On-Device Learning with Binary Neural Networks

On-Device Learning with Binary Neural Networks

Lorenzo Vorabbi[1,2][0000−0002−4634−2044], Davide Maltoni[2][0000−0002−6329−6756], and Stefano Santi[1]

[1] Datalogic Labs, Bologna 40012, IT
[2] University of Bologna, DISI, Cesena Campus, Cesena 47521, IT
{lorenzo.vorabbi2,davide.maltoni}@unibo.it
{lorenzo.vorabbi,stefano.santi}@datalogic.com

Abstract. Existing Continual Learning (CL) solutions only partially address the constraints on power, memory and computation of the deep learning models when deployed on low-power embedded CPUs. In this paper, we propose a CL solution that embraces the recent advancements in CL field and the efficiency of the Binary Neural Networks (BNN), that use 1-bit for weights and activations to efficiently execute deep learning models. We propose a hybrid quantization of CWR* (an effective CL approach) that considers differently forward and backward pass in order to retain more precision during gradient update step and at the same time minimizing the latency overhead. The choice of a binary network as backbone is essential to meet the constraints of low power devices and, to the best of authors' knowledge, this is the first attempt to prove on-device learning with BNN. The experimental validation carried out confirms the validity and the suitability of the proposed method.

Keywords: Binary Neural Networks · On-device Learning · Continual Learning.

1 Introduction

Integrating a deep learning model into an embedded system can be a challenging task for two main reasons: the model may not fit into the embedded system memory and, the time efficiency may not satisfy the application requirements. A number of light architectures have been proposed to mitigate these problems (MobileNets [1], EfficientNets [2], NASNets [3]) but they heavily rely on floating point computation which is not always available (or efficient) on tiny devices. Binary Neural Networks (BNN), where a single bit is used to encode weights and activations, emerged as an interesting approach to speed up the model inference relying on packed bitwise operations [4]. However, almost no literature work addresses the problem of training (or tuning) such models on-device, a task which is still more complex than inference because:

- quantization is known to affect back propagation and weights update
- popular inference engines (e.g. Tensorflow Lite, pytorch mobile, ecc.) do not support model training

This work proposes on-device learning of BNN to enable continual learning of a pre-trained model. We start from CWR* [5], a simple but effective continual learning approach that limits weight updates to the output head, and designs an ad-hoc quantization approach that preserves most of the accuracy with respect to a floating point implementation. We prove that several state of the art BNN models can be used in conjunction with our approach to achieve good performance on classical continual learning dataset/benchmarks such as CORe50 [6], CIFAR10 [7] and CIFAR100 [7].

2 Related Literature

2.1 Continual Learning

The classical deep learning approach is to train a model on a large batch of data and then freeze it before deployment on edge devices; this does not allow adapting the model to a changing environment where new classes (NC scenario) or new items/variation of known classes (NI scenario) can appear over time. Collecting new data and periodically retraining a model from scratch is not efficient and sometime not possible because of privacy, so the CL approach is to adapt an existing model by using only new data. Unfortunately, this is prone to forgetting old knowledge, and specific techniques are necessary to balance the model stability and plasticity. For a survey of existing CL methods see [8].

In this work we focus on Single Object Recognition task addressing the two CL scenarios of NI and NC; in both cases, the learning phase of the model is usually splitted in *experiences*, each one containing different training samples belonging or not to known classes (this depends on the CL scenario).
CWR* mantains two sets of weights for the output classification layer: cw are the consolidated weights used during inference while tw are the temporary weights that are iteratively updated during back-propagation. cw are initialized to 0 before the first batch and then updated according to Algorithm 1 (for more details see [5]), while tw are reset to 0 before each training mini-batch. CWR*, for each already encountered class (of current training batch), reloads the consolidated weights cw at the beginning of each training batch and, during the consolidation step, adopts a weighted sum based on the number of the training samples encountered in the past batches and those of current batch. The consolidation step has a negligible overhead and can be quantized adopting the same quantization scheme used for CWR* weights. In CWR*, during the first training experience (supposed to be executed offline) all the layers of the model are trained but from the second experience, only the weights of the output classification layer are adjusted during the back-prop stage, to simulate a real case scenario (lines 9 - 12 of Algorithm 1).

2.2 Binary Neural Networks

Quantization is a technique that yields compact models compared to their floating-point counterparts, by representing the network weights and activations with

Algorithm 1 CWR* pseudocode: $\overline{\Theta}$ are the class-shared parameters. Both tw and cw refer to the same layer index k of the model.

procedure CWR*
  $cw_k = 0$ ▷ k is the index of the classification layer
  $past = 0$ ▷ number of samples for each class i encountered
  init $\overline{\Theta}$ random or from pre-trained model
  for each training batch B_j **do** ▷ B_j is the mini-batch of index j
    expand layer k with neurons for the new classes in B_j never seen before
    $tw_k[i] = \begin{cases} cw_k[i], & \text{if class } i \text{ in } B_j \\ 0, & \text{otherwise} \end{cases}$
    train the model with SGD
    if $B_j = B_1$ **then**
      learn both $\overline{\Theta}$ and tw_k
    else
      learn tw_k while keeping $\overline{\Theta}$ fixed
    for each class i in B_j **do** ▷ consolidation step
      $wpast_i = \sqrt{\frac{past_i}{cur_i}}$, where cur_i is the number of patterns of class i in B_j
      $cw_k[i] = \frac{cw_k[i] \cdot wpast_i + (tw_k[i] - avg(tw_k))}{wpast_i + 1}$
      $past_i = past_i + cur_i$
    test the model by using $\overline{\Theta}$ and cw_k

very low precision. The most extreme quantization is binarization, where data can only have two possible values, namely $-1\,(0)$ or $+1\,(1)$. By representing weights and activations using only 1-bit, the resulting memory footprint of the model is dramatically reduced and also the heavy matrix multiplication operations can be replaced with light-weighted bitwise XNOR operations and Bitcount operations. According to [9], that compared the speedups of binary layers w.r.t. the 8-bit quantized and floating point layers, a binary implementation can achieve a lower inference time from 9 to $12\times$ on a low power ARM CPU. Therefore, Binary Neural Networks combine many hardware friendly properties including memory saving, power efficiency and significant acceleration; for some network topologies, BNN can be executed on device without the usage of floating-point operations [10] simplifying the deployment on ASIC or FPGA hardware. For a survey on binary neural networks see [4].

3 On-Device CWR Optimization

3.1 Gradients Computation

In this section we make explicit the weights update in the classification layer; without loss of generality, a neural network $M(\cdot)$ is composed by a sequence of k layers represented as:

$$M(\cdot) = f_{w_k}\left(f_{w_{k-1}}\left(\cdots f_{w_2}\left(f_{w_1}(\cdot)\right)\right)\right) \tag{1}$$

where w_i represents the weights of the i^{th} layer. In CWR* the temporary weights tw_k (lines 10 and 12 of Alg. 1) are updated according to Equations 9 and 10, whose quantization is discussed in the next section. Denoting with a_i and a_{i+1} [3] the input and output activations of the i^{th} layer respectively, with $\mathcal{L}$ the loss function, the backpropagation process consists in the computation of two different sets of gradients: $\frac{\partial \mathcal{L}}{\partial a_i}$ and $\frac{\partial \mathcal{L}}{\partial w_i}$.

In CWR* the on-device backpropagation algorithm is limited to the last layer which can be considered a linear layer (with a non-linear activation function) with the following forward formula:

$$a_{k+1} = f_k(o_{k+1}), \;\; o_{k+1} = a_k W_k + b_k \tag{2}$$

where a_{k+1} represents the output of the neural network.

Considering a classification task (with M classes) with an unitary batch size, the *Cross-Entropy* loss function is formulated as:

$$\mathcal{H}(y, a_{k+1}) = -\sum_{i=0}^{M-1} y^i log\left(a_{k+1}^i\right) \tag{3}$$

where y^i represents the element of an one-hot encoded vector of ground truth and a_{k+1}^i is the i^{th} output activation sample. Using the softmax as activation for the last layer, reported below:

$$a_{k+1}\left(o_{k+1}^t\right) = \frac{e^{o_{k+1}^t}}{\sum_{j=1}^{M} e^{o_{k+1}^j}} \tag{4}$$

, the gradient formulas for the last classification layer can be expressed using the chain rule:

$$\frac{\partial \mathcal{H}}{\partial W_k} = \frac{\partial \mathcal{H}}{\partial a_{k+1}} \frac{\partial a_{k+1}}{\partial o_{k+1}} \frac{\partial o_{k+1}}{\partial W_k} \tag{5}$$

$$\frac{\partial \mathcal{H}}{\partial b_k} = \frac{\partial \mathcal{H}}{\partial a_{k+1}} \frac{\partial a_{k+1}}{\partial o_{k+1}} \frac{\partial o_{k+1}}{\partial b_k} \tag{6}$$

The final expression for Eq. 5 using the Eq. 3 as loss function and 4 as non-linear $f_k(\cdot)$ is a well-known result, that can be easily derived:

$$\frac{\partial \mathcal{H}}{\partial W_k} = (a_{k+1} - y)\, a_k \tag{7}$$

$$\frac{\partial \mathcal{H}}{\partial b_k} = (a_{k+1} - y) \tag{8}$$

[3] Note that the output a_{i+1} of level i corresponds to the input of level $i+1$

Using a stochastic gradient descent optimizer with learning rate η, the weights update equation is:

$$W_k^{i+1} = W_k^i - \eta\,(a_{k+1} - y)\,a_k \tag{9}$$

$$b_k^{i+1} = b_k^i - \eta\,(a_{k+1} - y) \tag{10}$$

Therefore in CWR* the temporary weights tw_k (lines 10 and 12 of Alg. 1) are updated according to Equations 9 and 10, whose quantization is discussed in the next section.

3.2 Quantization Strategy

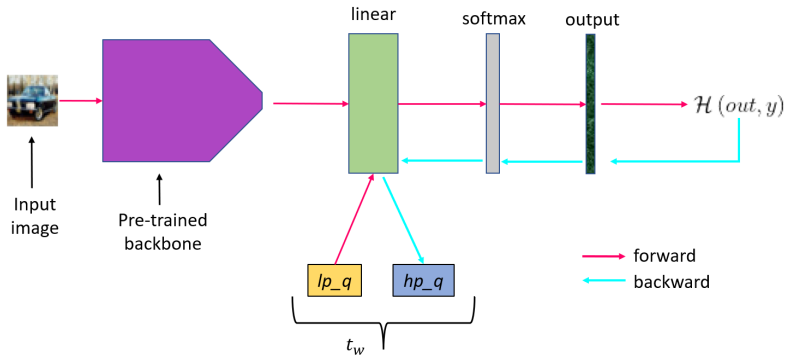


Fig. 1: Double quantization scheme that uses a different quantization level for weights/activations used in forward and backward pass.

Our approach considers two different quantizations: the former uses 1-bit (also called binarization) to represent weights and activations employed by the pre-trained backbone; the latter is used in the last classification layer, to quantize both forward and backward operations. This solution both reduces the latency and simplifies the adaptation of the model on new item/classes encountered.
In particular, for the last layer quantization we followed the scheme proposed in [11] and implemented in GEMMLOWP library [12]. The quantized output of a 32-bit floating point linear layer, reported in Eq. 2, can be represented as:

$$\overline{o_{k+1}^{int\_q}} = cast\_to\_int\_q\lfloor s_k^{int\_32}\left(\overline{W_k^{int\_q}}\,a_k^{int\_q} + \overline{b_k^{int\_q}}\right)\rceil \tag{11}$$

The quantization Eq. 11 depends on the number of the quantization q bits used $(8, 16, 32)$, $\bar{\cdot}$ represents the quantized version of a tensor and s^{int_32} is

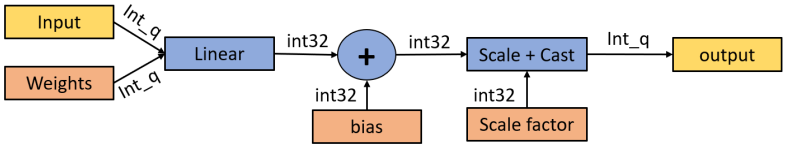


Fig. 2: Quantization scheme adopted using q bits for weights and activations.

the fixed-point scaling factor having 32-bit precision, as shown in Fig. 2. Similarly to previous works [13, 14], we used the straight-through estimator (STE) approach to approximate differentiation through discrete variables; STE represents a simple and hardware-friendly method to deal with the computation of the derivative of discrete variables that are zero almost everywhere.

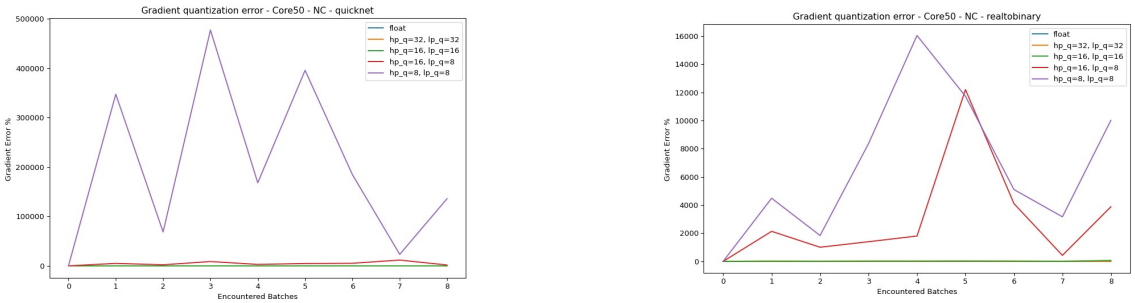
Based on the results reported in [15, 16, 17], the quantization of the gradients in Equations 7 and 8 represents the main cause of accuracy degradation during training and therefore we propose to use two separate versions of layer weights W_k, one with low-precision (lp_q) and another with higher precision (hp_q). As shown in Fig. 1, the idea is to use the lp_q version of the weights for the computations that have strict timing deadlines (forward pass), while the hp_q version is adopted during the weight update step (Equations 9 and 10), which has typically more relaxed timing constraints (it can be executed also as a background process). Every time a new high-precision copy of weights is computed, a lower version is derived from it and stored.

Gradient quantization inevitably introduces an approximation error that can affect the accuracy of the model; to check the amount of approximation for different quantization levels, for each mini-batch, we compute the Mean Absolute Error (MAE, in percentage) between the floating point gradient and the quantized one for the weight tensor of the CWR* layer (for the dataset CORe50 [6]). The MAE is then accumulated for all training mini-batches of each experience, as shown in Fig. 3. In order to evaluate only the quantization error introduced, both floating-point and quantized gradients are computed starting from the same W_k^i weights (Eq. 9). The plot curves of Figures 3a and 3b refer respectively to the *quicknet* [9] and *realtobinary* [18] models; it is evident that the quantization error introduced using the lp_q with 8 bits is much larger compared to higher quantization schemes (16/32 bits or floating point) whose gap w.r.t. the floating point implementation is quite low, as pointed out in Section 5.

4 Experiments

We evaluate the proposed approach on three classification datasets: CORe50, CIFAR10 and CIFAR100 with different BNN architectures. The BNN models employed for CORe50 have been pre-trained on ImageNet [19] and taken from Larq repository[4]; instead, the models used for CIFAR10 and CIFAR100 have

[4] `https://docs.larq.dev/zoo/api/sota/`



(a) *quicknet* (b) *realtobinary*

Fig. 3: Accumulation of gradient quantization errors (Mean Absolute Error in percentage) between quantized and floating-point versions for each experience. During the first experience the gradient computation is always executed in floating-point.

been pre-trained on Tiny Imagenet[5]. For each dataset, we conducted several tests using a different number of quantization bits with the same training procedure. Our work is targeting a model that could continuously learn and therefore we limited the number of epochs to 10 for the first experience and to 5 for the remaining. The results of Eq. 7 and 9 require the adoption of the Cross Entropy as loss function and the Stochastic Gradient Descent (SGD) as optimizer; the choice of SGD is encouraged as it requires a simple computation with a limited overhead compared to the Adam [20] optimizer. The binarization of weights and activations always happens at training time using an approximation of the gradient (STE introduced in Section 3.2 or derived solutions that are model dependent) for *sign* function.

Hereafter we provide some details on the datasets and related CL protocols:

CORe50 [6] It is a dataset specifically designed for Continuous Object Recognition containing a collection of 50 domestic objects belonging to 10 categories. The dataset has been collected in 11 distinct sessions (8 indoor and 3 outdoor) characterized by different backgrounds and lighting. For the continuous learning scenarios (NI, NC) we use the same test set composed of sessions #3, #7 and #10. The remaining 8 sessions are split in batches and provided sequentially during training obtaining 9 experiences for NC scenario and 8 for NI. No augmentation procedure has been implemented

[5] `http://cs231n.stanford.edu/tiny-imagenet-200.zip`

since the dataset already contains enough variability in terms of rotations, flips and brightness variation. The input RGB image is standardized and rescaled to the size of $128 \times 128 \times 3$.

CIFAR10 and CIFAR100 [7] Due to the lower number of classes, the NC scenario for CIFAR10 contains 5 experiences (adding 2 classes for each experience) while 10 are used for CIFAR100. For both datasets the NI scenario is composed by 10 experiences. Similar to CORe50, the test set does not change over the experiences. The RGB images are scaled to the interval $[-1.0\,;+1.0]$ and the following data augmentation was used: zero padding of 4 pixels for each size, a random 32×32 crop and a random horizontal flip. No augmentation is used at test time.

On CORe50 dataset, we evaluated the three binary models reported below:

Realtobinary [18] This network proposes a real-to-binary attention matching mechanism that aims to match spatial attention maps computed at the output of the binary and real-valued convolutions. In addition, the authors proposed to use the real-valued activations of the binary network before the binarization of the next layer to compute scaling factors, used to rescale the activations produced after the application of the binary convolution.

Quicknet and QuicknetLarge[9] This network follows the previous works [21, 22, 18] proposing a sequence of blocks, each one with a different number of binary 3×3 convolutions and residual connections over each layer. Transition blocks between each residual section halve the spatial resolution and increase the filter count. QuicknetLarge employs more blocks and feature maps to increase accuracy.

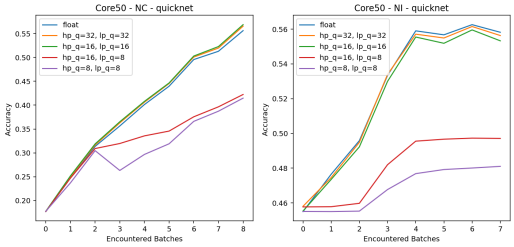
For CIFAR10 and CIFAR100 datasets, whose input resolution is 32×32, we evaluated the following networks (pre-trained on Tiny Imagenet):

BiRealNet[21] It is a modified version of classical ResNet that proposes to preserve the real activations before the sign function to increase the representational capability of the 1-bit CNN, through a simple shortcut. Bi-RealNet adopts a tight approximation to the derivative of the non-differentiable sign function with respect to activation and a magnitude-aware gradient to update weight parameters. We used the instance of the network that uses *18-layers*[6].

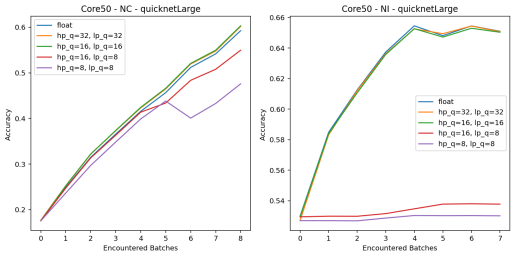
ReactNet[23] To further compress compact networks, this model constructs a baseline based on MobileNetV1 [1] and add shortcut to bypass every 1-bit convolutional layer that has the same number of input and output channels. The 3×3 depth-wise and the 1×1 point-wise convolutional blocks of MobileNet are replaced by the 3×3 and 1×1 vanilla convolutions in parallel with shortcuts in React Net[7]. As for Bi-Real Net, we tested the version of React Net that uses *18-layers*.

[6] Refer to the following `https://github.com/liuzechun/Bi-Real-net` repository for all the details.

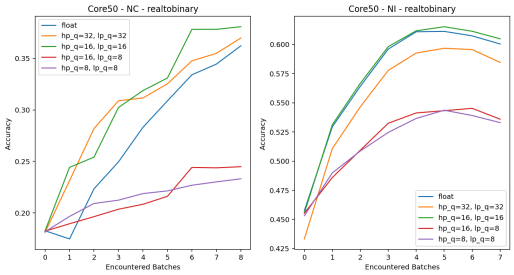
[7] Refer to the following `https://github.com/liuzechun/ReActNet` repository for all the details.



(a) Quicknet



(b) QuicknetLarge



(c) Realtobinary

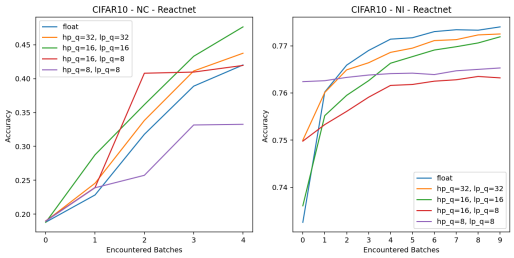
Fig. 4: CORe50 accuracy results using different quantization methods.

Our test were performed on both the NI and NC scenario (discussed in Section 2.1). Fig. 4, 5 and 6 summarize the experimental results. On CORe50 dataset (Fig. 4) NC scenario, the quantization scheme lp_8 gets a consistent accuracy drop over the experiences showing a limited learning capability; instead, the quantizations with lp_16 and lp_32 reach the same accuracy level of the floating point model. A similar situation can be observed in the NI scenario with the exception of the QuicknetLarge model where the lower quantization schemes are not able to increase the accuracy of the first experience. For datasets CIFAR10 and CIFAR100 (Fig. 5 and 6) we find similar results for the NI scenario, where the 8-bit quantization scheme limits the learning capability of the model during the experiences. Instead, in the NC scenario, both Bi-Realnet and Reactnet models with lp_8 quantization, are able to reach an accuracy result closed to the

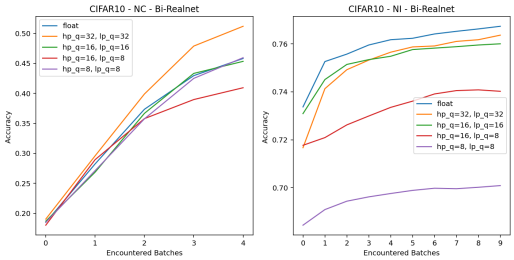
floating-point model. From our analysis it appears that the 8-bit quantization of the gradients limits noticeably the learning ability of a binary model when employed in a continual learning scenario for CWR* method. In order to reach accuracy comparable to a floating point implementation we devise the adoption of at least 16 bits both for lp and hp; it is worth noting that the computational effort of 16 bits is anyway limited in CWR* because the quantization is confined to the last classification layer.

5 Conclusion

On-device training (or adaptation) can play an essential role in the IoT, enabling the large adoption of deep learning solutions. In this work we focused on implementation of CWR* on edge-devices, relying on binary neural networks as backbone and proposing an ad-hoc quantization scheme. We discovered that 8-bit quantization degrades too much the learning capability of the model, while 16 bits is a good compromise. To the best of authors' knowledge, this work is the first to explore on-device continual learning with binary network; in the future work we intend to explore the application of binary neural networks in combination of CL methods relying on latent replay [24], which is particularly intriguing given the low memory footprint of 1-bit activation.

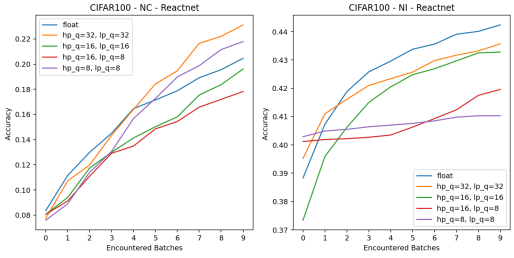


(a) Reactnet

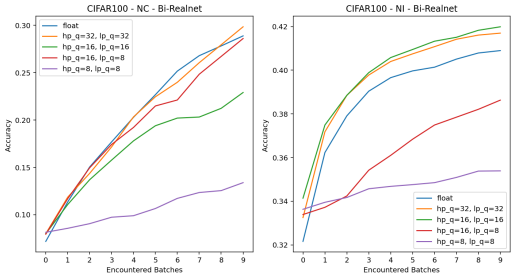


(b) Bi-Realnet

Fig. 5: CIFAR10 accuracy results using different quantization methods.



(a) Reactnet



(b) Bi-Realnet

Fig. 6: CIFAR100 accuracy results using different quantization methods.

References

[1] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[2] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[3] Han Cai, Ligeng Zhu, and Song Han. "Proxylessnas: Direct neural architecture search on target task and hardware". In: *arXiv preprint arXiv:1812.00332* (2018).

[4] Haotong Qin et al. "Binary neural networks: A survey". In: *Pattern Recognition* 105 (2020), p. 107281.

[5] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. "Rehearsal-Free Continual Learning over Small Non-IID Batches." In: *CVPR Workshops*. Vol. 1. 2. 2020, p. 3.

[6] 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.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[8] 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.

[9] 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.

[10] L. Vorabbi, D. Maltoni, and S. Santi. *Optimizing data-flow in Binary Neural Networks*. 2023. arXiv: 2304.00952 [cs.LG].

[11] 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.

[12] Benoit Jacob, P Warden, and ME Guney. "gemmlowp: a small self-contained low-precision GEMM library.(2017)". In: *URL https://github. com/google/gemmlowp* (2017).

[13] Itay Hubara et al. "Binarized neural networks". In: *Advances in neural information processing systems* 29 (2016).

[14] Asit Mishra et al. "WRPN: Wide reduced-precision networks". In: *arXiv preprint arXiv:1709.01134* (2017).

[15] Suyog Gupta et al. "Deep learning with limited numerical precision". In: *International conference on machine learning*. PMLR. 2015, pp. 1737–1746.

[16] Dipankar Das et al. "Mixed precision training of convolutional neural networks using integer operations". In: *arXiv preprint arXiv:1802.00930* (2018).

[17] Ron Banner et al. "Scalable methods for 8-bit training of neural networks". In: *Advances in neural information processing systems* 31 (2018).

[18] Brais Martinez et al. "Training binary neural networks with real-to-binary convolutions". In: *arXiv preprint arXiv:2003.11535* (2020).

[19] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[20] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[21] Zechun Liu et al. "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)*. 2018, pp. 722–737.

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

[23] Zechun Liu et al. "Reactnet: Towards precise binary neural network with generalized activation functions". In: *European conference on computer vision*. Springer. 2020, pp. 143–159.

[24] 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.