = 720$ for DL-Hazmat.

Additionally, Figure 7 shows the importance of the synthetic augmentation pipeline that adds the specular light reflections and creasing effects. To validate the benefits of such effects, we compared the accuracy of Quicknet and QuicknetLarge models (using floating-point precision) on DL-Hazmat, by removing the reflections and creasing effects (denoted as *NO R+C*) from the training dataset, with the results reported in Figures 6c and 6f. The results show a consistent accuracy drop of the models *NO R+C*, estimated approximately to 10 percentage points in the last experience.

5.2. Latent Replay Memory Reduction

The storage needs of latent replay memory are closely tied to the bitwidths used for representing latent activations. As the bitwidth increases, the memory footprint of the latent replay memory also grows. Our approach leverages the 1-bit activations characteristic of BNNs to greatly reduce memory storage requirements while keeping the accuracy gap minimal, as shown in Figure 8. Our experiments reveal that BNN models can achieve a minimal accuracy gap on both CIFAR-10 (Figure 8b), CORe50 (Figure 8a) , and DL-Hazmat (Figure 9c) datasets, even with 1-bit latent activations for replay memory. This approach results in significant memory savings of 32× compared to using floating-point latent activations.

In our analysis, we explored different sizes for the latent replay memory, allocating 15, 20, and 30 elements per class. Notably, we found that varying the number of past samples in the replay memory had a relatively minor effect on model accuracy, with accuracy loss remaining within 1% for CIFAR10 and CoRE50 while it is more consistent for DL-Hazmat as the quantization impacts considerably the performance of the Quicknet model. The use of 1-bit latent activations for replay memory allows for scaling applications to support thousands of classes, as demonstrated in Figure 8, due to the substantial reduction in memory requirements.

5.3. Splitting q_b in q_b^{bin} and $q_b^{non-bin}$

As detailed in Table 1, the memory consumption of BNN weights is primarily dominated by trainable binary weights,

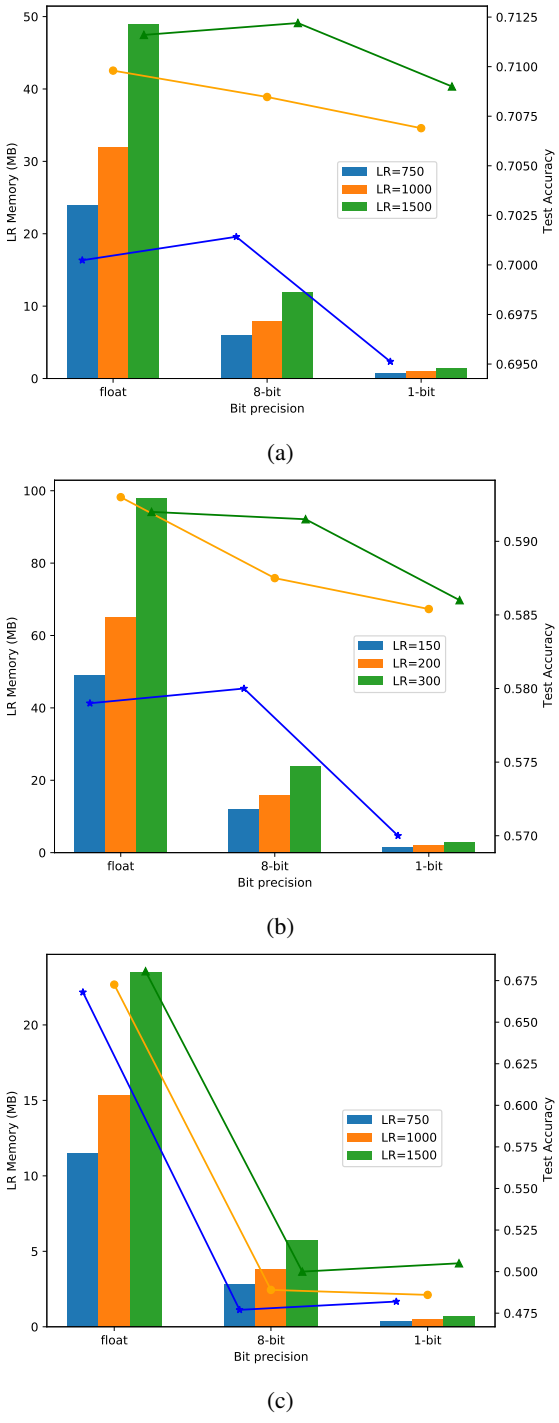
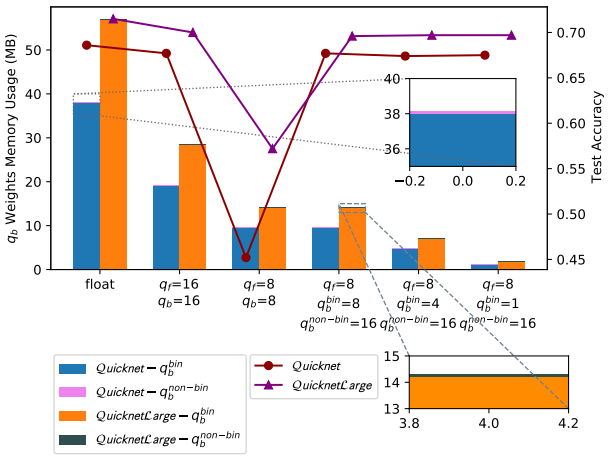
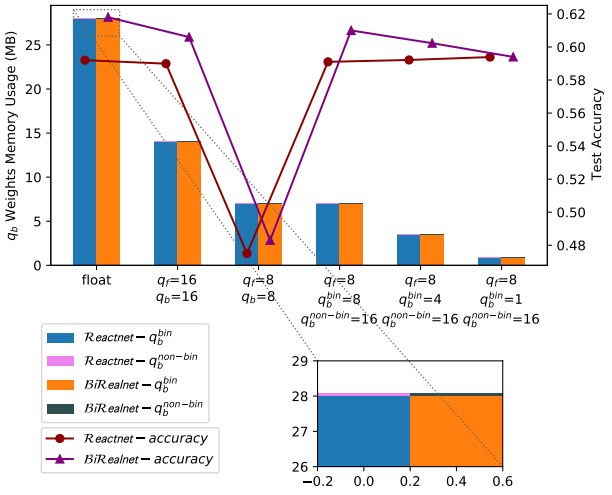


Figure 8: LR memory requirement using different quantization levels and corresponding test set accuracy on three different datasets: CIFAR10 (b), CORe50 (a), and DL-Hazmat (c). We considered 15, 20, and 30 elements for each class inside LR; for case (b) we adopted Reactnet-18 model while in (a) and c we used Quicknet.

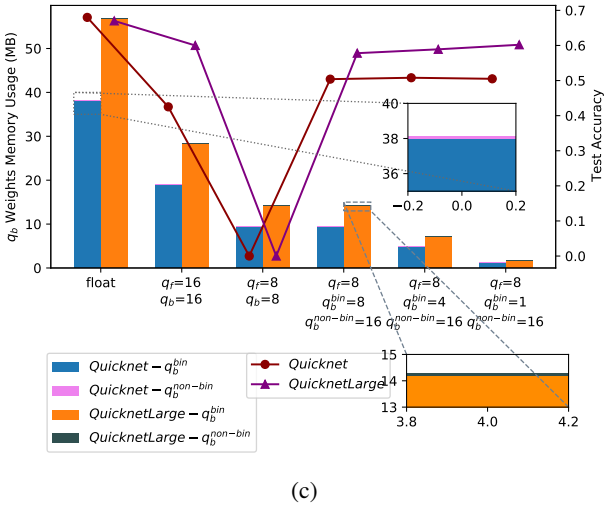
which account for nearly 100% of the memory usage. Traditionally, a binary layer is trained with latent floating-point weights (Helwegen, Widdicombe, Geiger, Liu, Cheng and Nusselder, 2019). However, if this method were applied on-device, it would significantly increase memory requirements



(a)



(b)



(c)

Figure 9: Bitwidth (q_b) memory requirement using different quantization bitwidths for backward layer on CORe50 (Figure 9a) CIFAR10 (Figure 9b,) and the introduced DL-Hazmat (Figure 9c) datasets.

during the backpropagation phase, necessitating a 32-bit representation for binary weights (q_b^{bin}).

The quantization strategy discussed in Section 3.3 provides a potential solution to this issue by reducing the bitwidth (q_b) to 8 bits. Despite this, as illustrated in Figures 9a, 9b, and 9c such a reduction leads to a noticeable decrease in model accuracy, especially for the DL-Hazmat dataset where the combination of using 8-bit both in forward and backward passes leads to poor results.

To tackle this problem, we investigated the effects of different quantization levels for binary weights (q_b^{bin}) and non-binary weights ($q_b^{non-bin}$). Specifically, we tested both 4-bit and 1-bit representations for binary weights. Our results, presented in Figure 9, show that using a 4-bit representation for binary layers does not significantly impact accuracy. Furthermore, a 1-bit representation for weights during the backpropagation stage is practical, as binary weights are fixed during on-device training. This allows the model to maintain accuracy effectively. This finding is crucial for on-device learning, as it reduces computational demands by limiting backpropagation to non-binary layers, particularly those using 16-bit quantization, as our experiments have demonstrated.

5.4. Considerations about Model Efficiency

To illustrate the practical application of our approach on real-world embedded boards, we present an estimation analysis of on-device performance. For this evaluation, we chose two widely used boards in the IoT space, both utilizing the single-thread ARMv8 architecture: the Raspberry Pi 3B and Raspberry Pi 4B.

Drawing from the efficiency assessments reported in (Bannink et al., 2021; Pellegrini et al., 2020), Table 2 compares the inference and backward pass timings of our BNN+LR+CWR* method against a non-binary approach based on Mobilenetv2 (Pellegrini et al., 2020). In the Mobilenetv2 case, the layers from the latent replay memory to the classification head use floating-point precision. The Mobilenetv2 backbone is quantized to 8-bit precision for latent activations and executed with TensorFlow Lite. Conversely, our BNN+LR+CWR* approach employs the Quicknet model with the following quantization settings: $q_f = 8$, $q_b^{bin} = 8$, and $q_b^{non-bin} = 16$, with binary inference executed using LCE (Bannink et al., 2021). The input image size is 224×224 and the batch size is 1.

Our empirical evaluation of the backward pass reveals that BNN+LR+CWR* achieves at least a 2× speedup compared to the non-binary solution. We analyzed the worst-case scenario for the backward step with $q_b^{bin} = 8$; however when q_b^{bin} is set to 1, the speedup shown in the fifth column of Table 2 is expected to improve significantly.

6. Conclusion

On-device training presents substantial opportunities within the Internet of Things (IoT) sector, enabling broader deployment of deep learning solutions on embedded devices. The lack of on-device continual learning datasets

Model	Raspberry 3B	Raspberry 4B	Binary	Quantization q_f	Quantization q_b	Forward (ms)	Backward (ms)	Speedup
Mobilenetv2 (Howard et al., 2017)	✓			8-bit	float	340	134	1.0×
Quicknet (Bannink et al., 2021)	✓		✓	1-bit	16-bit	**160**	**55**	**2.2**×
Mobilenetv2 (Howard et al., 2017)		✓		8-bit	float	225	90	1.0×
Quicknet (Bannink et al., 2021)		✓	✓	1-bit	16-bit	**105**	**38**	**2.2**×

Table 2
Efficiency comparison of our method implemented on two different embedded boards, *i.e.* Raspberry Pi 3B and 4B, using Mobilenetv2 and Quicknet model. As shown, our solution achieves up to 2.2× speedup on the same platform.

focused on real-world applications necessitates the introduction of the DataLogic HAZardous MATerial dataset, expressly designed to benchmark the classification accuracy in a Transportation&Logistics scenario. This study primarily examines the integration of Binary Neural Networks (BNNs) with Continual Learning algorithms, an area that remains underexplored in existing research. We propose employing the CWR* method alongside a replay memory, introducing customized quantization schemes to address memory limitations and computational challenges during the back-propagation process.

We can summarize the experimental findings as follows: i) through the utilization of 1-bit latent activations, we drastically lowered the memory storage needed for replay memory compared to the state-of-the-art approach, which uses 8-bit precision. Reducing storage requirements is essential for on-device training, especially in embedded systems with constrained storage capabilities. ii) We improve the accuracy of different binarized backbones and the BNN+CWR* approach. In particular, by implementing a latent replay method, we address the performance gap that often impacts BNNs, providing a safeguard against catastrophic forgetting. iii) We reduce the computational effort involved in backpropagating the latent replay by employing an effective quantization scheme. This approach enables the model to maintain high performance while minimizing computational demands during the learning phase. Alongside reduced memory usage, this advancement facilitates future on-device and real-world training of learning systems.

Concluding, a variety of future work is planned. For instance, we intend to effectively implement and optimize the approach proposed in this paper specifically for ARM CPUs, a widely used family of processors in IoT devices. Additionally, we foresee leveraging their instruction set, including NEON extensions, to further enhance the method's computational load and efficiency.

References

Alshehri, F., Muhammad, G., 2020. A comprehensive survey of the internet of things (iot) and ai-based smart healthcare. IEEE Access 9, 3660–3678.

Banner, R., Hubara, I., Hoffer, E., Soudry, D., 2018. Scalable methods for 8-bit training of neural networks. Advances in neural information processing systems 31.

Bannink, T., Hillier, A., Geiger, L., de Bruin, T., Overweel, L., Neeven, J., Helwegen, K., 2021. Larq compute engine: Design, benchmark and deploy state-of-the-art binarized neural networks. Proceedings of Machine Learning and Systems 3, 680–695.

Bethge, J., Yang, H., Bornstein, M., Meinel, C., 2019. Back to simplicity: How to train accurate bnns from scratch? arXiv preprint arXiv:1906.08637 .

Cai, H., Gan, C., Zhu, L., Han, S., 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. Advances in Neural Information Processing Systems 33, 11285–11297.

Chaudhry, A., Ranzato, M., Rohrbach, M., Elhoseiny, M., 2018. Efficient lifelong learning with a-gem. arXiv preprint arXiv:1812.00420 .

Chen, Y., Zheng, B., Zhang, Z., Wang, Q., Shen, C., Zhang, Q., 2020. Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions. ACM Computing Surveys (CSUR) 53, 1–37.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y., 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 .

Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., Sridharan, S., Vaidyanathan, K., Kaul, B., Georganas, E., et al., 2018. Mixed precision training of convolutional neural networks using integer operations. arXiv preprint arXiv:1802.00930 .

Ding, A., Qin, Y., Wang, B., Cheng, X., Jia, L., 2023. An elastic expandable fault diagnosis method of three-phase motors using continual learning for class-added sample accumulations. IEEE Transactions on Industrial Electronics .

Dong, K., Zhou, C., Ruan, Y., Li, Y., 2020. Mobilenetv2 model for image classification, in: 2020 2nd International Conference on Information Technology and Computer Application (ITCA), IEEE. pp. 476–480.

Graffieti, G., Borghi, G., Maltoni, D., 2022. Continual learning in real-life applications. IEEE Robotics and Automation Letters 7, 6195–6202.

Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P., 2015. Deep learning with limited numerical precision, in: International conference on machine learning, PMLR. pp. 1737–1746.

Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K.T., Nusselder, R., 2019. Latent weights do not exist: Rethinking binarized neural network optimization. Advances in neural information processing systems 32.

Hluchyj, M.G., Karol, M.J., 1991. Shuffle net: An application of generalized perfect shuffles to multihop lightwave networks. Journal of Lightwave Technology 9, 1386–1397.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 .

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, in: International conference on machine learning, pmlr. pp. 448–456.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2704–2713.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al., 2017. Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences 114, 3521–3526.

Koonce, B., Koonce, B., 2021a. Efficientnet. Convolutional neural networks with swift for Tensorflow: image recognition and dataset categorization , 109–123.

Koonce, B., Koonce, B., 2021b. Mobilenetv3. Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization , 125–144.

Koonce, B., Koonce, B., 2021c. Squeezenet. Convolutional neural networks with swift for tensorflow: image recognition and dataset categorization , 73–85.

Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images .

Le, Y., Yang, X., 2015. Tiny imagenet visual recognition challenge. CS 231N 7, 3.

Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X., 2021. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing 461, 370–403.

Lin, J., Zhu, L., Chen, W.M., Wang, W.C., Gan, C., Han, S., 2022. On-device training under 256kb memory. Advances in Neural Information Processing Systems 35, 22941–22954.

Lin, Z., Shi, J., Pathak, D., Ramanan, D., 2021. The clear benchmark: Continual learning on real-world imagery, in: Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track.

Liu, X., Wu, C., Menta, M., Herranz, L., Raducanu, B., Bagdanov, A.D., Jui, S., de Weijer, J.v., 2020a. Generative feature replay for class-incremental learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pp. 226–227.

Liu, Z., Shen, Z., Li, S., Helwegen, K., Huang, D., Cheng, K.T., 2021. How do adam and training strategies help bnns optimization?, in: International Conference on Machine Learning, PMLR.

Liu, Z., Shen, Z., Savvides, M., Cheng, K.T., 2020b. 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. pp. 143–159.

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.T., 2018. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm, in: Proceedings of the European conference on computer vision (ECCV), pp. 722–737.

Lomonaco, V., Maltoni, D., 2017. Core50: a new dataset and benchmark for continuous object recognition, in: Conference on robot learning, PMLR. pp. 17–26.

Lomonaco, V., Maltoni, D., Pellegrini, L., 2020. Rehearsal-free continual learning over small non-iid batches., in: CVPR Workshops, p. 3.

Maltoni, D., Lomonaco, V., 2019. Continuous learning in single-incremental-task scenarios. Neural Networks 116, 56–73.

Martinez, B., Yang, J., Bulat, A., Tzimiropoulos, G., 2020. Training binary neural networks with real-to-binary convolutions. arXiv preprint arXiv:2003.11535 .

Masana, M., Liu, X., Twardowski, B., Menta, M., Bagdanov, A.D., Van De Weijer, J., 2022. Class-incremental learning: survey and performance evaluation on image classification. IEEE Transactions on Pattern Analysis and Machine Intelligence 45, 5513–5533.

Mohamed, E., 2020. The relation of artificial intelligence with internet of things: A survey. Journal of Cybersecurity and Information Management 1, 30–24.

Nadalini, D., Rusci, M., Benini, L., Conti, F., 2023. Reduced precision floating-point optimization for deep neural network on-device learning on microcontrollers. arXiv preprint arXiv:2305.19167 .

Nadalini, D., Rusci, M., Tagliavini, G., Ravaglia, L., Benini, L., Conti, F., 2022. Pulp-trainlib: Enabling on-device training for risc-v multi-core mcus through performance-driven autotuning, in: International Conference on Embedded Computer Systems, Springer. pp. 200–216.

Ostapenko, O., Lesort, T., Rodriguez, P., Arefin, M.R., Douillard, A., Rish, I., Charlin, L., 2022. Continual learning with foundation models: An empirical study of latent replay, in: Conference on lifelong learning agents, PMLR. pp. 60–91.

Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S., 2019. Continual lifelong learning with neural networks: A review. Neural networks 113.

Pellegrini, L., Graffieti, G., Lomonaco, V., Maltoni, D., 2020. Latent replay for real-time continual learning, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE. pp. 10203–10209.

Qin, H., Gong, R., Liu, X., Bai, X., Song, J., Sebe, N., 2020. Binary neural networks: A survey. Pattern Recognition 105, 107281.

Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A., 2016. Xnor-net: Imagenet classification using binary convolutional neural networks, in: European conference on computer vision, Springer. pp. 525–542.

Ravaglia, L., Rusci, M., Nadalini, D., Capotondi, A., Conti, F., Benini, L., 2021. A tinyml platform for on-device continual learning with quantized latent replays. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 11, 789–802.

Rebuffi, S.A., Kolesnikov, A., Sperl, G., Lampert, C.H., 2017. icarl: Incremental classifier and representation learning, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, pp. 2001–2010.

Ren, H., Anicic, D., Runkler, T.A., 2021. Tinyol: Tinyml with online-learning on microcontrollers, in: 2021 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–8.

Ren, X., Qin, Y., Li, B., Wang, B., Yi, X., Jia, L., 2024. A core space gradient projection-based continual learning framework for remaining useful life prediction of machinery under variable operating conditions. Reliability Engineering & System Safety 252, 110428.

Rusu, A.A., Rabinowitz, N.C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., Hadsell, R., 2016. Progressive neural networks. arXiv preprint arXiv:1606.04671 .

Shaheen, K., Hanif, M.A., Hasan, O., Shafique, M., 2022. Continual learning for real-world autonomous systems: Algorithms, challenges and frameworks. Journal of Intelligent & Robotic Systems 105, 9.

Shi, X., Qi, Z., Cai, J., Fu, K., Zhao, Y., Li, Z., Liu, X., Liu, H., 2022. Repbnn: towards a precise binary neural network with enhanced feature map via repeating. arXiv preprint arXiv:2207.09049 .

Shin, H., Lee, J.K., Kim, J., Kim, J., 2017. Continual learning with deep generative replay. Advances in neural information processing systems 30.

Thomee, B., Shamma, D.A., Friedland, G., Elizalde, B., Ni, K., Poland, D., Borth, D., Li, L.J., 2016. Yfcc100m: The new data in multimedia research. Communications of the ACM 59, 64–73.

Vitter, J.S., 1985. Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) 11, 37–57.

Vorabbi, L., Maltoni, D., Borghi, G., Santi, S., 2024a. Enabling on-device continual learning with binary neural networks and latent replay. Proceedings Copyright 25, 36.

Vorabbi, L., Maltoni, D., Santi, S., 2023. On-device learning with binary neural networks, in: International Conference on Image Analysis and Processing, Springer. pp. 39–50.

Vorabbi, L., Maltoni, D., Santi, S., 2024b. Optimizing data flow in binary neural networks. Sensors 24. URL: https://www.mdpi.com/1424-8220/24/15/4780, doi:10.3390/s24154780.

Zhang, Y., Zhang, Z., Lew, L., 2022. Pokebnn: A binary pursuit of lightweight accuracy, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12475–12485.

Lorenzo Vorabbi received B.S. and M.S. degrees in electronic engineering from Alma Mater Studiorum University, Bologna, Italy, in 2009 and 2011, respectively. In 2011, he joined Datalogic, where he covered different positions and he currently leads the Machine Learning and Computer Vision team in EMEA. In 2024, he received a Ph.D. degree in computer science and engineering from

the same university (DISI department), focusing on new architectures and training methods to efficiently deploy binary neural networks on embedded devices with real-time constraints. He holds seven patents related to the barcode technology.

Angelo Carraggi achieved his M.S. degree in Computer Engineering from the University of Modena and Reggio Emilia, Italy, in 2018. In 2018, he joined Datalogic, where he presently holds a position in the R&D department, focusing on Machine Learning and Computer Vision. He holds one patent related to the computer vision field.

Davide Maltoni is a Full Professor at University of Bologna (Dept. of Computer Science and Engineering - DISI). His research interests are in the area of Pattern Recognition, Computer Vision, Machine Learning and Computational Neuroscience. Davide Maltoni is co-director of the Biometric Systems Laboratory (BioLab), which is internationally known for its research and publications in the field. Several original techniques have been proposed by BioLab team for fingerprint feature extraction, matching and classification, for hand shape verification, for face location and for performance evaluation of biometric systems. Davide Maltoni is co-author of the Handbook of Fingerprint Recognition published by Springer, 2022 and holds three patents on Fingerprint Recognition. He has been elected IAPR (International Association for Pattern Recognition) Fellow 2010.

Guido Borghi is an Associate Professor within the Department of Education and Humanities, University of Modena and Reggio Emilia. He received the M.Sc. degree in Computer Engineering and the Ph.D. in Information and Communication Technologies from the University of Modena and Reggio Emilia, Italy, in 2015 and 2019, respectively. His research interests include Computer Vision and Deep Learning techniques applied to intensity and depth images for Face Analysis, Biometrics, Driver Monitoring and Human Computer Interaction.

Stefano Santi holds a Ph.D. and is a Senior Technology Advisor of the CTO at Datalogic, where he previously led the advanced research in new technology and deep learning in North America. His interests range from machine learning and deep learning technology applied to the state-of-the-art barcode decoding methods, to the code optimization for strict real-time performance on embedded systems. Stefano Santi holds nine patents and several other publications on a variety of aspects related to acquisition systems and computer vision applied to barcode technology.