CHARLES: a C++ Fixed-Point Library for Photonic-Aware Neural Networks

Emilio Paolini[a,b,c], Lorenzo De Marinis[a], Luca Maggiani[c], Marco Cococcioni[d], Nicola Andriolli[b,e]

[a]*Scuola Superiore Sant'Anna, Pisa, 56124, Italy.*
[b]*National Research Council of Italy - Institute of Electronics, Information Engineering and Telecommunications (CNR-IEIIT), Pisa, 56122, Italy.*
[c]*Sma-RTy Italia Srl, Carugate, 20061, Italy.*
[d]*Department of Information Engineering, University of Pisa, Pisa, 56122, Italy.*
[e]*National Inter-University Consortium for Telecommunications (CNIT), Pisa, 56122, Italy.*

Abstract

In this paper we present CHARLES (C++ pHotonic Aware neuRaL nEtworkS), a C++ library aimed at providing a flexible tool to simulate the behavior of Photonic-Aware Neural Network (PANN). PANN are neural network architectures aware of the constraints due to the underlying photonic hardware, mostly in terms of low equivalent precision of the computations. For this reason, CHARLES exploits fixed-point computations for inference, while it supports both floating-point and fixed-point numerical formats for training. In this way, we can compare the effects due to the quantization in the inference phase when the training phase is performed on a classical floating-point model and on a model exploiting high-precision fixed-point numbers.

To validate CHARLES and identify the most suited numerical format for PANN training, we report the simulation results obtained considering three datasets: Iris, MNIST, and Fashion-MNIST. Fixed-training is shown to outperform floating-training when executing inference on bitwidths suitable for photonic implementation. Indeed, performing the training phase in the floating-point domain and then quantizing to lower bitwidths results in a very high accuracy loss. Instead, when fixed-point numbers are exploited in the training phase, the accuracy loss due to quantization to lower bitwidths is significantly reduced.

In particular, we show that for Iris dataset, fixed-training achieves a performance similar to floating-training. Fixed-training allows to obtain an accuracy of 90.4% and 68.1% with the MNIST and Fashion-MNIST datasets using only 6 bits, while the floating-training reaches an accuracy of just 25.4% and 50.0% when exploiting the same bitwidths.

Keywords: Photonic Neuromorphic Computing, Photonic Aware Neural Networks, Hardware Accelerators, Fixed-Point training/inference, C++ Library

1. Introduction

Deep Learning (DL) models have been successfully deployed in various domains, such as cybersecurity, natural language processing, and computer vision, including image classification [1] and object detection [2]. These results have been made possible thanks to the ever-increasing complexity of the developed models, which has doubled every 3.4 months since 2012 [3]. However, the exponential growth in computing required to train state-of-the-art Neural Network (NN) models collides with the end of Moore's law [4]. Thus, researchers are focusing on the exploitation of alternative hardware platforms to support the acceleration of NN models [5, 6]. This is of particular interest in cybersecurity contexts, where advanced information processing and computing capability are required [7, 8]. In this context, photonic accelerators appear as a promising solution to replace traditional hardware in specific applications,

able to bring significant improvements in terms of speed up ($> 10^3$), power consumption (10^2) and footprint reduction ($> 10^2$) with respect to the electronic counterparts [9]. Indeed, photonic computations can leverage passive optical circuits to perform both linear [9] and nonlinear operations of NNs [10]. By relying on passive elements, photonic accelerators can perform operations at ultra-high speed without energy consumption beyond transmitters and receivers. Furthermore, the inherent parallelism of the photonic hardware [11] makes it a suitable platform for NN computations, where each neuron performs small parts of the overall processing in parallel. Moreover, with many companies focusing on photonic accelerators [12, 13, 14], the aforementioned advantages of photonic hardware will have practical applications in the foreseeable future.

Despite the advantages that Photonic Neural Networks (PNNs) may bring in the acceleration of NN computations, especially for the most demanding models, i.e., Deep Neural Networks (DNNs), these processors have constraints deriving from the analog nature of their processing that need to be properly taken into account. To address this issue, in [15] we have introduced the concept of Photonic-Aware Neural Network (PANN), NN architectures compliant with photonic constraints. In particular, the main limitations introduced by the underlying hardware concern the limited resolution of the computations (in terms of the effective number of bits) and the requirement to use positive-valued inputs. Furthermore, although some works focus on the training phase of NNs in the photonic domain [16, 17, 18, 19], the current limitation of photonic hardware imposes to focus on the acceleration for the inference phase [20]. This is due to the complexity of the training phase, which involves gradient computation or even more complex operations. Thus, many photonic implementations support only the inference phase, and the weights are obtained by using software frameworks. Moreover, in some implementations NNs cannot be trained at all since the weights are frozen in hardware during fabrication [10].

In the field of PANNs, the lack of a flexible tool that allows the simulation of their behavior is noticeable. Two frameworks, namely Neuroptica [21] and Neurophox [22], have been developed for the simulation of photonic neural networks with specific architectures. In particular, the two libraries allow to simulate and train PNN based on Mach-Zender Interferometer (MZI) meshes. Neuroptica provides both Keras-like API, allowing the training of stacked structures, and lower-level abstraction, enabling the direct control of MZI phase shifters. Neurophox implements the Haar random technique [22], allowing the simulation of the calibration and the training of matrix multiplier processors in the optical domain using MZIs. In particular, the framework enables the software simulation for mesh network layers in orthogonal and unitary neural networks. These kinds of networks have been studied for synthetic Natural Language Processing (NLP) tasks. Even though these two frameworks are powerful tools, they are limited to PNN based on unitary photonics.

Another open-source library has been developed on top of PyTorch, namely ONNet [23] for the simulation of optical diffractive DNNs. However, even this library does not offer a solution that can address the required flexibility for the simulation of PANNs.

This implies that a tool able to offer a high degree of flexibility to study the behavior of generic PANN was missing, even though many software libraries have been developed for NNs implemented with electronic circuitry. Indeed, previous works have considered quantization of NNs for electronic implementations [24, 25, 26] to speed up computations and lower the power consumption. However, they did not consider the peculiar constraints derived from the underlying photonic hardware. Other works in the field of photonic accelerators have experimentally shown NN operations in the photonic domain, even though without properly assessing the impact of varying ENOB [27]. Finally, in another work [28] simulations have been carried out for a ring-based architecture, studying the impact of varying ENOB on a different dataset, i.e., ImageNet. However, the architectures studied by the authors are not amenable to a photonic implementation in the short term.

For these reasons, we decided to realize CHARLES [29], an easy-to-use, yet powerful library that aims to fill this gap. The developed library offers the flexibility needed to enable users to design generic PANN architectures in a fast and easy way. Furthermore, the same tool could be then used to obtain suitable weights to be mapped into photonic hardware.

In the remainder of this paper, (i) we first introduce state-of-the-art photonic architectures for accelerating NN computations; (ii) we describe how photonic constraints due to the underlying hardware can be faced, with a focus on the limited bit resolution, resulting in PANN architectures; (iii) we discuss the main aspects related to CHARLES library, highlighting its main functionalities; (iv) we carry out experiments on three well-known datasets, i.e., Iris, MNIST, and Fashion-MNIST to show the CHARLES operation and discuss the most suited numerical format for PANN training; and finally, we conclude the paper.

2. Photonic Architectures for Neural Networks

Many research efforts have been focused on accelerating NN computations leveraging novel hardware platforms, among which photonics is one of the most promising [9, 30]. The exploitation of photonics-based hardware to accelerate matrix-vector multiplications in the inference phase, which are the most computationally demanding operations in NNs [31], is a currently very active research branch. The first optical solutions were investigated many years ago: in [32] a PNN prototype was demonstrated, where an optical system implemented a fully-connected network of 32 neurons. Another relevant milestone in the PNN field was reached thanks to the exploitation of nonlinear crystals, i.e., 3D devices that could store a large number of weights [33]. In this context, in [34] the authors proposed an optical network capable of performing face recognition by leveraging a single photorefractive crystal. Despite the promising potential of these solutions, the interest in PNNs started to fade due to the fact that (i) the photonic hardware is advantageous only for large NN; (ii) the related technology was not mature enough for this purpose, which prevented fruitful operations and control of large optical networks [33].

The situation has drastically changed in recent years [35, 36], thanks to advancements in both the manufacturing processes due to the increased maturity of bulk optoelectronic devices and of entire photonic integration platforms (e.g., silicon photonics) and in DNN architectures. A programmable nanophotonic processor based on MZIs in a silicon photonic integrated circuit has been demonstrated [16], highlighting its advantages in terms of computational speed and power efficiency over state-of-the-art electronics. The general architecture for one layer of the optical NN is represented in Fig. 1(a). It is composed of two optical interference units, an optical amplification unit, and an optical nonlinearity unit. In Fig. 1(b) the structure of the MZI is sketched, composed of optical 50% couplers and phase shifters. This element is the building block of the optical interference units needed to perform arbitrary matrix multiplications on the input optical signal. This mesh of MZI can implement the singular value decomposition, which is a factorization of any matrix M with two unitary matrices U, V and a diagonal matrix Σ in the form of $\mathrm{M} = \mathrm{U}\Sigma\mathrm{V}^{\dagger}$, realizing a matrix multiplication with very low power consumption, thus making it a perfect hardware for DNNs. In [37], the authors proposed an MZI-based convolution engine and demonstrate image classification tasks. Such architecture is supposed to perform million inferences per second, outperforming state-of-the-art Application Specific Integrated Circuit (ASIC) in speed by 30 times.

More recently, in [38], a hybrid architecture has been proposed, relying on both optics and electronics. In this solution, an optical layer is deployed prior to an electronic one, showing similar accuracies with respect to comparable electronic implementations, while keeping energy consumption very low. Furthermore, another recent work [39] exploits a wavelength division multiplexing architecture to deploy a circuit for PNN. The building block for this architecture is depicted in Fig. 2. The structure exploits a Phase-Change Material (PCM) cell to weight the inputs; since the switching of the PCM cell happens only if a certain power threshold is exceeded, then an output pulse, i.e., spike, is generated only if the weighted sum of the input power exceeds this threshold. Several of these blocks can be stacked together to form larger networks using many-wavelength inputs and outputs. The architecture, proven to be scalable and capable of both supervised and unsupervised learning, promises to offer high-speed and potentially all-optical neuromorphic processing.

Another work proposes to exploit photonic microring resonator weight banks [40]. This type of photonic hardware accelerator showed a three-order-of-magnitude execution time improvement over electronic counterparts. Furthermore, a coherent matrix multiplier is presented in [41], where both inputs and weights are encoded in optical signals. Leveraging this architecture, the system is able to passively perform the linear operation of NNs. The inference phase of this system has been simulated, with an expected energy consumption in the tens of fJ for networks composed of hundreds of neurons. The results proved that this architecture can outperform by 2 to 3 orders of magnitude state-of-the-art CMOS solutions.

Other interesting works concern photonic spiking NNs and photonic reservoir computing. In the first category, one of the most interesting works is [42], where a processor based on a spiking laser neuron suitable for the Broadcast-and-Weight protocol [36] is proposed. This device can process $\sim 10^{12}$ MAC/s operations with an energy efficiency of $\sim$ 270 fJ per MAC operation. Regarding photonic reservoir computing, a nonlinear passive microring resonator architecture is discussed in [43] and its theoretical performance simulated on the delayed XOR task. Reported results showed that for a Bit Error Rate (BER) $< 10^{-3}$ a data rate of 20 Gbps can be obtained with a power consumption of 2.4 mW.

Although all the previously mentioned solutions seem very promising and potentially bring breakthrough changes

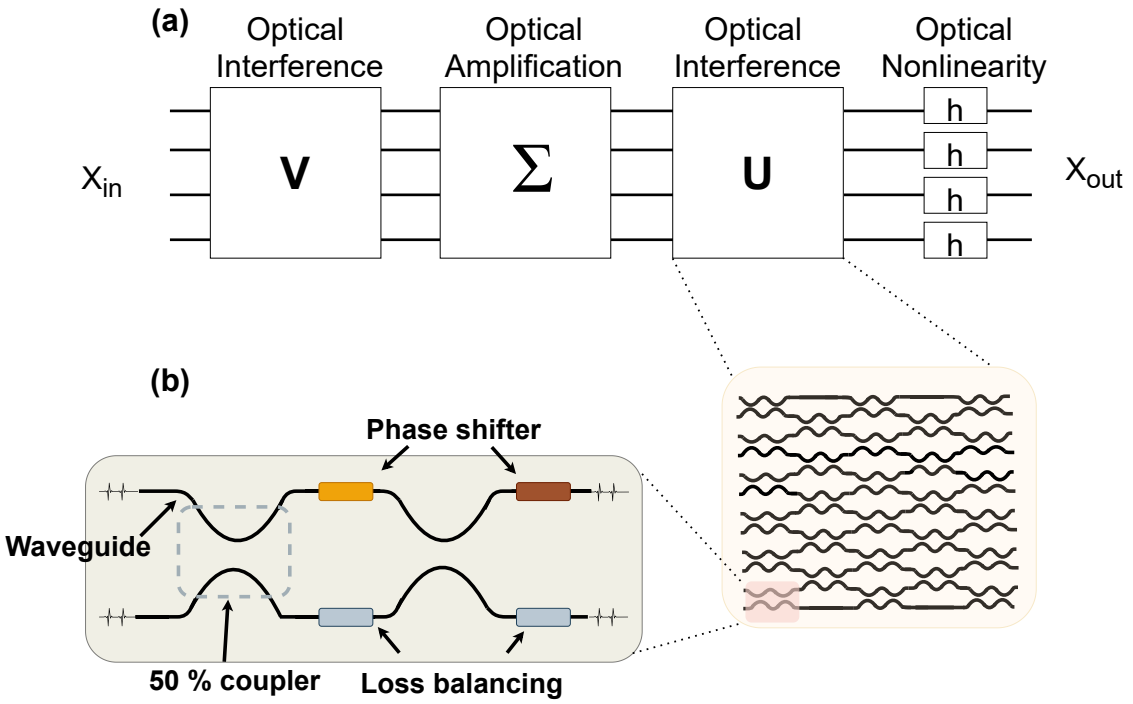

Figure 1: (a) A single layer of the optical NN proposed in [16]; (b) MZI, made of 50% couplers and phase shifters.

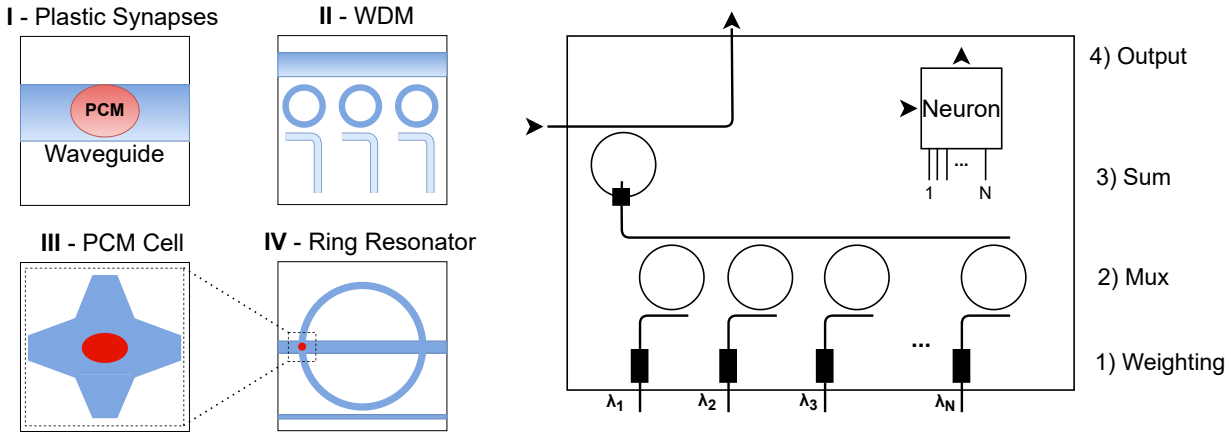

Figure 2: The integrated optical neuron, as proposed in [39], where the inputs are weighted using Phase-Change Material (PCM) cells and summed up using a Wavelength Division Multiplexing (WDM) multiplexer (MUX). (I) PCM cells are used to weigh inputs; (II) inputs are summed using a Wavelength Division Multiplexer on a Micro-ring resonator array; (III) spiking mechanism is enabled by a PCM cell over a ring resonator (IV).

in the field of DNN hardware acceleration, the underlying photonic hardware still shows limitations that need to be addressed. To this aim, in the next section, we describe these constraints and their impacts on the final PANN architectures.

3. Photonic-Aware Neural Networks

Although being one of the most promising accelerators, photonic hardware still presents several limitations, that researchers are trying to address [20]. First of all, performing the training phase in the optical domain is very cumbersome, thus photonic solutions are being investigated aimed at carrying out just the inference phase [33]. The complexity of neural layers supported by photonic technologies is also limited: for both fully-connected and convolutional layers the underlying hardware imposes constraints on the maximum number of multiplications that can be performed [15].

Moreover, photonic accelerators work with analog values, that can in principle vary in a continuous set of values. However, noise and distortions limit the resolution of computations. Furthermore, the introduced noise does not depend on the represented values [44], therefore photonic engines are characterized by essentially constant noise intervals. For this reason, photonic devices can distinguish only a finite number of different equally-spaced levels. This behavior justifies the exploitation of the Effective Number Of Bits (ENOB) to relate the number of distinguishable values to the corresponding number of bits needed for digital storage. For example, an ENOB of 4 corresponds to 16 distinguishable levels. In the context of PNN, the typical bit resolution is very limited i.e., ≤ 6 bits [9, 45, 46], well below the 32-bit floating-point utilized in conventional digital electronic implementations. In particular, such accelerators suffer from a trade-off between attainable operation speed and supported resolution, as well as power consumption. [46].

For this reason, the floating-point type typically exploited for the NN computations cannot be used to simulate the photonic hardware. In [15], we have proposed to use PANN trained with a quantization-aware technique to overcome this issue. Indeed, the exploitation of such a type allows for compliance with the underlying photonic hardware limitations, if defined on the specific bitwidths supported by the accelerators.

Another constraint introduced by photonic accelerators regards the use of positive-valued inputs, as they are typically encoded in the intensity of optical signals. However, the impact of this requirement is lower with respect to the previous one: to comply with this limitation it is necessary to use an activation function that returns positive-only values, such as the sigmoid or the ReLU.

4. CHARLES Library

The aim of CHARLES library is to offer a flexible tool to obtain suited PANN parameters and simulate their behavior. In the following, we discuss the main aspects related to its implementation and the provided functionalities.

4.1. Implementation

CHARLES is a composition of multiple libraries, each one implementing specific functionalities, as depicted in Fig. 3. The lack of the native support of Fixed-Point (FxP) arithmetic in C++ results in the need to exploit a dedicated library. Many available C++ libraries provide custom FxP implementation i.e., fp: Fixed-Point Arithmetic [47], fpm: Fixed-Point Math [48], Compositional Numeric Library (CNL) [49], and many others. All the mentioned libraries offer the possibility to use a base type available in C++ for defining the total number of bits and the corresponding bits reserved for the fractional part (even if none of them allows to work with a total number of bits < 8). CNL is the most up-to-date and supported solution, and provides functionalities that are missing in the others (e.g., saturating arithmetic). Furthermore, CNL gives the possibility to exploit alternative integer types as base types for the FxP type. More precisely, *definitions* and *specializations* are provided for adapting `Boost.Multiprecision` to be used within `cnl::scaled_integer`, with the latter representing FxP numbers. The `Boost.Multiprecision` is part of Boost [50], a set of libraries for C++ that offers support for tasks and structures such as linear algebra, pseudo-random number generation, multithreading, and many more. Specifically, the Multiprecision library provides extended precision arithmetic types for floating point, integer, and rational arithmetic. This extension is pivotal for our target since we need to simulate photonic hardware characterized by a bit resolution ≤ 6 bits.

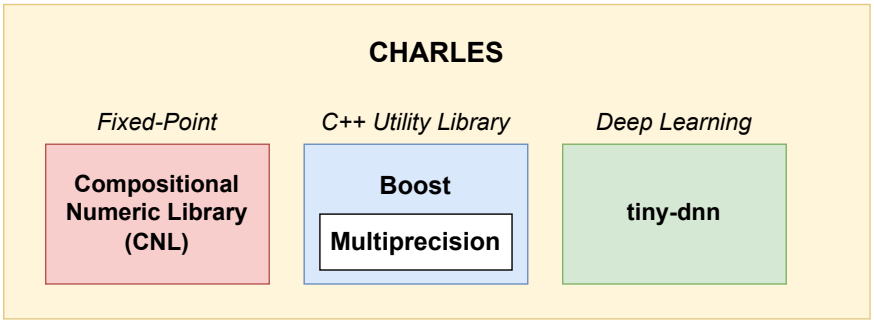

Figure 3: CHARLES structure. The library is composed of three sub-modules, each one aimed at a specific aspect.

```
template <int Digits,int Exponent, class baseType=cnl::signed_multiprecision<Digits>>
using saturated_scaled_integer = cnl::scaled_integer<
    cnl::rounding_integer<
        cnl::overflow_integer<
            baseType, cnl::saturated_overflow_tag>,
            cnl::nearest_rounding_tag>,
    cnl::power<-Exponent>>;
typedef saturated_scaled_integer<TOTAL_DIGITS,FRACTIONAL_DIGITS> float_t;
```

Listing 1: The computation type used in CHARLES.

Finally, the module of CHARLES that implements DL functionalities resorts to tiny-dnn library [51], a DL framework that implements the commonly used neural network layer types, activation functions, loss functions, and optimization algorithms. The fact that tiny-dnn is a header-only library (i.e., without dependencies on external libraries) makes it very suitable to be effectively implemented in a larger library, like CHARLES.

In the realization of CHARLES, the aforementioned libraries have been modified in order to be easily integrated with each other. One key aspect regards the computation type exploited in CHARLES, reported in Listing 1. The FxP type, namely `saturated_scaled_integer`, is a composition of different types provided by both CNL and `Boost.Multiprecision`. We can summarize its main functionalities: (i) it approximates real numbers by using FxP type, i.e., `cnl::scaled_integer`; (ii) it saturates when an overflow occurs, i.e., `cnl::overflow_integer`; (iii) it rounds to the nearest representable value in situations where the actual value cannot be represented, i.e., `cnl::rounding_integer`; (iv) it allows to use a number of bits ≤ 8 by resorting to `cnl::signed_multiprecision`.

Since this type is not compatible with Cereal serialization library [52], a header-only C++ serialization library used in tiny-dnn for saving/loading NN models into files, the definition of external serialization functions has been carried out. This is a relevant feature of the library since it enables cross-benchmarking, i.e., training with one type and testing with another. Serialization is performed exploiting the primitive type `float`, convertible from/to FxP type, as reported in Listing 2. By exploiting these two functions it is possible to save the PANN weights amenable to a photonic implementation.

One last important issue regards the lack of *std* arithmetic functions in CNL, i.e., logarithmic, exponential, trigonometric, and many others, thus preventing to perform the nonlinear activation function of NN models. To solve this problem, the C++ primitive type `float` is once again used, as shown in Listing 3. This method allows to convert to floating-point before exploiting the arithmetic function and then converting back to CHARLES computation type, i.e., float_t. Currently, this approach introduces an additional workload in terms of computation time required to evaluate the activation function. This issue will be overcome in a future release by leveraging CORDIC algorithms [53].

4.2. Functionalities

As already mentioned, our aim is to provide a tool as flexible as possible, while keeping it very easy to use. To achieve this target, CHARLES exploits *templatization* to provide a very easy way to define the number of bits used in

```
namespace cnl{
    template<class Archive,int Digits,int Exponent,
            class baseType=cnl::signed_multiprecision<Digits>>
    void save(Archive & archive,
        saturated_scaled_integer<Digits,Exponent,baseType> const & m){
        archive((float)m);
    }

    template<class Archive,int Digits,int Exponent,
            class baseType=cnl::signed_multiprecision<Digits>>
    void load(Archive & archive, saturated_scaled_integer<Digits,Exponent,baseType> & m){
        float d;
        archive(d);
        m = saturated_scaled_integer<Digits,Exponent,baseType>(d);
    }
}
```

Listing 2: Serialization function implementation for CHARLES computation type.

```
namespace tiny_dnn{
    float_t exp(float_t x){
        float tmp(x);
        return float_t(std::exp(tmp));
    }
}
```

Listing 3: Implementation of missing arithmetic functions for CHARLES computation type.

the computation type, by resorting to the compiler directive `#DEFINE`, which allows to define a *macro*. Specifically, both the total number of bits and the digits reserved for the fractional part of the FxP type are relevant in this context. Thus, it is possible to define two macros, namely `TOTAL_DIGITS` and `FRACTIONAL_DIGITS`, that correspond to these two parameters.

Furthermore, in the pursuit of flexibility, CHARLES offers simple APIs to read images, allowing users to use custom datasets. Functions to read MNIST idx and Cifar-10 binary formats are also available.

Moreover, by resorting to tiny-dnn library, many building blocks are offered by CHARLES, that can be stacked together to build the final NN model. The most relevant ones are *Convolutional*, *Pooling* (i.e., both max and average, with the latter more suited for a photonic implementation), *Activation* (e.g., ReLu, Sigmoid or Tanh), and *Fully-Connected* layers. Each building block is characterized by its parameters; for example, the convolutional layer is characterized by (i) input width and height, (ii) kernel width and height, (iii) number of input and output channels, (iv) padding, and (v) stride.

Finally, resorting to Cereal, it is possible to save the NN model into different representations, i.e., binary encodings, XML, and JSON. One can also choose to save only the weights, the architecture of the NN, or both of them.

5. Experiments

To validate CHARLES and show the potential impact in the PANN simulation scenario, experiments on three common datasets have been performed:

- *Iris* [54]: this dataset contains 3 classes of 50 instances each, where each class refers to a type of iris plant.
- *MNIST* [55]: this dataset, containing handwritten digits from 0 to 9, is composed of 60 000 training images and 10 000 testing images. Each image is a 28×28 8-bit grayscale image.
- *Fashion-MNIST* [56]: this dataset is an alternative to MNIST dataset, containing images of Zalando's articles. The structure of the images and the number of samples of this dataset are the same as the MNIST.

In our experiments, we investigate two different scenarios: one where training is performed in the floating-point domain, i.e., floating-training, and the other one where training is performed in the FxP domain, i.e., fixed-training. Specifically, the floating-training is composed of the following steps: (i) training performed in the floating-point domain to obtain the baseline model; (ii) performance analysis of the model using the test set; (iii) conversion of the baseline model to CHARLES computation types, exploiting different bitwidths; (iv) performance analysis of the different bitwidths using the test set. On the other hand, the fixed-training is done by performing the following steps: (i) training performed using CHARLES high-precision fixed-point computation type to obtain the baseline model; (ii) performance analysis of the model using the test set; (iii) quantization at different bitwidths exploiting CHARLES computation type; (iv) performance analysis of the different bitwidths using the test set [1].

Indeed, due to limitations of state-of-the-art photonic accelerators, it is not currently feasible to perform the training phase of generic NN architectures on actual photonic hardware [20]. This means that the training phase must be simulated in other hardware platforms (i.e., CPU, GPU) to obtain suitable parameters for the inference in the photonic hardware. Thus, the objective of our experiments with CHARLES is to find the most suited training strategy to obtain the best parameters for PANN inference that produce the highest accuracy. For this reason, in the experiments we have considered just the accuracy as performance metric.

5.1. Iris

The Iris dataset is pretty simple and well-studied in the literature; for these reasons, this dataset is very useful to assess the performance of CHARLES. Iris dataset is given in csv format, thus it is necessary to parse it. Furthermore, the dataset has been split into training (80% of the samples) and test sets (the remaining 20% of the samples). To classify samples in this dataset, a simple Fully-Connected PANN architecture has been developed, composed of two layers, containing 20 and 3 neurons, respectively, as depicted in Fig. 4. The sigmoid has been adopted as the activation function. In this architecture the maximum number of multiplications for each neuron is 4, given by the inputs multiplied by the weights in the first layer. In the floating-training, the PANN architecture has been trained

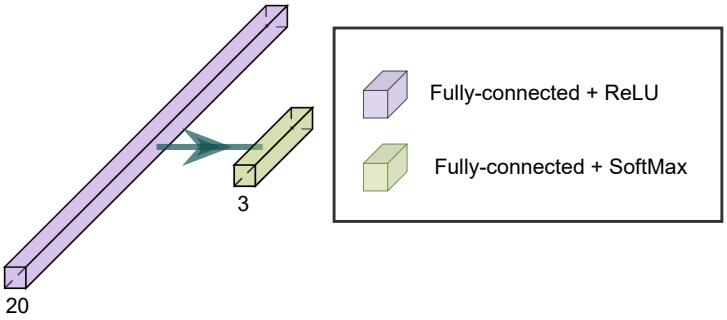

Figure 4: PANN architecture for Iris dataset.

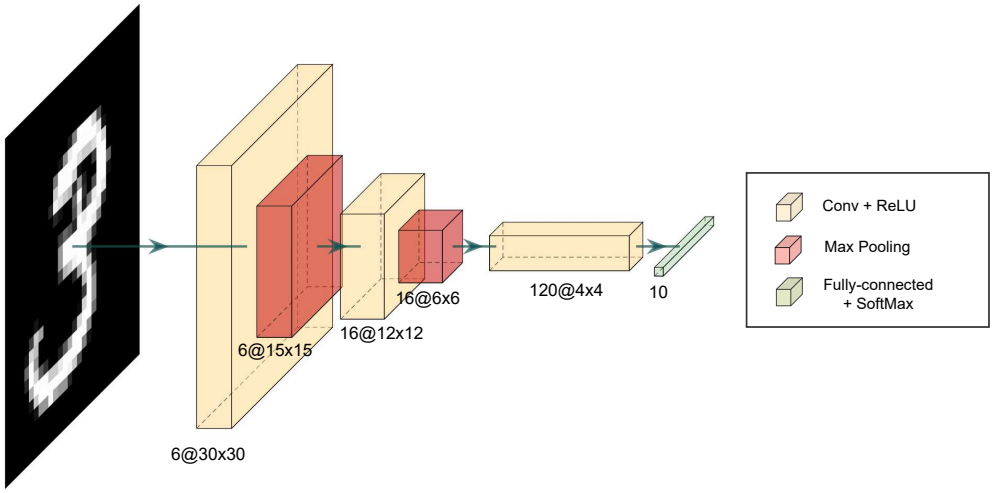

Figure 5: PANN architecture for MNIST/Fashion-MNIST datasets.

for 100 epochs with the following parameters: batch size = 1, gradient descent as optimizer, learning rate = 0.001, and cross-entropy multiclass as loss function. Results are reported in Table 1(a). As expected, a bitwidth reduction

Type	Accuracy
float	96.7%
fixed<32,16>	96.7%
fixed<16,8>	96.7%
fixed<12,6>	96.7%
fixed<8,3>	90.0%
fixed<8,4>	96.7%
fixed<8,5>	96.7%
fixed<8,6>	70.0%
fixed<6,3>	90.0%
fixed<6,4>	73.3%
fixed<6,5>	63.3%

(a) Floating-training.

Type	Accuracy
fixed<41,20>	96.7%
fixed<32,16>	96.7%
fixed<16,8>	96.7%
fixed<12,6>	96.7%
fixed<8,3>	90.0%
fixed<8,4>	96.7%
fixed<8,5>	96.7%
fixed<8,6>	70.0%
fixed<6,3>	90.0%
fixed<6,4>	73.3%
fixed<6,5>	63.3%

(b) Fixed-training.

Table 1: Iris inference results.

leads to a degradation in classification accuracy, as in fixed<6,x> where $x \in \{3, 4, 5\}$, fixed<8,3>, and fixed<8,6>. Nonetheless, the PANN model, even when using low-bitwidth fixed-point (e.g., fixed<6,3>), can reach a pretty high accuracy and, compared to the floating-point baseline model, it does not lose accuracy, as in fixed<8,4> and in fixed<8,5>. Moreover, it is possible to notice how the choice of the number of bits reserved for the fractional part highly influences the final accuracy: for both fixed<6,4> and fixed<6,5> a severe drop in accuracy is observed.

Results of the fixed-training are reported in Table 1(b). The high-precision computation type used for the training phase is fixed<41,20> since it has proven to be the best trade-off between accuracy and time required for training. Furthermore, this can be treated as an ideal case of a noise- and distortion-less analog implementation. The parameters for the training phase are the same as the floating-point baseline model.

The inference results obtained when using the fixed-point baseline are exactly the same as those obtained when using the floating-point baseline. Thus, even in this case, the number of bits reserved for the fractional part highly influences the final accuracy.

5.2. MNIST

Concerning the MNIST dataset, a PANN architecture derived from LeNet-5 [57] has been used. The model, as sketched in Fig. 5, is composed of three convolutional layers, with the first and second layers followed by a pooling layer; to perform classification, a fully-connected layer, made of 10 neurons, is used. Regarding the activation function, the ReLU has been adopted to be compliant with the positive-valued input constraint arising from photonic hardware. In this architecture the heaviest computational burden is given by convolutions, with the largest dimension in 4×4 kernels.

For both fixed-training and floating-training, the PANN architecture has been trained for 20 epochs on the whole training set and evaluated on the test set. Specifically, the parameters used for training are the followings: batch size = 1, gradient descent as optimizer, learning rate = 0.001 and cross-entropy multiclass as loss function. Results of the floating-training are reported in Table 2(a). For larger bitwidths, i.e., fixed<32,16>, fixed<16,8>, and fixed<12,6>, the developed PANN architecture is comparable with the floating-point counterpart. However, when shrinking down the number of bits, a non-negligible accuracy degradation can be observed. In particular, for fixed<6,x> where $x \in \{3, 4, 5\}$ and fixed<8,y> where $y \in \{5, 6\}$, a relevant loss occurs, reaching 82.1% in the fixed<6,5> configuration when compared to the floating-point baseline.

[1] In the following, the fixed<w,f> notation will be used, which represents a FxP number of w bits, with f bits used for the fractional part. For comparison, float represents the baseline model exploiting floating point parameters and computations during the inference phase.

Type	Accuracy
float	98.1%
fixed<32,16>	98.1%
fixed<16,8>	98.1%
fixed<12,6>	98.1%
fixed<8,3>	87.0%
fixed<8,4>	79.8%
fixed<8,5>	46.6%
fixed<8,6>	27.1%
fixed<6,3>	25.4%
fixed<6,4>	24.0%
fixed<6,5>	16.0%

(a) Floating-training.

Type	Accuracy
fixed<41,20>	93.9%
fixed<32,16>	93.9%
fixed<16,8>	93.9%
fixed<12,6>	94.0%
fixed<8,3>	93.6%
fixed<8,4>	92.8%
fixed<8,5>	92.7%
fixed<8,6>	87.3%
fixed<6,3>	90.4%
fixed<6,4>	82.3%
fixed<6,5>	61.1%

(b) Fixed-training.

Table 2: MNIST inference results.

Regarding the fixed-training scenario, results are reported in Table 2(b). Even in this experiment, the high-precision computation type used for the training phase is fixed<41,20>. Inference results of the fixed<41,20>, fixed<32,16>, fixed<16,8>, and fixed<12,6> are slightly lower than the floating-training scenario. Nonetheless, in this context, using FxP type with lower bitwidths does not produce a severe accuracy degradation, as in the previous experiment. Indeed, considering the fixed<6,3> i.e., the best case for the fixed<6,x> configuration, we can observe a very limited accuracy loss i.e., 3.5%. Moreover, the fixed<8,3> inference further lowers this degradation to 0.3%. When compared to the floating-training, the accuracy reached by the best case of fixed<6,x> configuration, i.e., fixed<6,3>, outperforms the same configuration of the floating-training by 65%. The reason for this behavior may be due to the exploitation of the FxP type during the training phase: forcing FxP representation during training may result in NN parameters that are more similar to the numbers representable with the specific inference format (e.g. fixed<6,3>). Thus, when quantizing to these bitwidths, the distance between the actual value and the representable value is smaller, leading to a smaller decrease in accuracy.

Finally, even in this case, as in the Iris experiment, the balance between the number of bits reserved for the integer and the fractional part is an important factor that highly influences the final accuracy. Indeed, we can observe that for fixed<6,x>, we can have very different accuracy, depending on the number of bits allocated for the fractional part i.e., 90.4% for fixed<6,3> and 61.1% for fixed<6,5>.

5.3. Fashion-MNIST

This dataset can be used as a slightly more complex computer vision benchmark for testing CHARLES library compared to MNIST. Concerning the NN, we have used the same PANN architecture as in MNIST experiments, with a batch size of 32 and a number of epochs equal to 30. The other training parameters for both floating-training and fixed-training are the same exploited in MNIST scenario, i.e., learning rate = 0.001, gradient descent as optimizer and cross-entropy multiclass as loss function.

Results of the floating-training experiments are reported in Table 3(a). The model, using floating-point representation, reaches an accuracy of 87.7%; this accuracy does not decrease when using fixed<32,16>, fixed<16,8>, and fixed<12,6>. Although an accuracy degradation can be observed with fixed<8,3>, this drop is still limited. Additionally, when exploiting more bits for the fractional part, as in fixed<8,4>, fixed<8,5>, and fixed<8,6>, the accuracy further decreases. This behavior highlights once again the need for a balance between the number of bits reserved for the fractional and the integer part of the FxP number. Moreover, the accuracy degradation compared to the floating-point baseline when exploiting fixed<6,x> where $x \in \{3, 4, 5\}$ is severe, reaching a drop of 71.5% in the worst case, i.e., fixed<6,5>.

Concerning the fixed-training experiment, results are reported in Table 3(b). By exploiting the high-precision fixed<41,20> representation, an accuracy of 78.7% was obtained, not as high as the floating-training. However, we are mainly interested in the inference phase carried out using lower bitwidths since photonic hardware supports a

Type	Accuracy
float	87.7%
fixed<32,16>	87.7%
fixed<16,8>	87.7%
fixed<12,6>	87.6%
fixed<8,3>	82.2%
fixed<8,4>	79.9%
fixed<8,5>	62.4%
fixed<8,6>	42.9%
fixed<6,3>	50.0%
fixed<6,4>	41.3%
fixed<6,5>	16.2%

(a) Floating-training.

Type	Accuracy
fixed<41,20>	78.7%
fixed<32,16>	78.7%
fixed<16,8>	78.7%
fixed<12,6>	78.3%
fixed<8,3>	76.8%
fixed<8,4>	75.6%
fixed<8,5>	70.7%
fixed<8,6>	53.0%
fixed<6,3>	68.1%
fixed<6,4>	59.9%
fixed<6,5>	21.8%

(b) Fixed-training.

Table 3: Fashion-MNIST inference results.

limited resolution. Regarding the fixed<8,3> and fixed<8,4> configurations, the floating-training performs better than the fixed-training. However, when considering fixed<8,5>, fixed<8,6>, and fixed<6,x> accuracies where $x \in \{3, 4, 5\}$, the fixed-training scenario outperforms floating-training experiments. As a matter of fact, by using fixed<6,3> the accuracy in the fixed-training is around 70%, while it is 50% in the floating-training, showing an increment of 20%. This confirms that the hypothesis made in the MNIST dataset also holds for this scenario: if FxP types are used during the training phase, then the loss due to the quantization in the inference phase is reduced.

6. Discussion

As shown in Table 1, when the floating-training and the fixed-training approaches are compared for the Iris dataset, no difference can be appreciated. However, if we compare the two approaches for more complex datasets, such as MNIST and Fashion-MNIST (reported in Table 2 and Table 3, respectively), we can clearly observe that the fixed-training reaches a better accuracy in all cases, with a lower loss with respect to the baseline when compared to the floating-training. Regarding the trade-off between the number of bits reserved for the integer and fractional parts, the highest accuracy is obtained with the fixed configuration that reserves a number of bits for the fractional part lower or equal to the number of bits reserved for the integer part. This highlights the need for a balance between the representable range and the resolution for the PANN architecture.

The most important results for our aim are those related to low bitwidths, amenable for a photonic implementation. In all the fixed<6,x> configurations (i.e., $x = 3, 4, 5$) the fixed-training outperforms the floating-training, with a mean difference equal to 56.1% and 14.1% for the MNIST and Fashion-MNIST experiments, respectively.

Furthermore, we highlight the likely reasons behind the highest accuracy reached by the fixed-training in most cases (with two exceptions: fixed<8,3> and fixed<8,4> configurations for the Fashion-MNIST) with respect to the floating-training. If we resort to fixed-point representation during the training phase, we are constraining NN parameters to take values that are nearer in the representation to the numbers that we can represent with the specific inference format (e.g., fixed<6,4>). This means that, when we quantize those numbers to smaller bitwidths, the distance between the actual value and the nearest representable value is reduced, thus leading to a smaller decrease in accuracy. This does not happen in the floating-training experiment, where no fixed-point constraints are imposed during the training phase, resulting in most cases (especially for small bitwidths) in a severe loss of accuracy.

7. Conclusion

The exploitation of photonic hardware is one of the most promising solutions for accelerating DL computations, capable of bringing substantial improvements in terms of speed up, power consumption, and footprint reduction. However, these advantages are limited by the constraints arising due to the underlying photonic hardware. To deal

with these limitations, tailored NN architectures and appropriate tools to simulate their behavior are needed. Thus, in this paper, we have introduced CHARLES, a C++ FxP library for PANN. The goal of this library is to serve as a flexible tool to simulate the theoretical behavior of PANN, with the possibility to exploit two different use-cases: one in which the training phase is performed in the floating point domain, i.e., floating training, and the other in which FxP types are used in the learning phase, i.e., fixed-training. In both cases, the inference is performed in FxP representation domain to address PANN constraint. With respect to already available alternatives like Neuroptica and Neurophox, CHARLES is not limited to specific architectures but offers a generalization to the problem of simulating PANN. CHARLES is made publicly available in a GitHub repository [29], including a tutorial on its use.

Furthermore, to show the potential offered by CHARLES three datasets typically used for benchmarking NN models have been considered, i.e., Iris, MNIST, and Fashion-MNIST. For each of these datasets, two experiments have been carried out: floating-training and fixed-training. Results showed no difference in accuracy for Iris dataset; however, when considering more complex datasets and lower bitwidths, such as MNIST and Fashion-MNIST, the fixed-training outperforms the floating-training scenario. Indeed, for both datasets, the fixed<6,x> configurations (i.e., $x = 3, 4, 5$) in the fixed-training experiment outperform the same configurations in the floating-training scenario. Specifically, for MNIST the best fixed<6,x> configuration, i.e., fixed<6,3>, reaches an accuracy of 90.4%, while the floating-point counterpart is able to reach only 25.4%. Moreover, in the Fashion-MNIST, the same FxP configuration in the fixed-training reaches an accuracy of 68.1%, while the floating-training counterpart achieves 50.0%.

This behavior is due to the type exploited in the training phase, namely a floating-point representation or a FxP type with a high number of bits, representing an ideal case. Forcing the FxP representation during training can result in NN parameters that are more similar to the numbers representable with the specific inference format, and thus the quantization process has a smaller impact on accuracy.

Another interesting aspect highlighted by our experiments is the need for a good balance in the FxP representation between the number of bits used for the integer and fractional parts. Indeed, this choice highly influences the final accuracy, as shown in all the experiments. Indeed, for Iris dataset and fixed<6,x> configurations, the accuracy ranges from 90.0% for the fixed<6,3> to 63.6% for the fixed<6.5>. The same behavior can be observed for MNIST and Fashion-MNIST with both floating and fixed training.

CHARLES is a first step towards the simulation of the behavior of PANN. The current release can be fruitfully used to evaluate the theoretical performance of PANN architectures. Future works include the implementation of quantization-aware training algorithms [58, 59] to improve the accuracy obtained in the inference phase. Furthermore, CNL supports vectorized operation for floating-point type. However, the support of this functionality for the FxP type will be added. Finally, the arithmetic functions for the FxP type will be implemented, possibly exploiting CORDIC algorithms [53].

8. Acknowledgements

This work has been partially supported by the project Smart Computing and Communication at the Edge (SMART-COM) funded by Sma-RTy and CNR. This work received funding from the ECSEL JU project BRAINE (grant agreement No 876967). The JU receives support from the EU Horizon 2020 research and innovation programme and the Italian Ministry of Education, University, and Research (MIUR).

References

[1] K. B. Obaid, S. Zeebaree, O. M. Ahmed, et al., Deep learning models based on image classification: a review, International Journal of Science and Business 4 (11) (2020) 75–81.
[2] Z.-Q. Zhao, P. Zheng, S.-t. Xu, X. Wu, Object detection with deep learning: A review, IEEE transactions on neural networks and learning systems 30 (11) (2019) 3212–3232.
[3] R. Perrault, Y. Shoham, E. Brynjolfsson, J. Clark, J. Etchemendy, B. Grosz, T. Lyons, J. Manyika, S. Mishra, J. C. Niebles, The AI index 2019 annual report, AI Index Steering Committee, Human-Centered AI Institute, Stanford University, Stanford, CA (2019).
[4] J. B. Aimone, Neural algorithms and computing beyond Moore's law, Communications of the ACM 62 (4) (2019) 110–110.
[5] R. Wu, X. Guo, J. Du, J. Li, Accelerating neural network inference on FPGA-based platforms—A survey, Electronics 10 (9) (2021) 1025.
[6] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, W. H. Pernice, H. Bhaskaran, C. D. Wright, P. R. Prucnal, Photonics for artificial intelligence and neuromorphic computing, Nature Photonics 15 (2) (2021) 102–114.
[7] V. J. Sorger, Photonic devices, ASICs and systems for machine intelligence, in: Active Photonic Platforms 2022, SPIE, 2022, p. PC121960V.

[8] Y. Masson, B. A. Marquez, B. J. Shastri, Silicon Photonic Neural Networks for Chaos-based Secure Communication, in: 2020 IEEE Photonics Conference (IPC), IEEE, 2020, pp. 1–2.

[9] M. A. Nahmias, T. F. de Lima, A. N. Tait, H.-T. Peng, B. J. Shastri, P. R. Prucnal, Photonic Multiply-Accumulate Operations for Neural Networks, IEEE Journal of Selected Topics in Quantum Electronics 26 (1) (2020) 1–18. doi:10.1109/JSTQE.2019.2941485.

[10] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi, A. Ozcan, All-optical machine learning using diffractive deep neural networks, Science 361 (6406) (2018) 1004–1008.

[11] T. F. De Lima, H.-T. Peng, A. N. Tait, M. A. Nahmias, H. B. Miller, B. J. Shastri, P. R. Prucnal, Machine learning with neuromorphic photonics, Journal of Lightwave Technology 37 (5) (2019) 1515–1534.

[12] Lightelligence, `https://www.lightelligence.ai/`, accessed: 2022-12-12.

[13] Lightmatter, `https://lightmatter.co/`, accessed: 2022-12-12.

[14] Luminous Computing, `https://www.luminous.com/`, accessed: 2022-12-12.

[15] E. Paolini, L. De Marinis, M. Cococcioni, L. Valcarenghi, L. Maggiani, N. Andriolli, Photonic-aware neural networks, Neural Computing and Applications (2022) 1–13.

[16] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, et al., Deep learning with coherent nanophotonic circuits, Nature photonics 11 (7) (2017) 441–446.

[17] T. W. Hughes, M. Minkov, Y. Shi, S. Fan, Training of photonic neural networks through in situ backpropagation and gradient measurement, Optica 5 (7) (2018) 864–871.

[18] N. Passalis, G. Mourgias-Alexandris, A. Tsakyridis, N. Pleros, A. Tefas, Training deep photonic convolutional neural networks with sinusoidal activations, IEEE Transactions on Emerging Topics in Computational Intelligence 5 (3) (2019) 384–393.

[19] M. J. Filipovich, Z. Guo, M. Al-Qadasi, B. A. Marquez, H. D. Morison, V. J. Sorger, P. R. Prucnal, S. Shekhar, B. J. Shastri, Silicon photonic architecture for training deep neural networks with direct feedback alignment, Optica 9 (12) (2022) 1323–1332.

[20] L. De Marinis, M. Cococcioni, P. Castoldi, N. Andriolli, Photonic Neural Networks: A Survey, IEEE Access 7 (2019) 175827–175841. doi:10.1109/ACCESS.2019.2957245.

[21] B. Bartlett, M. Minkov, T. Hughes, I. A. D. Williamson, Neuroptica: Flexible simulation package for optical neural networks, `https://github.com/fancompute/neuroptica` (2019).

[22] S. Pai, I. A. Williamson, T. W. Hughes, M. Minkov, O. Solgaard, S. Fan, D. A. Miller, Parallel fault-tolerant programming of an arbitrary feedforward photonic network, arXiv preprint arXiv:1909.06179 (2019).

[23] C. Yingshi, Optical Neural Networks on PyTorch, `https://github.com/closest-git/ONNet` (2019).

[24] R. Banner, Y. Nahshan, D. Soudry, Post training 4-bit quantization of convolutional networks for rapid-deployment, Advances in Neural Information Processing Systems 32 (2019).

[25] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, K. Keutzer, Zeroq: A novel zero shot quantization framework, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 13169–13178.

[26] L. Wang, X. Dong, Y. Wang, L. Liu, W. An, Y. Guo, Learnable Lookup Table for Neural Network Quantization, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 12423–12433.

[27] G. Mourgias-Alexandris, M. Moralis-Pegios, S. Simos, G. Dabos, N. Passalis, M. Kirtas, T. Rutirawut, F. Y. Gardes, A. Tefas, N. Pleros, A Silicon Photonic Coherent Neuron with 10GMAC/sec processing line-rate, in: Optical Fiber Communication Conference, Optical Society of America, 2021, pp. Tu5H–1.

[28] S. Garg, J. Lou, A. Jain, Z. Guo, B. J. Shastri, M. Nahmias, Dynamic precision analog computing for neural networks, IEEE Journal of Selected Topics in Quantum Electronics 29 (2: Optical Computing) (2022) 1–12.

[29] CHARLES: a C++ Fixed-Point Library for Photonic-Aware Neural Networks, `https://github.com/emiliopaolini/charles`, accessed: 2022-12-12.

[30] K. Shiflett, D. Wright, A. Karanth, A. Louri, PIXEL: Photonic Neural Network Accelerator, in: 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2020, pp. 474–487. doi:10.1109/HPCA47549.2020.00046.

[31] V. Sze, Y.-H. Chen, T.-J. Yang, J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proceedings of the IEEE 105 (12) (2017) 2295–2329.

[32] N. H. Farhat, D. Psaltis, A. Prata, E. Paek, Optical implementation of the Hopfield model, Applied optics 24 (10) (1985) 1469–1475.

[33] G. Wetzstein, A. Ozcan, S. Gigan, S. Fan, D. Englund, M. Soljačić, C. Denz, D. A. Miller, D. Psaltis, Inference in artificial intelligence with deep optics and photonics, Nature 588 (7836) (2020) 39–47.

[34] H.-Y. S. Li, Y. Qiao, D. Psaltis, Optical network for real-time face recognition, Applied Optics 32 (26) (1993) 5026–5035.

[35] A. E.-J. Lim, J. Song, Q. Fang, C. Li, X. Tu, N. Duan, K. K. Chen, R. P.-C. Tern, T.-Y. Liow, Review of silicon photonics foundry efforts, IEEE Journal of Selected Topics in Quantum Electronics 20 (4) (2013) 405–416.

[36] A. N. Tait, T. F. De Lima, E. Zhou, A. X. Wu, M. A. Nahmias, B. J. Shastri, P. R. Prucnal, Neuromorphic photonic networks using silicon photonic weight banks, Scientific reports 7 (1) (2017) 1–10.

[37] H. Bagherian, S. Skirlo, Y. Shen, H. Meng, V. Ceperic, M. Soljacic, On-chip optical convolutional neural networks, arXiv preprint arXiv:1808.03303 (2018).

[38] J. Chang, V. Sitzmann, X. Dun, W. Heidrich, G. Wetzstein, Hybrid optical-electronic convolutional neural networks with optimized diffractive optics for image classification, Scientific reports 8 (1) (2018) 1–10.

[39] J. Feldmann, N. Youngblood, C. D. Wright, H. Bhaskaran, W. H. Pernice, All-optical spiking neurosynaptic networks with self-learning capabilities, Nature 569 (7755) (2019) 208–214.

[40] A. Mehrabian, Y. Al-Kabani, V. J. Sorger, T. El-Ghazawi, PCNNA: A Photonic Convolutional Neural Network Accelerator, in: 2018 31st IEEE International System-on-Chip Conference (SOCC), 2018, pp. 169–173. doi:10.1109/SOCC.2018.8618542.

[41] R. Hamerly, L. Bernstein, A. Sludds, M. Soljačić, D. Englund, Large-scale optical neural networks based on photoelectric multiplication, Physical Review X 9 (2) (2019) 021032.

[42] A. N. Tait, M. A. Nahmias, B. J. Shastri, P. R. Prucnal, Broadcast and weight: an integrated network for scalable photonic spike processing, Journal of Lightwave Technology 32 (21) (2014) 3427–3439.

[43] F. Denis-Le Coarer, M. Sciamanna, A. Katumba, M. Freiberger, J. Dambre, P. Bienstman, D. Rontani, All-optical reservoir computing on a photonic chip using silicon-based ring resonators, IEEE Journal of Selected Topics in Quantum Electronics 24 (6) (2018) 1–8.
[44] T. F. de Lima, A. N. Tait, H. Saeidi, M. A. Nahmias, H.-T. Peng, S. Abbaslou, B. J. Shastri, P. R. Prucnal, Noise Analysis of Photonic Modulator Neurons, IEEE Journal of Selected Topics in Quantum Electronics 26 (1) (2019) 1–9.
[45] L. De Marinis, A. Catania, P. Castoldi, P. Bruschi, M. Piotto, N. Andriolli, A Codesigned Photonic Electronic MAC Neuron with ADC-Embedded Nonlinearity, in: CLEO: Science and Innovations, Optical Society of America, 2021, pp. AW3E–4.
[46] L. De Marinis, A. Catania, P. Castoldi, G. Contestabile, P. Bruschi, M. Piotto, N. Andriolli, A Codesigned Integrated Photonic Electronic Neuron, IEEE Journal of Quantum Electronics (2022) 1–1doi:10.1109/JQE.2022.3177793.
[47] M. Izvekov, fp: Fixed-Point Arithmetic, `https://github.com/mizvekov/fp`.
[48] M. Lankamp, Fixed-point Math Library, `https://github.com/MikeLankamp/fpm`.
[49] J. McFarlane, Compositional Numeric Library (CNL), `https://github.com/johnmcfarlane/cnl`.
[50] J. Maddock, C. Kormanyos, Boost.Multiprecision, `https://github.com/boostorg/multiprecision`.
[51] T. Nomi, tiny-dnn, `https://github.com/tiny-dnn/tiny-dnn`.
[52] iLab University of Southern California, Cereal, `https://uscilab.github.io/cereal/`.
[53] M. Garrido, P. Källström, M. Kumm, O. Gustafsson, CORDIC II: a new improved CORDIC algorithm, IEEE Transactions on Circuits and Systems II: Express Briefs 63 (2) (2015) 186–190.
[54] D. Dua, C. Graff, UCI machine learning repository, `http://archive.ics.uci.edu/ml` (2017).
[55] L. Deng, The MNIST database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine 29 (6) (2012) 141–142.
[56] H. Xiao, K. Rasul, R. Vollgraf, Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms, CoRR abs/1708.07747 (2017). arXiv:1708.07747.
[57] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE 86 (1998) 2278 – 2324. doi:10.1109/5.726791.
[58] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, Y. Zou, Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients, arXiv preprint arXiv:1606.06160 (2016).
[59] M. Rastegari, V. Ordonez, J. Redmon, A. Farhadi, Xnor-net: Imagenet classification using binary convolutional neural networks, in: European conference on computer vision, Springer, 2016, pp. 525–542.