Experimental Implementation of a Neural Network Optical Channel Equalizer in Restricted Hardware Using Pruning and Quantization

Diego Argüello Ron (d.arguello.ron@aston.ac.uk)
Aston Institute of Photonic Technologies, Aston University
Pedro Jorge Freire De Carvalho Sourza
Aston Institute of Photonic Technologies, Aston University
Jaroslaw E. Prilepsky
Aston Institute of Photonic Technologies, Aston University
Morteza Kamalian-Kopae
Aston Institute of Photonic Technologies, Aston University
Antonio Napoli
Infinera
Sergei K. Turitsyn
Aston Institute of Photonic Technologies, Aston University

Research Article

Keywords:

Posted Date: January 12th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1236203/v1

Experimental implementation of a neural network optical channel equalizer in restricted hardware using pruning and quantization

Diego R. Argüello[1,*], Pedro J. Freire[1,2], Jaroslaw E. Prilepsky[1], Morteza Kamalian-Kopae[1], Antonio Napoli[2], and Sergei K. Turitsyn[1,†]

[1]Aston Institute of Photonic Technologies, Aston University, B4 7ET, Birmingham, UK
[2]Infinera, Sankt-Martin-Str. 76, 81541, Munich, Germany
*d.arguello.ron@aston.ac.uk, †s.k.turitsyn@aston.ac.uk

ABSTRACT

The deployment of artificial neural networks-based optical channel equalizers on edge-computing devices is critically important for the next generation of optical communication systems. However, this is a highly challenging problem, mainly due to the computational complexity of the artificial neural networks (NNs) required for the efficient equalization of nonlinear optical channels with large memory. To implement the NN-based optical channel equalizer in hardware, a substantial complexity reduction is needed, while keeping an acceptable performance level. In this work, we address this problem by applying pruning and quantization techniques to an NN-based optical channel equalizer. We use an exemplary NN architecture, the multi-layer perceptron (MLP), and address its complexity reduction for the 30 GBd 1000 km transmission over a standard single-mode fiber. We demonstrate that it is feasible to reduce the equalizer's memory by up to 87.12%, and its complexity by up to 91.5%, without noticeable performance degradation. In addition to this, we accurately define the computational complexity of a compressed NN-based equalizer in the digital signal processing (DSP) sense and examine the impact of using different CPU and GPU settings on power consumption and latency for the compressed equalizer. We also verify the developed technique experimentally, using two standard edge-computing hardware units: Raspberry Pi 4 and Nvidia Jetson Nano.

Introduction

Optical communications form the backbone of the global digital infrastructure. Nowadays, optical networks are the main providers of global data traffic, not only interconnecting billions of people, but also supporting the life-cycle of a huge number of different autonomous devices, control systems, and machines. One of the major factors limiting the throughput of contemporary fiber-optic communication systems is the nonlinearity-induced transmission impairments[1,2], emerging from both the fiber media's nonlinear response and the system's components. The existing and potential solutions to this problem include, e.g., the mid-span optical phase conjugation, digital back-propagation (DBP), and inverse Volterra series transfer function, to mention a few methods[2–4]. It should be stressed that in the telecommunication industry, the competition between possible solutions occurs not only in terms of performance but also in terms of hardware deployment options, operational costs, and power consumption. During the last years, the approaches based on machine learning techniques and, in particular, those utilizing NNs, have become an increasingly popular topic of research, as they can efficiently unroll both fiber and component-induced impairments[5–15].

One of the straightforward ways of using a NN for signal's corruption compensation in optical transmission systems is to plug it into the system as a post-equalizer[7,10,14], a special signal processing device at the receiver side, aimed at counteracting the detrimental effects emerging during the data transmission[16]. A number of NNs architectures have been already studied in different types of optical systems (submarine, long-haul, metro, and access). These architectures include the feed-forward designs such as the MLP[7,10,14,15], considered in the current study, or more sophisticated recurrent-type NN structures[10–12,17]. The predominant number of preceding studies have demonstrated the potential of using the NNs in the optical channel equalization task, especially in terms of the transmission quality improvement[7,8]. However, the practical deployment of real-time NN-based channel equalizers implies that their computational complexity is, at least, comparable, or, desirably lower than that of existing conventional digital signal processing (DSP) solutions[18], and stays the matter of debate. This is a critical bottleneck when deploying these algorithms on modern optical transceivers, because, typically, the good NN performance is linked to the use of a large number of parameters and floating-point operations[10]. A high computational complexity leads to high memory and computing power requirements, increasing the energy and resource consumption[19,20]. Thus, the use of NN-based methods, while being, undoubtedly, promising and attractive, faces a major challenge in optical channel equalization, where the

computational complexity emerges as an important limiting real-time deployment factor[10,12,20,21]. We note here that it is, of course, well known that some NN architectures can be simplified without significantly affecting their performance, thanks, e.g., to strategies such as pruning and quantization[19,20,22–25]. However, their application in the experimental environment of the resource-restricted hardware has not been yet fully studied *in the context of coherent optical channel equalization.* It is also necessary to understand and further analyze the trade-off between the complexity reduction and the degradation of the system performance, as well as the complexity's impact on the end device's energy consumption.

In this paper, we apply the pruning and quantization techniques to reduce the hardware requirements of an NN-based coherent optical channel equalizer, while keeping its performance at a high level. We also emphasize the importance of an accurate evaluation of the equalizer's computational complexity in the DSP sense. Apart from the complexity and inference time analysis, an additional novelty and advance of our work lie in the energy consumption analysis and the study of the impact that the characteristics of both the hardware and the model have on these metrics.

Results

We develop and experimentally demonstrate a low-complexity NN-based equalizer that can be deployed on resource-constrained hardware and, at the same time, can successfully mitigate nonlinear transmission impairments in optical communication systems. This is achieved by applying the pruning and quantization methods to the NN architecture[23], and by studying the optimal trade-off between the complexity of the solution and its performance. The obtained results can be split into three main categories.

First, we quantify how complexity reduction techniques affect the performance of the NN model and establish a compression limit for optimal performance versus complexity trade-off. Second, we analyze the computational complexity of the pruned and quantized NN-based equalizer in terms of DSP. Finally, we experimentally evaluate the impact that the characteristics of the hardware and the NN model have on the signal processing time and energy consumption by deploying the latter on both Raspberry Pi 4 and Nvidia Jetson Nano.

Now we briefly review the previous results in the field of compression techniques applied to NN-based equalizers in optical links, to underline the novelty of our current approach. The use of these techniques to reduce the NNs complexity in optical systems is, clearly not a new concept[25]. However, the compression methods have recently gained a new wave of attention due to the question of how realistic is the hardware implementation of NN-based equalizers in optical transmission systems. In a direct detection transmission system, a parallel-pruned NN equalizer for a 100-Gbps PAM-4 link was tested experimentally using the enhanced version of the one-shot pruning method[26], which decreased by 50% the resource consumption without significant performance degradation. When considering coherent optical transmission, the complexity of the so-called learned DBP nonlinearity mitigation method was reduced by pruning the coefficients in the finite impulse response filters[27] (see more technical explanations in the Methods section below). In that case, using a cascade of three filters, a sparsity level of around 92% can be achieved with a negligible impact on the overall performance. Recently, some advanced techniques for avoiding multiplications in such equalizers using additive powers-of-two quantization were tested[28]. In that work, 99% percent of the weights could be removed using advanced pruning techniques, and instead of multiplications, just bit-shift operations were required. However, none of those works deal with the experimental demonstration of hardware implementation, and our study addresses exactly the latter problem.

So, unlike previous works, in the current study, we implement the compressed NN-based equalizer for the coherent optical channel in two different hardware platforms: Raspberry Pi 4 and Nvidia Jetson Nano. We also evaluate the impact of the compression techniques on the system's latency for each hardware type and study the performance-complexity trade-off. Finally, we carry out an analysis of energy consumption and of the impact that the characteristics of the hardware and the NN model have on it.

Optical Communication System and Equalizer Design

To address the use of an MLP as a NN-based equalizer, an accurate measurement system for both the inference time and the power consumption, on both the Raspberry Pi and the Nvidia Jetson Nano, has been designed, so that the effects that pruning and quantization have on these metrics can be characterized. In Refs.[10,14] the non-compressed MLP post-equalizer was considered, and it was shown that it can successfully compensate for the nonlinearity-induced impairments in a coherent optical communication system. We analyze the equalizer's performance in terms of the standard achieved Q-factor using the simulated data for a 0.1 root-raised cosine (RRC) dual-polarization signal, with 30 GBd, and 64-QAM modulation, for the transmission over the 20×50 km links of standard single-mode fiber (SSMF). We have used the same simulator as described in Refs.[10,29], to generate our training and testing datasets, and the same procedure to train the NN-based equalizer. In our configuration, the NN is placed at the receiver (Rx) side after the DSP block, see Fig. 1. The results for three launch power levels are analyzed: 0 dBm, 1 dBm, and 2 dBm.

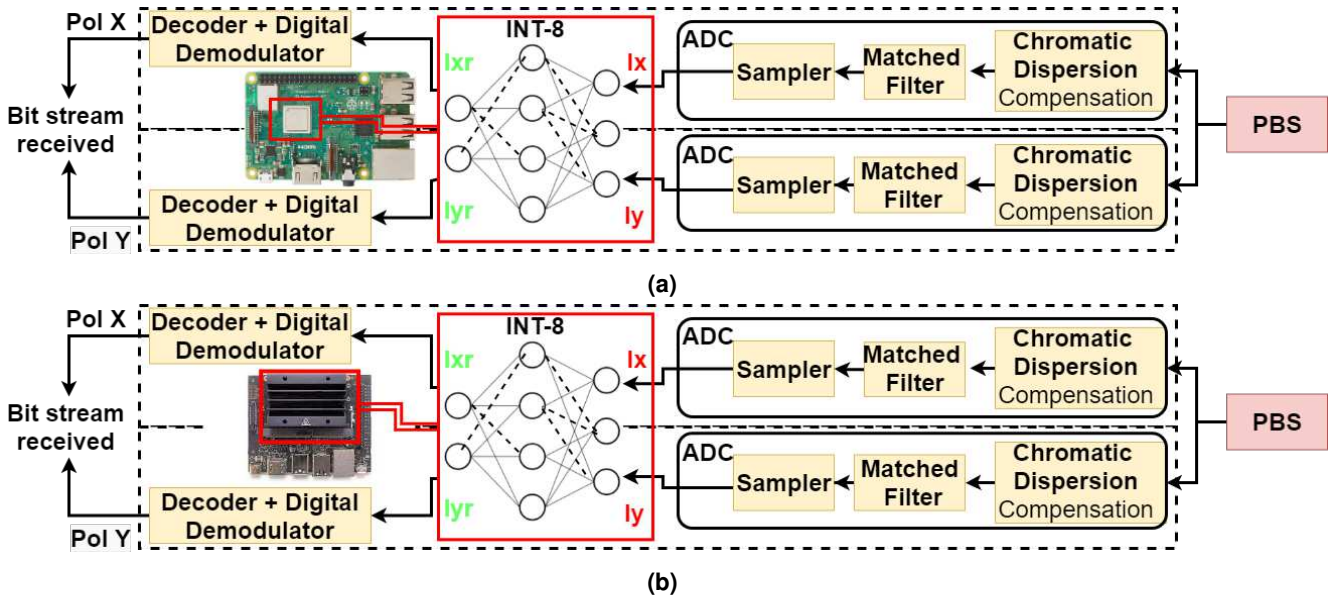


Figure 1. Structure of a communication channel that is equalized using a pruned and quantized neural network deployed on a Raspberry Pi (a) and on a Nvidia Jetson Nano (b).

The hyperparameters that define the structure of the NN are obtained using a Bayesian optimizer[10,30], where the optimization is carried out concerning the signal's restoration quality performance, similar to Ref.[10]. The resulting optimized MLP has three hidden layers (we did not optimize the number of layers, but only the number of neurons and the activation type), with 500, 10, and 500 neurons, respectively. The "tanh" is used as an activation function and no bias is employed. The length of the equalizer's input (number of taps) used in this paper is 10. The NN was subjected to pruning and quantization after it has been trained and tested. We analyzed the performance of different NN models depending on their sparsity level; the latter ranges from 20% to 90%, with a 10% increment. The weights and activations are quantized, converting their data type from 32-bit single-precision floating-point (FP32) to 8-bit integer (INT8). This was carried out to enable a real-time use of the model once deployed on resource-constrained hardware. The final system is described in Fig. 1. The inference process (the signal equalization) was, first, carried out using an MSI GP76 Leopard personal computer, equipped with an Intel® Core™ i9-10870H processor, 32GB of RAM as well as a GPU Nvidia RTX2070. The results obtained on this computer were used as a benchmark and compared to those obtained on two small single-board computers: the Raspberry Pi 4 and the Nvidia Jetson Nano.

Finally, the NNs were developed using the open-source platform for machine learning known as Tensorflow. Moreover, the pruning and quantization techniques were implemented employing the TensorFlow Model Optimization Toolkit — Pruning API and TensorFlow Lite[31]

Compressing Process for Neural Network Equalizers

When designing a NN, the traditional approach consists in using dense and over-parameterized structures, as it provides improved performance and learning capabilities. This is due to the smoothing effect on the loss function, which benefits the convergence of the gradient descent techniques used to optimize it. However, some precautions must be taken while training an over-parametrized model, because such models often tend to overfit, and their generalization capability can be degraded[32,33].

The good performance achieved due to the over-parametrization comes at the cost of larger computational and memory resources. This also results in a longer inference time and higher energy consumption. Note that these costs are a consequence of the parameter redundancy and a large number of floating-point operations[20,23]. Therefore, the capabilities of high-complexity NN-based equalizers do not translate yet into end-user applications on resource-constrained hardware. As a consequence, reducing the gap between the algorithmic solutions and the experimental real-world implementations is an increasingly active topic of research. During the past several years, noticeable efforts have been invested into developing the techniques that can help to simplify NNs without significantly decreasing their performance. This simplification process is known as "NNs compression", and the most common approaches are: down-sizing the models, factorizing the operators, quantization, parameter sharing or pruning[20,23,24]. When these techniques are applied, the final model becomes less complex, and therefore, its latency, or time it takes to make a prediction, decreases, which also results in a lower energy consumption[20]. In this work, we focus on both pruning and quantization for compressing our NN equalizer and quantify a trade-off between complexity reduction and

system performance, see the Methods section for a detailed description of both approaches.

Performance vs. Compression Trade-off

Firstly, we note that the complexity reduction for an equalizer must not affect its performance drastically, i.e. the system's performance is still required to be within an acceptable range. In Fig. 2a, the Q-factor achieved by the NN equalizer is depicted versus different sparsity values, for three launch power levels: 0 dBm, blue; 1 dBm, red; and 2 dBm, green. The results are shown using dotted lines and stars, which are those obtained on the PC, Raspberry Pi, and Nvidia Jetson Nano, using the pruned and quantized model. For each of these launch powers, two baselines for the Q-factor are depicted: one corresponds to the level achieved by the uncompressed model, defined by straight lines, while the other provides the benchmark when we do not employ any NN equalization and use only standard linear chromatic dispersion compensation (LE, linear equalization); the latter levels for the three different launch powers are marked by dotted lines having the appropriate colors.

Fig. 2b quantifies the impact that each compression technique has on the performance: in that figure, we plotted the Q-factor achieved by the NN equalizer versus different values of sparsity, for the 1 dBm launch power. The blue and red straight lines represent the Q-factor of the original model and the Q-factor achieved by it after being quantized. The dotted lines with asterisks, show the performance of a model that has been only pruned (blue), and the performance in the case of both pruning and quantization (red). It is seen that a visible reduction of the complexity can be achieved without a dramatic degradation of the performance. The points of fast deterioration of the performance are also clearly seen in this figure.

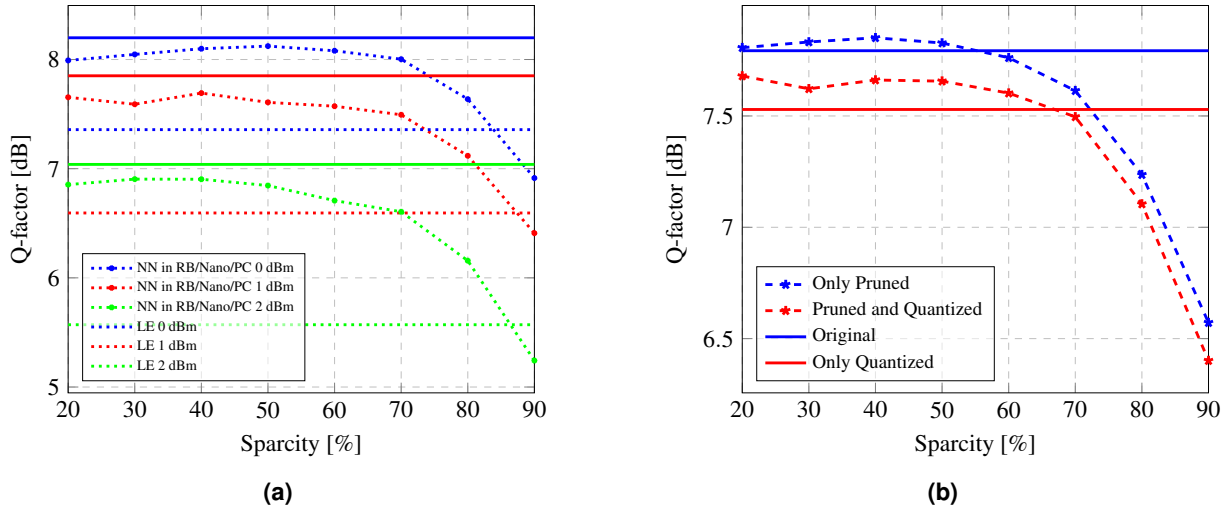


Figure 2. a) Q-factor achieved for pruned and quantized models versus the level of sparsity for datasets corresponding to three launch powers: 0 dBm, 1 dBm, and 2 dBm; The solid lines correspond to the Q-factor achieved by the original model. The dashed lines show the Q-factor when only linear equalization (LE) is implemented. b) Q-factor achieved after pruning compared to the one achieved after both pruning and quantization, for different levels of sparsity and for a dataset corresponding to the 1 dBm launch power. The blue and red solid lines correspond to the Q-factor achieved by the original model and the one achieved by this model after quantization, respectively.

First, it is clear that the results of the hardware implementation and PC are in agreement, indicating that the programming part was correct and identical on all devices. In addition, it can be observed from Fig. 2a that the quantization and pruning process does not cause a significant performance degradation until the 60% of sparsity is reached, with just a 4% performance reduction. However, when we move to sparsity levels around 90%, the performance is close to the one achieved using a linear equalization (i.e., the Q-factor curves drop down to the levels marked with dashed lines).

It is possible to appreciate that the decrease in performance is mainly an effect of the quantization process when the levels of sparsity are above 60%. A drop in the Q-factor value of at least 2.5% has been measured when quantizing an already pruned model. Once the levels of sparsity are higher than 60%, the reduction in performance is accelerated. Moreover, we observe that some degree of sparsification improves the performance with respect to the unpruned model, as had already been reported [32].

Computational Complexity Analysis

Fig. 3 depicts the reduction in the size of the model as well as the model's computational complexity for different sparsity values, after having applied quantization (for the computational complexity definitions, see subsection Computational Complexity Metrics in Methods). Overall, we have achieved an 87.12% reduction in the memory size after pruning 60% of the NN equalizer

weights and quantizing the remaining ones. As a consequence, the size of the model went down from 201.4 to 25.9 kilobytes. With respect to the decrease of the model's computational complexity, it goes from 54272000 to 4582400 bit operations (BoPs) after applying the same compression strategy, which is a 91.5% reduction (see the explicit BoPs definition in the Methods section). We would like to point out once more that sparsity levels of 60% can be reached without a substantial performance loss. Therefore, approximately the same high level of performance can be achieved with a model that is significantly less complex than the initial NN structure, which is one of the main findings of our work.

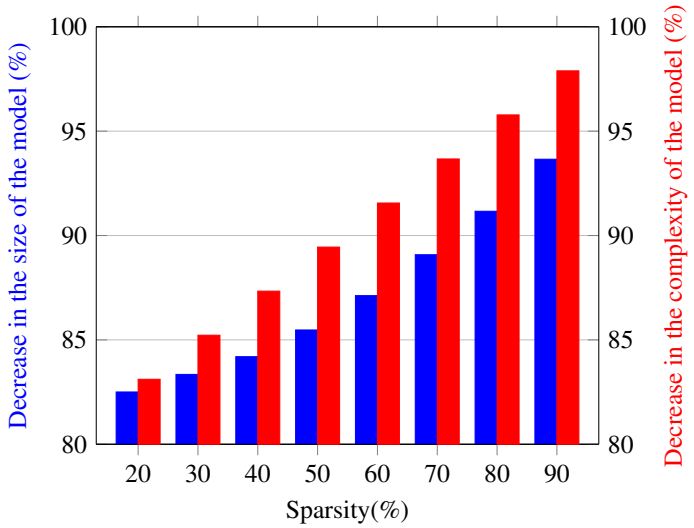


Figure 3. Complexity and size reduction achieved via pruning and quantization for different levels of sparsity.

It is worth mentioning the individual impact that quantization and pruning have on the computational complexity of the model. When the computational complexity is calculated for a quantized, but unpruned model, the BoPs value obtained is equal to 11456000. Therefore, if this value is compared with the already mentioned 54272000 BoPs for the unpruned and unquantized NN, a reduction in complexity of a 78.8% is obtained thanks to quantization. As it can be seen in Fig. 3, the remaining gain comes from the pruning technique, and it grows linearly as indicated in Eq. (2).

Online Latency Evaluation

Numerous deep learning applications are latency-critical, and therefore the inference time must be within the bounds specified by service level objectives. Optical communication applications that employ deep learning techniques are a good example of this. Note that the latency is highly dependent on the NN model implementation and the hardware employed (e.g., FPGA, CPU, GPU). Even though we utilized the same code based on TensorflowLite in all three scenarios, the purpose of our work is to demonstrate the impact of the compression techniques on the latency reduction; please refer to the Methods section for more details on the devises' inference time measurements.

Fig. 4 shows the latency of the considered NN model before and after quantization. In this case, the latency is the time to process one symbol: we have averaged it over 30k symbols. With the quantized model, we observe approximately a 56% reduction in latency for all three values of power, when compared to the original model. For the latency, consideration pruning is not taken into account here because TensorflowLite does not support sparse inference yet, which makes the algorithm still use the same amount of cache memory and so the pruning does not affect the latency. Also, we could observe that Raspberry Pi has the longest inference time among our devices. This is in line with the fact that Raspberry is designed as a low-cost and general-purpose single-board computer[34]. On the other hand, the Nvidia Jetson Nano was developed with GPU capabilities, which makes it more suitable for deep learning applications, allowing us to achieve lower latencies.

Online Energy Consumption Evaluation

Within the context of edge computing, not only is speed an important factor, but also power efficiency. In this work, the metric used to evaluate the energy consumption and compare the different types of hardware for the coherent optical channel equalization task is the energy per recovered symbol. When using a quantized model with a pruning level of 60%, the average energy consumed during inference for the Raspberry Pi 4 and the Nvidia Jetson Nano is 2.98 W and 3.03 W, respectively. On the other hand, if the original model is employed, there is an increase in energy consumption of around 3%, which is congruent with the findings in previous works[23]. Thus, during inference, the Raspberry Pi 4 consumes 3.06 W and the Nvidia Jetson Nano

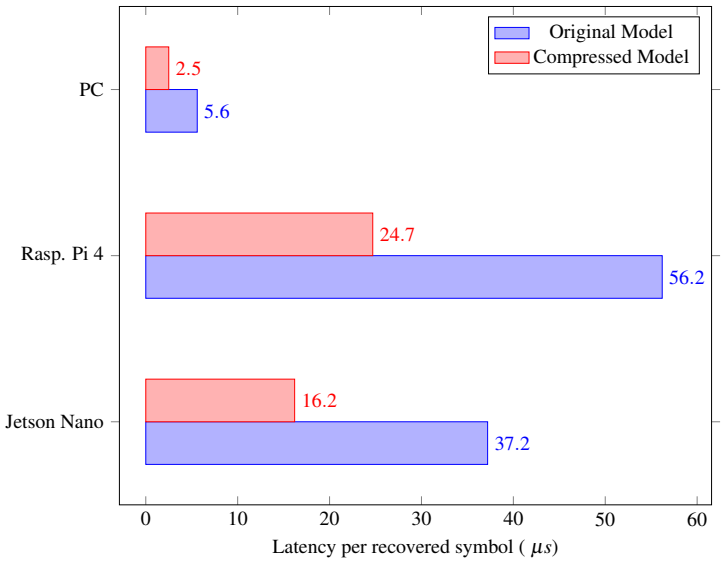
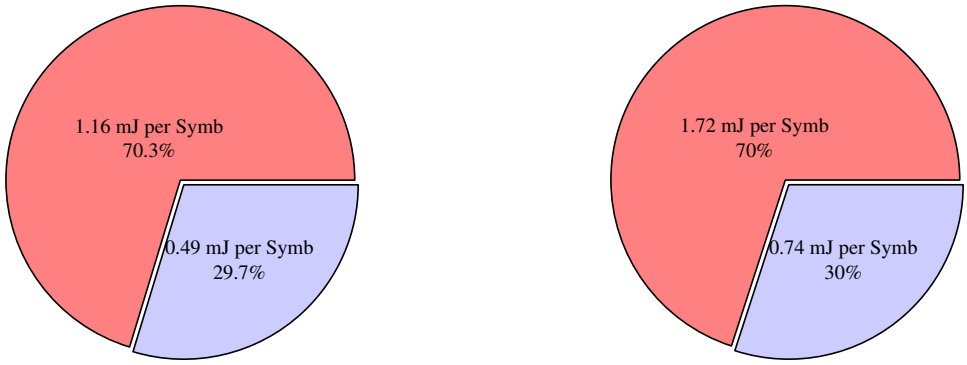


Figure 4. Summary of the symbol processing (inference) time for compressed NN models (after pruning and quantization) and the original models for three devices under evaluation: Raspberry Pi 4, Jetson Nano, and a standard PC.

3.13 W, respectively. Multiplying these values by the NN processing times per recovered symbol reported in Fig. 4, we obtain the results presented in Fig. 5. We note that Raspberry Pi has the highest energy consumption per recovered symbol. This is a consequence of the lack of a GPU, which results into longer inference times. Thus, Nvidia Jetson Nano consumes 33.78% less energy than the Raspberry Pi 4. Regarding pruning and quantization, the use of these techniques allows an energy saving of 56.98% for the Raspberry Pi 4 and a 57.76% saving for the Nvidia Jetson Nano. See the Methods section for more details on the devices' energy consumption measurement.



(a) Nvidia Jetson Nano

(b) Rasp. Pi 4

Figure 5. Energy consumption for Raspberry Pi 4 and Nvidia Jetson Nano. The blue section represents the energy consumption per recovered symbol when using the compressed model, and its relative energy cost expressed as a percentage with respect to the sum of the energy consumed by both the original and compressed models. Likewise, the red section describes the energy consumption per recovered symbol when using the original model and its relative energy cost.

Discussion

We investigated how we can use pruning and quantization to reduce the complexity of the hardware implementation of an NN-based channel equalizer in a coherent optical transmission system. With this, we tested the implementation of the designed equalizer experimentally, using the Raspberry Pi 4 and the Nvidia Jetson Nano. It was demonstrated that it is possible to reduce the NN's memory usage by 87.12% and its computational complexity by 91.5% without any serious penalty in performance, thanks to the two aforementioned compression techniques.

Moreover, the effect of using different types of hardware was characterized by measuring the inference time and energy consumption in both the Raspberry Pi 4 and the Nvidia Jetson Nano. It has been demonstrated that the Nvidia Jetson Nano allows 34% faster inference times than the Raspberry Pi, and that, thanks to the quantization process, a 56% inference time reduction can be achieved. Finally, thanks to the use of pruning and quantization techniques, we achieve a 56.98% energy savings for the Raspberry Pi 4 and a 57.76% for the Nvidia Jetson Nano; we also observed that the latter device consumes 33.78% less energy.

Overall, our findings demonstrate that the usage of pruning and quantization can be suitable strategies for the implementation of NN-based equalizers that are efficient in impairments mitigation in high-speed optical transmission systems when deployed on resource-restricted hardware. We believe that these model compression techniques can be used for the deployment of NN-based equalizers in real optical communication systems, and for the development of novel online optical signal processing tools. We hope that our results can also be of interest to the researchers developing sensing and laser systems, where the application of machine learning for field processing and characterization is a rapidly developing area of research[35].

Methods

Pruning Technique

With pruning, the redundant NN elements can be removed to sparsify the network without limiting much its ability to carry out a required task[24,32,36]. Thus, the networks with lower computational complexity and size are obtained, resulting in lower memory requirements as well as reduced prediction times[23,24]. Furthermore, the sparsification acts as a regularization technique, improving the model quality by helping to reduce over-fitting[32]. On the other hand, retraining an already pruned NN can help to escape local loss function minima, which can lead to a better prediction accuracy[24]. Finally, a significant model compression can often be achieved without a noticeable effect on the NN's performance[32].

Depending on what is going to be pruned, the sparsification techniques can be classified into two types: model sparsification and ephemeral sparsification[32]. In the first case, the sparsification is permanently applied to the model, while in the second case, the sparsification only takes place during the computing process. In our work, we will use the model sparsification, because of the effects it has on the final NN's computing and memory hardware requirements. Adding to this, the model sparsification can consist in removing not only weights but also larger building blocks, such as neurons, convolutional filters, etc.[32]. Here we apply the specification to just the weights of the network, for the sake of simplicity and as far as it matches the NN structure (the MLP) that is considered.

After having defined what to prune, it is necessary to define when the pruning occurs. Based on this, there are two main types of pruning: static and dynamic[24]. In the static case, the elements are to be removed from the NN after the training, and in this work, to demonstrate the resulting effect, we use the static pruning variant because of its simplicity.

The static pruning is generally carried out in three steps. First, we decide upon what requires to be pruned. A simple approach to define the pruning objects can be to evaluate the NN's performance with and without particular (pruned) elements. However, this poses scalability problems: we have to evaluate the performance when pruning each particular NN's parameters, and there may be millions of these.

Alternatively, it is possible to select the elements to be removed randomly, which can be done faster[32,37,38]. Following this latter approach, we beforehand prune the weights. Once it has been decided which elements are to be pruned, it is necessary to establish the criteria for how the elements are to be removed from the NN, ensuring that the high levels of sparsity are achieved without a significant loss in performance. When pruning the weights of the NN, it is possible to remove them based on different aspects: considering their magnitude (i.e., the weights having values close to zero are to be pruned, with the pruning percentage is defined by the sparsity level we aim to achieve), or their similarity (if two weights have a similar value, only one of those is kept); we mention that the other selection procedures also exist[32,38]. Here, we pick the relatively simple weights pruning strategy based on their magnitude. In Fig. 6 we show the impact when we have pruned our NN equalizer by 40%. When comparing the weight distributions of the original and pruned models, it is clear that the sparsity level defines the number of weights that need to be pruned. Thus, the pruning process starts by removing the smallest weight and continues until the desired sparsity level is reached. Finally, a retraining or fine-tuning phase should be done, to reduce the degradation in the modified NN performance[24].

Quantization Technique

Besides the reduction in the number of operations involved in the NN signal processing, the precision of such arithmetic operations is yet another crucial factor when determining the model's complexity and, therefore, the inference latency, as well as equalizer's memory and energy requirements[23,39–41]. The process of approximating a continuous variable with a specified set of discrete values is known as quantization. The number of discrete values will determine the number of bits necessary to represent the data. Thus, when applying this technique in the context of deep learning, the objective is in reducing the numeric precision used to encode the weights and activations of the models, but avoid a noticeable decrease in NN's performance[20,41].

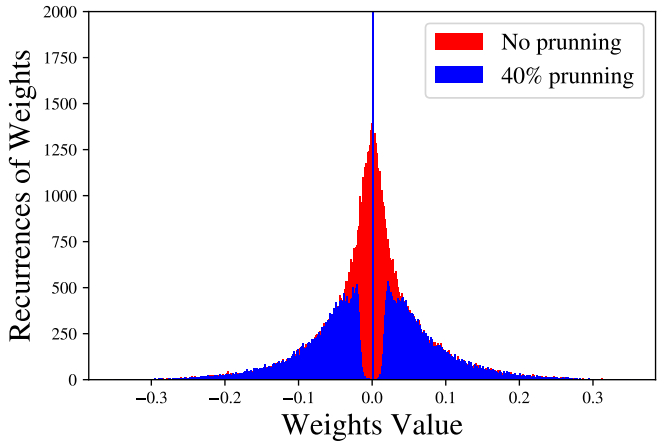


Figure 6. A typical distribution of the weights of the NN-based MLP equalizer without pruning and with pruning when the sparsity level is set to 40%.

Using low-precision formats allows us to speed up math-intensive operations, such as convolution and matrix multiplication. On the other hand, the inference (signal processing) time depends not only on the format representation of the digits involved in the mathematical operations but is also affected by transporting the data from memory to the computing elements. Moreover, heat is generated during the latter process and, therefore, using a lower-precision representation can result in energy savings. Finally, another benefit of using low-precision formats is that a reduced number of bits is needed to store the data, which reduces the memory footprint and size requirements[23,41].

FP32 has been traditionally used as the numerical format for encoding weights and activations (output of the neurons) in a NN, to take advantage of a wider dynamic range. However, as it has already been mentioned, this results in higher inference times, which is an important factor when a real-time signal processing is considered[20]. A variety of alternatives to the FP32 numerical format for NN's elements representation have been proposed lately, to reduce the inference time, as well as to decrease the hardware implementation requirements. For example, it is becoming popular to train NNs in FP16 formats, as it is supported by most deep learning accelerators[20]. On the other hand, math-intensive tensor operations executed on INT8 types can see up to a 16× speed-up compared to the same operations in FP32. Moreover, memory-limited operations could see up to a 4× speed-up compared to the FP32 version[22–24,41]. Therefore, in addition to pruning, we will reduce the precision of the weights and activations to further decrease the computational complexity of the equalizer, employing the technique known as integer quantization[41].

The integer quantization maps a floating point value $x \in [\alpha, \beta]$ to a bit integer $x_q \in [\alpha_q, \beta_q]$. This mapping can be defined mathematically using the following formula: $x_q = \text{round}\left(\frac{1}{s}x + z\right)$, where s (a positive floating point number) is known as the scale, and z is the zero point (an integer). The previous equation can be refactored in order to take into account that x can be outside the range $[\alpha, \beta]$, and therefore x_q is outside of $[\alpha_q, \beta_q]$. Thus, it is necessary to clip the values that are outside the range; as a consequence, the mapping formula becomes: $x_q = \text{clip}(\text{round}\left[\frac{1}{s}x + z\right], \alpha_q, \beta_q)$, where the clip function takes the values[24,42]:

$$\text{clip}(x, l, u) = \begin{cases} l & \text{if } x < l, \\ x & \text{if } l \leq x \geq u, \\ u & \text{if } x > u. \end{cases}$$

Integer quantization can take different forms, depending on the spacing between quantization levels, the symmetry of the clipping range (determined by the value of the zero-point z)[42]. For the sake of simplicity, in this work, we used symmetric and uniform integer quantization.

The quantization process can occur after the training or during it. The first case is known as post-training quantization (PTQ) and the second one is the quantization aware training[22–24]. In PTQ, a trained model has its weight and activations quantified. After this, a small unlabeled calibration set is used to determine the activations' dynamic ranges[23,41–43]. No retraining is needed, which makes this method very popular because of its simplicity and as it needs fewer data[42,43]. Nonetheless, when a trained model is directly quantized, this may perturb the trained parameters, moving the model away from the convergence point reached during the training with a floating-point precision. In other words, we notice that the PTQ can have accuracy-related issues[42].

In this work, the quantization is carried out after the training stage, i.e., we use the PTQ, in addition to a fine-tuning process from FP32 to INT8. Therefore, we present here the design of a compressed NN-based channel equalizer, demonstrating that it is possible to achieve a good balance between the performance of the model and its hardware requirements. Additionally, we demonstrate that even with relatively straightforward NN compression techniques, such NN-based equalizers can be deployed on resource-constrained hardware, by performing the NN-based optical signal processing with Raspberry Pi 4 and Nvidia Jetson Nano. We also explicitly show the impact that this process has on the device's energy consumption and inference time.

Computational Complexity Metrics

Finally, it is important to discuss how we can correctly evaluate the computational complexity of such models. In this regard, we quantitatively evaluate the reduction of computation complexity achieved by applying pruning and quantization, calculating the number of bits used during an inference step. The most common operations in a NN are multiply-and-accumulate operations (MACs). These are operations of the form $a = a + w \cdot x$, where three terms are involved: firstly, x corresponds to the input signal of the neuron; secondly, w refers to the weight; and, finally, the accumulate variable a[44]. Traditionally, the network complexity arithmetic has been measured using the number of MAC operations. However, in terms of the DSP processing, the number BoPs is a more appropriate metric to describe the computational complexity of the model[22,45], as different data types for the network weights and activations are used. Thus, in this work, we use BoPs to quantify the NN equalizer complexity.

Considering the effect of unstructured pruning, BoPs – in a pruned fully connected layer – can be defined as:

$$\mathrm{BoPs}_i = m_i n_i [(1 - f_{p_i}) b_{a_i} b_{w_i} + b_{a_i} + b_{w_i} + \log_2(n_i)], \tag{1}$$

where b_{w_i} and b_{a_i} are the bit width of the weights and activations of the i-th layer, n_i and m_i the number of inputs and outputs, f_{p_i} the fraction of pruned layer weights, and for this example $i \in [1,2,3]$. Therefore, the arithmetic complexity of the model is:

$$\mathrm{BoPs} = \sum_i \mathrm{BoPs}_i.$$

It is worth noticing that the tool used in this work to carry out pruning is the Model Optimization API provided by Tensorflow[31], and that the pruning is carried out uniformly, meaning that we have an equal f_{p_i} for $i \in [1,2,3]$. Therefore, the overall number of BoPs that we have for the three-layer MLP is given as follows:

$$\mathrm{BoPs}_{\mathrm{MLP}} = (n_i n_1 b_i + n_1 n_2 b_a + n_2 n_3 b_a + n_3 n_o b_a)(1 - f_{p_i}) b_w. \tag{2}$$

Taking into account the above definition of the BoPs metric, it can be seen that we gain a reduction in multiplication operations because of pruning. Moreover, BoPs are quadratically dependent on the bit widths and linearly dependent on the pruning fraction. Therefore, the quantization will have a more pronounced reduction effect when we aim at reducing the complexity of our NN equalizer.

Memory and Processor Restricted Hardware

In many deep learning applications, low power consumption and a reduced inference time are especially desirable characteristics. Moreover, the use of graphics processing units (GPU) to attain high performance has some costs-related issues which are far from being ultimately solved[34,46]. Therefore, small, portable, and low-cost hardware is desired to bring a solution to this problem. As a result, single-board computers have become popular, e.g., Raspberry Pi 4 and Nvidia Jetson Nano are among the most used ones[34]. Hence, here we analyze the functioning of our NN-based equalizer using these two aforementioned popular hardware types.

Raspberry Pi

Raspberry Pi is a small single-board computer. It is equipped with a Broadcom Video Core VI (32-bit) GPU, Quad-core ARM CortexA72 64-bit 1.5 GHz CPU, 2 USB 2.0 ports and 2 USB 3.0 ports; for data storage, it uses a MicroSD card. Moreover, connections are provided through a Gigabit Ethernet / WiFi 802.11ac. It uses a OS known as Raspbian and has no GPU capability as well as no specialized hardware accelerator[34,47].

NVIDIA Jetson Nano

NVIDIA Jetson Nano is a small GPU-based single-board computer that allows parallel operation of multiple NNs. It has a reduced size (100 mm $\times$ 80 mm $\times$ 29 mm) and is equipped with a Maxwell 128-core GPU, Quad-core ARM A57 64-bit 1.4 GHz CPU. Like in the case of Raspberry Pi, a MicroSD card is used to store the data. Finally, connections are established via Gigabit Ethernet and the OS employed is Linux4Tegra, based on Ubuntu 18.04[34,46].

Energy Consumption Measurement System

In this work, together with the latency and accuracy attributed to each model processing, we also address the issue of the power consumption of both, the Nvidia Jetson Nano and the Raspberry Pi 4.

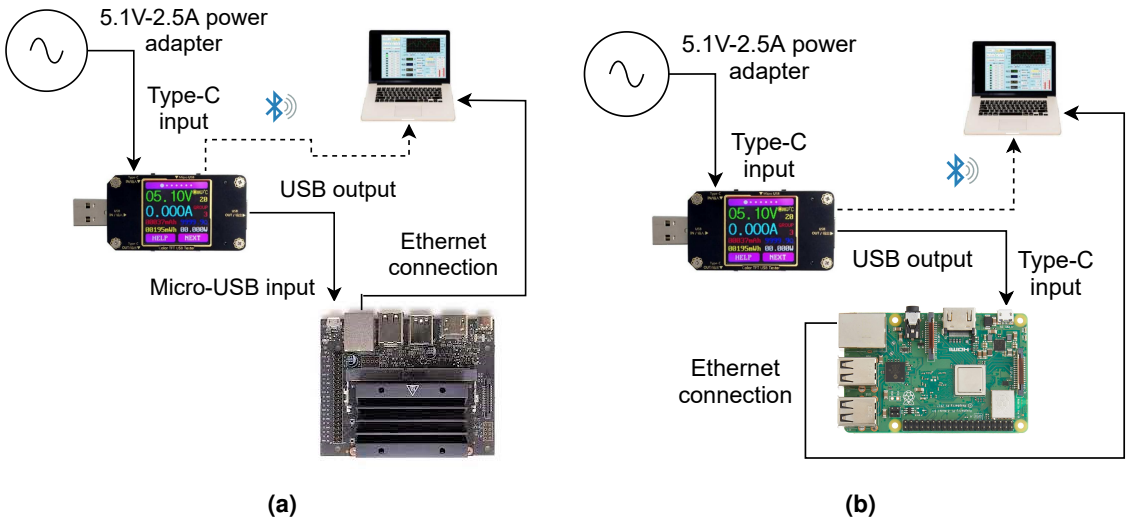


Figure 7. (a) Power measurement set-up for Nivida Jetson Nano. (b) Power measurement set-up for Raspberry Pi.

Power Measurement

It is possible to measure the power consumption of both the Nvidia Jetson Nano and the Raspberry Pi in different ways. Regarding Nvidia Jetson Nano, there are three onboard sensors located at the power input of the board, the GPU, and the CPU. Thus, the precision of the measurements is limited by these sensors. To read the recordings of these sensors, it is possible to do it automatically using the tegrastats tool, or manually by reading sys files, a pseudo-file system on Linux. By using both approaches, it is possible to obtain information of the measurements on power, voltage, and current[48]. In contrast, Raspberry Pi 4 has no system to easily get power consumption measurements. Some software-based methods have been developed, as well as some empirical estimations[49]. Nevertheless, it has been demonstrated that the first ones are just an approximation that can not be used if very precise results are required[49]. On the other hand, the second strategy to measure the power consumption of the Raspberry Pi is specific for this type of hardware and can not be used in Nvidia Jetson Nano.

To compare the power consumption of the equalizer in these two types of hardware, it is desirable to use the same method in both of them. In this paper, we developed a platform-agnostic method through the use of a digital USB multimeter. The proposed power consumption measurement system addresses the problem of these devices having no onboard shunt resistors, which would allow us to easily measure power with an external energy probe. A description of the measurement set-up is given in Fig. 7b.

In the case of Raspberry Pi, the power is supplied through a USB type C port via a 5.1 V–2.5 A power adapter. For Nvidia Jetson Nano, the power can be supplied through a Micro-USB connector using a 5.1 V–2.5 A power adapter or a Barrel jack 5 V–4 A (20 W) power supplier. It is possible to change from one configuration to the other by setting a jumper and moving from the 5 W Mode to the 10 W one. To use the same source of power as in Raspberry Pi, the Micro-USB configuration is used.

As energy is going to be supplied through a USB connection, it is possible to measure the power using a USB digital multimeter. The model used in this work is the A3-B/A3 manufactured by Innovateking-EU. It records voltage, current, impedance, and power consumption. The input voltage and current ranges are 4.5 V–24 V and 0 A–3 A, respectively. Moreover, it is possible to measure the energy in a range that goes from 0 mWh to 99999 mWh. The voltage and current measurement resolution are 0.01 V and 0.001 A, being the measurement accuracies $\pm 0.2\%$ and $\pm 0.8\%$, respectively.

Data Recording

To evaluate the inference time for each model, no peripherals are connected either to the Raspberry Pi or to Nvidia Jetson Nano, except the Ethernet port, which is used to establish communication over the Secure Shell protocol. Moreover, any initialization time (e.g., library loading, data generation, and model weight loading) is ignored because this is a one-time cost that occurs during the device's setup.

Regarding the power measurement, it is worth mentioning that the USB digital multimeter A3-B/A3 comes with software named UM24C PC Software V1.3, which allows sending the measured data to a computer in real-time, as it is shown in Fig. 7b. A similar strategy to the one employed when measuring the inference time is followed. Thus, no peripherals are connected either to the Raspberry Pi or to Nvidia Jetson Nano, except the ethernet port. This is used for communication over SSH, Fig. 7b. Moreover, 100 inferences were run in each device and an average value for the power consumption was calculated, not taking

into account the power consumed during the initialization phase.

References

1. Winzer, P. J., Neilson, D. T. & Chraplyvy, A. R. Fiber-optic transmission and networking: the previous 20 and the next 20 years. *Opt. Express* **26**, 24190–24239, DOI: 10.1364/OE.26.024190 (2018).
2. Cartledge, J. C., Guiomar, F. P., Kschischang, F. R., Liga, G. & Yankov, M. P. Digital signal processing for fiber nonlinearities. *Opt. Express* **25**, 1916–1936, DOI: 10.1364/OE.25.001916 (2017).
3. Rafique, D. Fiber nonlinearity compensation: Commercial applications and complexity analysis. *J. Light. Technol.* **34**, 544–553, DOI: 10.1109/JLT.2015.2461512 (2016).
4. Dar, R. & Winzer, P. J. Nonlinear interference mitigation: Methods and potential gain. *J. Light. Technol.* **35**, 903–930, DOI: 10.1109/JLT.2016.2646752 (2017).
5. Musumeci, F. *et al.* An overview on application of machine learning techniques in optical networks. *IEEE Commun. Surv. Tutorials* **21**, 1383 – 1408, DOI: 10.1109/COMST.2018.2880039 (2019).
6. Nevin, J. W. *et al.* Machine learning for optical fiber communication systems: An introduction and overview. *APL Photonics* DOI: 10.1063/5.0070838 (2021).
7. Jarajreh, M. A. *et al.* Artificial neural network nonlinear equalizer for coherent optical ofdm. *IEEE Photonics Technol. Lett.* **27**, 387–390, DOI: 10.1109/LPT.2014.2375960 (2015).
8. Häger, C. & Pfister, H. D. Nonlinear interference mitigation via deep neural networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 1–3 (IEEE, 2018).
9. Zhang, S. *et al.* Field and lab experimental demonstration of nonlinear impairment compensation using neural networks. *Nat. Commun.* **10**, 3033, DOI: 10.1038/s41467-019-10911-9 (2019).
10. Freire, P. J. *et al.* Performance versus complexity study of neural network equalizers in coherent optical systems. *J. Light. Technol.* **39**, 6085–6096, DOI: 10.1109/JLT.2021.3096286 (2021).
11. Deligiannidis, S., Bogris, A., Mesaritakis, C. & Kopsinis, Y. Compensation of fiber nonlinearities in digital coherent systems leveraging long short-term memory neural networks. *J. Light. Technol.* **38**, 5991–5999, DOI: 10.1109/JLT.2020.3007919 (2020).
12. Deligiannidis, S., Mesaritakis, C. & Bogris, A. Performance and complexity analysis of bi-directional recurrent neural network models versus volterra nonlinear equalizers in digital coherent systems. *J. Light. Technol.* **39**, 5791–5798, DOI: 10.1109/JLT.2021.3092415 (2021).
13. Freire, P. J. *et al.* Experimental study of deep neural network equalizers performance in optical links. In *2021 Optical Fiber Communications Conference and Exhibition (OFC)*, 1–3 (2021).
14. Sidelnikov, O., Redyuk, A. & Sygletos, S. Equalization performance and complexity analysis of dynamic deep neural networks in long haul transmission systems. *Opt. Express* **26**, 32765–32776, DOI: 10.1364/OE.26.032765 (2018).
15. Sidelnikov, O. S., Redyuk, A. A., Sygletos, S. & Fedoruk, M. P. Methods for compensation of nonlinear effects in multichannel data transfer systems based on dynamic neural networks. *Quantum Electron.* **49**, 1154, DOI: 10.1070/QEL17158 (2019).
16. Barry, J. R., Lee, E. A. & Messerschmitt, D. G. *Digital communication* (New York: Springer, 3rd edn, 2004).
17. Ming, H. *et al.* Ultralow complexity long short-term memory network for fiber nonlinearity mitigation in coherent optical communication systems. *arXiv preprint arXiv:2108.10212* (2021).
18. Kaneda, N. *et al.* Fpga implementation of deep neural network based equalizers for high-speed pon. In *Optical Fiber Communication Conference (OFC) 2020*, T4D.2, DOI: 10.1364/OFC.2020.T4D.2 (Optical Society of America, 2020).
19. Blalock, D., Ortiz, J. J. G., Frankle, J. & Guttag, J. What is the state of neural network pruning? (2020). 2003.03033.
20. Han, S., Mao, H. & Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (2016). 1510.00149.
21. Srinivas, S., Subramanya, A. & Babu, R. V. Training sparse neural networks. *2017 IEEE Conf. on Comput. Vis. Pattern Recognit. Work. (CVPRW)* 455–462 (2017).
22. Hawks, B. *et al.* Ps and qs: Quantization-aware pruning for efficient low latency neural network inference. *Front. Artif. Intell.* **4**, DOI: 10.3389/frai.2021.676564 (2021).

23. Sze, V., Chen, Y.-H., Yang, T.-J. & Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* **105**, 2295–2329, DOI: 10.1109/JPROC.2017.2761740 (2017).

24. Liang, T., Glossner, J., Wang, L., Shi, S. & Zhang, X. Pruning and quantization for deep neural network acceleration: A survey (2021). 2101.09671.

25. Fujisawa, S. *et al.* Weight pruning techniques towards photonic implementation of nonlinear impairment compensation using neural networks. *J. Light. Technol.* DOI: 10.1109/JLT.2021.3117609 (2021).

26. Li, M., Zhang, W., Chen, Q. & He, Z. High-throughput hardware deployment of pruned neural network based nonlinear equalization for 100-gbps short-reach optical interconnect. *Opt. Lett.* **46**, 4980–4983 (2021).

27. Oliari, V. *et al.* Revisiting efficient multi-step nonlinearity compensation with machine learning: An experimental demonstration. *J. Light. Technol.* **38**, 3114–3124 (2020).

28. Koike-Akino, T., Wang, Y., Kojima, K., Parsons, K. & Yoshida, T. Zero-multiplier sparse dnn equalization for fiber-optic qam systems with probabilistic amplitude shaping. In *2021 European Conference on Optical Communications (ECOC)*, 1–4 (IEEE, 2021).

29. Freire, P. J. *et al.* Transfer learning for neural networks-based equalizers in coherent optical systems. *J. Light. Technol.* **39**, 6733–6745, DOI: 10.1109/JLT.2021.3108006 (2021).

30. Pelikan, M., Goldberg, D. E., Cantú-Paz, E. *et al.* Boa: The bayesian optimization algorithm. In *Proceedings of the genetic and evolutionary computation conference GECCO-99*, vol. 1, 525–532 (Citeseer, 1999).

31. Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org.

32. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N. & Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks (2021). 2102.00554.

33. Neill, J. O. An overview of neural network compression (2020). 2006.03669.

34. Hadidi, R. *et al.* Characterizing the deployment of deep neural networks on commercial edge devices. In *2019 IEEE International Symposium on Workload Characterization (IISWC)*, 35–48 (IEEE, 2019).

35. Närhi, M. *et al.* Machine learning analysis of extreme events in optical fibre modulation instability. *Nat. Commun.* **9**, 4923, DOI: 10.1038/s41467-018-07355-y (2018).

36. Dong, X. & Zhou, L. Understanding over-parameterized deep networks by geometrization (2019). 1902.03793.

37. Bondarenko, A., Borisov, A. & Alekseeva, L. Neurons vs weights pruning in artificial neural networks. In *ENVIRONMENT. TECHNOLOGIES. RESOURCES. Proceedings of the International Scientific and Practical Conference*, vol. 3, 22–28 (2015).

38. Hu, H., Peng, R., Tai, Y. & Tang, C. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR* **abs/1607.03250** (2016). 1607.03250.

39. Choukroun, Y., Kravchik, E., Yang, F. & Kisilev, P. Low-bit quantization of neural networks for efficient inference (2019). 1902.06822.

40. Yang, J. *et al.* Quantization networks (2019). 1911.09464.

41. Wu, H., Judd, P., Zhang, X., Isaev, M. & Micikevicius, P. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602* (2020).

42. Gholami, A. *et al.* A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630* (2021).

43. Hubara, I., Nahshan, Y., Hanani, Y., Banner, R. & Soudry, D. Accurate post training quantization with small calibration sets. In *International Conference on Machine Learning*, 4466–4475 (PMLR, 2021).

44. de Lima, T. F. *et al.* Machine learning with neuromorphic photonics. *J. Light. Technol.* **37**, 1515–1534 (2019).

45. Baskin, C. *et al.* Uniq: Uniform noise injection for non-uniform quantization of neural networks. *ACM Transactions on Comput. Syst.* **37**, DOI: 10.1145/3444943 (2021).

46. Valladares, S., Toscano, M., Tufiño, R., Morillo, P. & Vallejo-Huanga, D. Performance evaluation of the nvidia jetson nano through a real-time machine learning application. In *International Conference on Intelligent Human Systems Integration*, 343–349 (Springer, 2021).

47. Tang, R., Wang, W., Tu, Z. & Lin, J. An experimental analysis of the power consumption of convolutional neural networks for keyword spotting. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5479–5483 (IEEE, 2018).

48. Holly, S., Wendt, A. & Lechner, M. Profiling energy consumption of deep neural networks on nvidia jetson nano. In *2020 11th International Green and Sustainable Computing Workshops (IGSC)*, 1–6 (IEEE, 2020).

49. Kaup, F., Gottschling, P. & Hausheer, D. Powerpi: Measuring and modeling the power consumption of the raspberry pi. In *39th Annual IEEE Conference on Local Computer Networks*, 236–243 (IEEE, 2014).

Acknowledgements

SKT and MKK are partially supported by the EPSRC programme grant TRANSNET, EP/R035342/1. PJF and DAR acknowledge the support from the EU Horizon 2020 Marie Skodowska-Curie Action projects No. 813144 (REAL-NET) and 860360 (POST-DIGITAL), respectively. JEP and SKT acknowledge the support of the Leverhulme Trust project RPG-2018-063.

Author contributions statement

DAR, PJF, and JEP conceived the study. DAR and PJF proposed the neural network model. DAR performed the numerical simulations, designed the experimental set-up and obtained the experimental results. PJF generated the data and performed the architecture optimization. DAR and PJF designed the figures and tables. DAR, PJF, and JEP wrote the manuscript, with the assistance of MKK and SKT. All authors reviewed the manuscript. The work of DAR was supervised by MKK and SKT. The work of PJF was supervised by JEP, AN and SKT.

Data availability

Data underlying the results presented in this paper are not publicly available at this time, but can be obtained from the authors upon request.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to D.A.R or S.K.T