Q-EEGNet: an Energy-Efficient 8-bit Quantized Parallel EEGNet Implementation for Edge Motor-Imagery Brain–Machine Interfaces

Author(s):
Schneider, Tibor; Wang, Xiaying; Hersche, Michael; Cavigelli, Lukas Arno Jakob; Benini, Luca

Publication date:
2020

Permanent link:
https://doi.org/10.3929/ethz-b-000457247

Originally published in:
https://doi.org/10.1109/SMARTCOMP50058.2020.00065

Q-EEGNet: an Energy-Efficient 8-bit Quantized Parallel EEGNet Implementation for Edge Motor-Imagery Brain–Machine Interfaces

Tibor Schneider*, Xiaying Wang*, Michael Hersche*, Lukas Cavigelli†, Luca Benini*‡

*ETH Zürich, Dept. EE & IT, Switzerland ‡University of Bologna, DEI, Italy
†Huawei Technologies, Zurich Research Center, Switzerland

***Abstract*—Motor-Imagery Brain–Machine Interfaces (MI-BMIs) promise direct and accessible communication between human brains and machines by analyzing brain activities recorded with Electroencephalography (EEG). Latency, reliability, and privacy constraints make it unsuitable to offload the computation to the cloud. Practical use cases demand a wearable, battery-operated device with low average power consumption for long-term use. Recently, sophisticated algorithms, in particular deep learning models, have emerged for classifying EEG signals. While reaching outstanding accuracy, these models often exceed the limitations of edge devices due to their memory and computational requirements. In this paper, we demonstrate algorithmic and implementation optimizations for EEGNET, a compact Convolutional Neural Network (CNN) suitable for many BMI paradigms. We quantize weights and activations to 8-bit fixed-point with a negligible accuracy loss of 0.4% on 4-class MI, and present an energy-efficient hardware-aware implementation on the Mr. Wolf parallel ultra-low power (PULP) System-on-Chip (SoC) by utilizing its custom RISC-V ISA extensions and 8-core compute cluster. With our proposed optimization steps, we can obtain an overall speedup of 64× and a reduction of up to 85% in memory footprint with respect to a single-core layer-wise baseline implementation. Our implementation takes only 5.82 ms and consumes 0.627 mJ per inference. With 21.0 GMAC/s/W, it is 256× more energy-efficient than an EEGNET implementation on an ARM Cortex-M7 (0.082 GMAC/s/W).**

***Index Terms*—brain–machine interface, edge computing, parallel computing, machine learning, deep learning, motor imagery.**

I. INTRODUCTION

A Brain–Machine Interface (BMI) is a system that enables direct communication between humans and devices based on signals recorded from brain activities. One promising BMI approach is based on Motor-Imagery (MI), which describes the cognitive process of thinking about motions without actually performing them. Patients with severe physical disabilities could rely on MI-BMIs to regain independence [1], [2].

MI-BMIs are often based on Electroencephalography (EEG), an accessible and widely used method for measuring brain activities. However, EEG data show high variability across different subjects as well as different recordings of the same subject, making accurate classification a challenging task. A common approach is to rely on domain-specific knowledge, extracting human-interpretable features such as Filter-Bank Commmon Spatial Patterns (FBCSP) [3] or Riemannian geometry features [4]. Promising alternatives are Convolutional Neural Networks (CNNs), which are gaining increasing attention in the MI-BMI field thanks to high state-of-the-art (SoA) classification accuracy [5], [6]. A popular competitor is EEGNET [7], a CNN-based approach generally applicable to many different BMI paradigms, achieving comparable accuracy to architectures tailored to the specific use case while still being compact (<3000 parameters), compared to other CNNs for MI-BMIs [8].

Most existing BMI systems rely on offline remote computing for classification; however, having those networks mapped to low-cost, low-power embedded platforms, e.g., Microcontroller units (MCUs), is very beneficial. MCU-based platforms and devices are comfortable, light, and less power-hungry. Classifying signals on the edge eliminates the latency due to communication, and the energy required for the data transfer. Besides, processing brain signals on the recording device itself allows users to maintain their privacy. Several energy-efficient platforms have been proposed in both industry and academia for enabling continuous long-term classification on battery-operated edge devices. The most popular energy-efficient MCUs are from the ARM Cortex-M family, with Cortex-M7 being the highest-performing member. Recently, researchers have developed the parallel ultra-low power (PULP) platform based on the RISC-V Instruction Set Architecture (ISA) [9], [10] , which is built around the concept of using simple cores for energy efficiency, while recovering and scaling up performance through parallelism. PULP MCUs have proven to outperform the Cortex-M family by at least one order of magnitude in energy efficiency [11], [12]. In particular, Mr. Wolf, with its 8-core compute cluster and custom ISA extensions, can reach up to 274 GOp/s/W [13].

Nevertheless, both Cortex-M and RISC-V based MCU platforms are tightly constrained both in memory and compute resources, which forced other embedded solutions to tailor and scale down EEGNET for the target system resulting in lower classification accuracy [14]. To address this challenge, we present Q-EEGNET, an adapted and quantized EEGNET [7] with algorithmic and implementation optimizations to execute BMI inference on resource-limited edge devices. The proposed methods overcome the necessity of network reduction for embedded implementations and are generally applicable to other CNNs in MI-BMIs. The main contributions of this paper are as follows:

- We quantize weights and activations of EEGNET from 32-bit float to 8-bit fixed-point representation using quantization-aware training and Random Partition Relaxation (RPR) [15], resulting in a negligible loss of 0.4% accuracy on the 4-class BCI Competition IV-2a dataset [16]

Corresponding emails: {xiaywang, herschmi}@iis.ee.ethz.ch

(Section III). This allows the use of vectorized integer operations and the compression of the weights and feature maps by 4×.
- We present an optimized hardware-aware implementation of the quantized model on Mr. Wolf (Section IV). The concurrent execution and the use of the RISC-V ISA extensions yield a speedup of 36.1× compared to the baseline single-core implementation.
- We overcome the traditional layer-by-layer computation paradigm and propose an interleaved implementation that achieves up to 85% reduction in memory footprint and an overall speedup of 64×.
- Experimental measurements, in Section V, show that the execution of Q-EEGNET on Mr. Wolf takes 5.82 ms per inference consuming only 0.627 mJ, yielding an energy-efficiency of 20.957 GMAC/s/W. Compared to another implementation of a reduced EEGNET on an ARM Cortex-M7 [14] with 0.082 GMAC/s/W, Q-EEGNET on Mr. Wolf is 256× more energy-efficient.

Finally, we release open-source code developed in this work[1].

II. Background

A. Dataset description

In this work, we use the BCI Competition IV-2a dataset [16], which contains recordings from 9 different subjects and distinguishes between four classes of imagined movements: left and right hand, both feet, and the tongue. 22 different EEG channels were recorded, sampled at 250 Hz. The data is pre-processed with a bandpass filter between 0.5 and 100 Hz. Each subject completed two recording sessions on two different days. Recordings from the first day are used only for training, and samples from the second session are used exclusively for testing. Per subject and per session, 288 trials were recorded, of which almost 10% were excluded due to artifacts originating mostly from eye movements. The dataset, however, remains balanced. Per trial, 6 s of EEG data is recorded: 2 s before the MI-cue, 1 s of showing the cue, and 3 s when the subject was executing MI.

B. EEGNet

EEGNET [7] is a Convolutional Neural Network (CNN) designed to apply to many different BMI paradigms such as P300 event-related potential (P300), feedback error-related negativity (ERN), movement-related cortical potential (MRCP), and sensory-motor rhythm (SMR) encountered in MI. Another design goal of EEGNET is to contain as few model parameters as possible, which is essential in many applications due to the limited amount of labeled training data. It consists of three convolutional layers in the Temporal, Spatial, and Separable Convolution blocks, depicted in Fig. 1. Each convolution is followed by a Batch Normalization (BN) layer and a linear or Exponential Linear Unit (ELU) activation. All convolutional kernels are 1-dimensional (1D). The network contains two average pooling layers to reduce the size of the feature maps. The final classification is a linear fully-connected (FC) layer. Thanks to the use of depth-wise convolutions and pooling layers, EEGNET requires only 2548 parameters and 13.14 million[2] Multiply Accumulate (MAC) operations per inference. Nevertheless, it achieves an accuracy of 71.0% on 4-class MI, which is 3% more accurate than the winner of the BCI competition IV-2a [3].

C. Mr. Wolf

Mr. Wolf [13] is a System-on-Chip (SoC) for embedded, low-power applications. Mr. Wolf is split into two computation domains: the SoC domain and the compute cluster. The SoC domain is responsible for handling inputs and outputs, as well as computationally simple tasks. It is based around the fabric controller with a RISC-V processor called IBEX [9]. The SoC domain contains 448 kB of shared L2 memory. The compute cluster consists of eight in-order four-stage RISC-V RV32IMFCXPULPV2 processors called RI5CY [10] (now maintained by the OpenHW Group as CV32E40P), which support the RVC32IMF instruction set and the XPULPV2 extension, adding support for Single Instruction, Multiple Data (SIMD), load and store post-increment, and hardware loops. The cluster is available on demand; individual cores can be disabled to save energy. All cores have access to 64 kB of shared L1 memory via the Tightly Coupled Data Memory (TCDM) interconnect. A Direct Memory Access (DMA) controller is responsible for moving data between L1 and L2 memory.

III. Related work

As a result of the emerging Internet of Things (IoT), which brings intelligence close to the sensor, the current literature is rich in implementing inference of neural networks on low-power edge devices and is also gaining increasing attention in MI-BCI. CUBE.AI converts trained models from Keras and generates an optimized code for several embedded platforms of the STM32 series. In contrast, TENSORFLOW LITE supports various platforms, including RISC-V [17]. However, the resulting implementation for RISC-V does not support parallel execution. FANN-ON-MCU [12] is a different framework for exporting optimized neural networks to ARM processors, and to PULP-based systems. However, this framework does not offer convolutional layers required for EEGNET. PULP-NN [11] is a library containing highly optimized implementations for typical (convolutional) neural networks targeting the PULP-platform.

In [14], EEGNET was applied to the Physionet Motor Movement/Imagery Dataset, achieving SoA accuracy. The model was quantized and ported to an ARM Cortex-M7 using CUBE.AI, i.e., the X-CUBE-AI expansion package of STM32CubeMX. However, the current package expansion can quantize only the FC layer to 8 bits, which is almost insignificant in terms of computation compared to the rest of EEGNET. The input feature map had to be scaled down significantly from (64 channels × 480 time-samples) to (38 channels × 80 time-samples) by sub-sampling, EEG channel reduction, and narrowing the time window, such that the feature maps fitted on the available SRAM.

[1] https://github.com/pulp-platform/q-eegnet

[2] Each convolutional layer contributes with $h_{out} \cdot w_{out} \cdot n_{out} \cdot h_k \cdot w_k \cdot n_{in}/g$ MACs, with g being the number of groups (commonly g=1, for depth-wise sep. conv. g=n_{in})

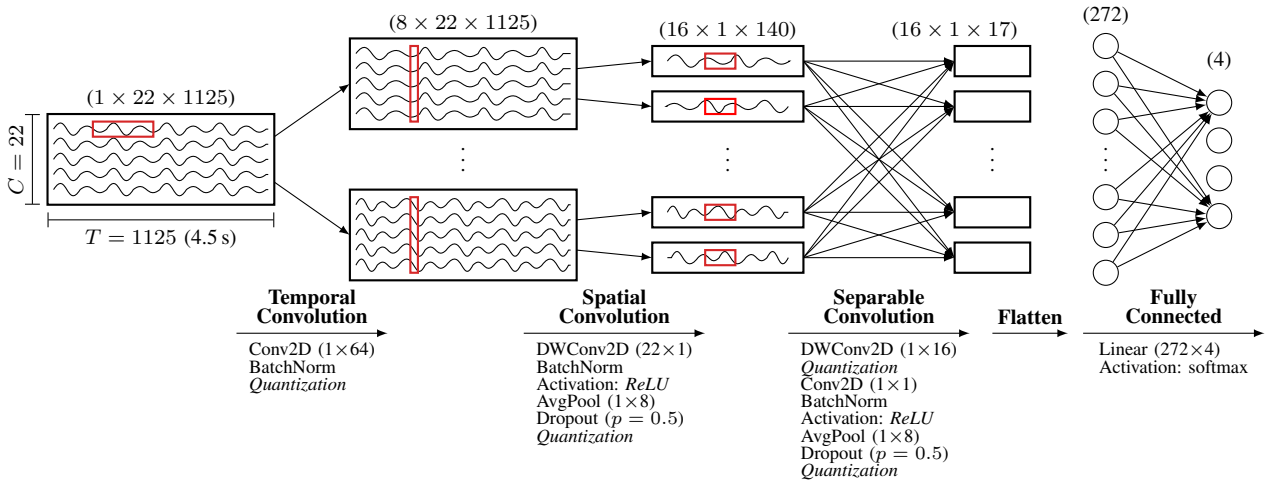

Fig. 1. Q-EEGNET architecture applied to BCI Competition IV-2a dataset [7]. The input signal is quantized to 8-bit fixed-point.

IV. Model Design and Quantization

This section explains how EEGNET is modified, quantized, and trained, resulting in Q-EEGNET targeting a low-power implementation. Fig. 1 illustrates the layers of Q-EEGNET, which processes 4.5 s of EEG data, starting 0.5 s before the onset of the MI-cue according to the timing scheme of the BCI Competition IV-2a dataset. We have modified the original EEGNET as follows:

- The computationally expensive ELU activations are replaced with Rectified Linear Unit (ReLU).
- The weight regularization is removed from the training procedure since it has no effect on the accuracy and interferes with the quantization procedure.

In this work, all weights and activations, including the input signals, are quantized independently to 8-bit fixed-point representations. This reduces the memory footprint and enables maximal use of the underlying microprocessor architecture with its 4-way SIMD instructions.

As shown in Fig. 1, we do not introduce quantization between every single layer of the network. Instead, we requantize only before the convolutional and the FC layers. The reason is that all other layers (i.e., BN, ReLU, and average pooling layers) are defined locally. They can easily be computed one after the other, without writing back to memory. Requantizing those values to less than 32-bit fixed-point values would increase the quantization error and introduce a higher overhead than the subsequent speedup.

A quantization layer first rescales the activations according to their expected range, and then reduces the precision from 32-bit to 8-bit fixed-point. Usually, it is beneficial to choose the scaling factor to be a power of two, such that it can be implemented with an efficient bit-shift instead of an expensive integer division. However, the scaling factors and offsets of the BN layers are learned during training, and cannot be approximated as powers of two. Thus, we require a full integer division. Alternatively, the BN layer could be merged into the preceding convolution before the quantization-aware retraining and the shift be constrained to a power-of-two. However, this removes a degree of freedom and might thus adversely affect the final accuracy.

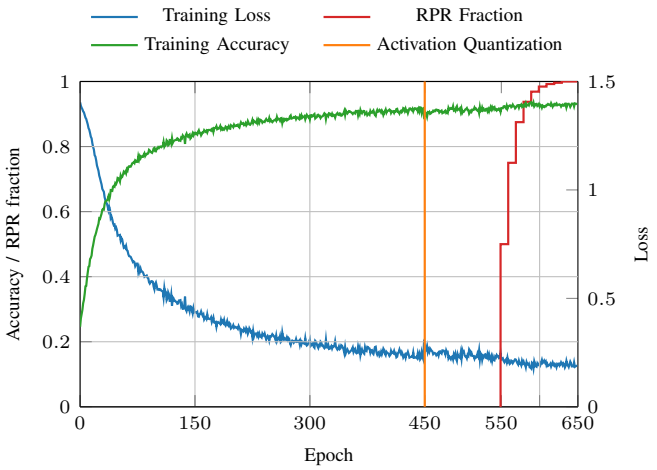

Fig. 2. Training loss and accuracy of Q-EEGNET on Subject 5.

The network is first trained in full precision, for 450 epochs, to get a pre-trained model. In the 450th epoch, the value range of the activations is monitored. In subsequent epochs, they are quantized using the straight-through estimator (STE), i.e., the values are quantized in the forward pass while the full precision values are used for backpropagation [18]. The next 100 epochs are necessary for the network to adapt to the quantized activations. During the last 100 epochs of the training process, the weights of the network are quantized incrementally using Random Partition Relaxation (RPR) [15]. Fig. 2 illustrates the training process and shows the training loss and accuracy for Subject 5.

V. Implementation and Optimizations

This section elaborates on implementing Q-EEGNET on Mr. Wolf and highlights our novel optimizations, which enable high energy-efficiency.

EEGNET contains multiple 2D convolutions. However, the kernel size in all layers spans only one dimension, which means that they can be computed using exclusively 1D convolutions. We start with a baseline implementation on a single RI5CY core, not utilizing the SIMD instructions. For the baseline, the weights and feature maps are transferred layer-

by-layer from L2 to L1 memory using the DMA unit; the computation is done after every completed transfer, and the result is subsequently moved back to L2 memory. On top of this, we incrementally add the following optimization steps:

A. SIMD and loop unrolling

Since all activations and weights are quantized to 8 bits, four of them are packed into a single 32-bit word to have much more effective loads and exploit SIMD instructions [10]. Moreover, loop unrolling is applied to reduce the pipeline stalls after load operations. For 1D convolutions, we use the optimized implementation from the PULP-DSP library [19], which computes four elements of the output vector at a time using 8-bit operands and 32-bit accumulators.

Based on Mr. Wolf's architecture and the SIMD instructions available on the RI5CY cores, we choose the time dimension to be the innermost dimension to exploit the optimized 1D convolutions. Additionally, we align every feature map and weight matrix to 4 Bytes, eliminating all misaligned memory accesses.

B. Transpose feature maps

The kernels of the Spatial Convolution span only the space dimension (along the different EEG channels), whereas the feature maps are packed along the time dimension. This prohibits the use of SIMD for this layer. However, transposing the feature maps (switching space and time dimension) allows SIMD instructions for the convolution, analogous to the Temporal Convolution. Additionally, since the kernel size is equal to the number of EEG channels, the Spatial Convolution can be implemented as a series of dot products.

Similarly, the pointwise convolution (the second part of the Separable Convolution) only consists of (1×1) kernels. By switching the time dimension with the channel dimension, we can compute the convolution as a series of SIMD dot products.

C. Concurrent execution

The compute cluster of the target platform contains eight cores; using all of them for inference has a significant impact on the performance and the energy efficiency. For the Temporal Convolution, all the $22 \times 8 = 176$ different 1D convolutions are distributed among the eight cores. The Spatial Convolution is computed in parallel by assigning each core of the cluster one of the eight feature maps. The analogous is done for both convolutional layers in the Separable Convolution. Finally, we implement the FC layer on a single core, since it contains less than 0.01% of all MAC operations in the entire network; therefore, any improvement at this stage has no significant effect on the overall performance.

D. Cross-correlation instead of convolution

Cross-correlation is a close sibling to convolution. The only difference between those operations is that, for the convolution, the weight vector must be flipped. Computing a convolution requires reversing the weight vector, resulting in an additional instruction in the innermost loop. By storing the weight vector in reverse order on the target, and using cross-correlations instead of the convolutions, this shuffle instruction is no longer necessary, reducing the number of instructions.

E. Interleaved layers

The Temporal Convolution creates eight different output channels, increasing the size of the feature maps by $8\times$. The Spatial Convolution then reduces the number of EEG channels from 22 down to 1, and the subsequent pooling layer reduces the time resolution by a factor of $8\times$. Therefore, not storing the output of the Temporal Convolution would reduce the total memory requirements of Q-EEGNET by 85%.

This reduction in memory can be achieved by exploiting the nature of convolution layers; a small input region fully determines a single output feature. In the case of the Spatial Convolution, an output feature can be computed from a single column (spatial dimension) at the input feature map. Thus, we interleave the computation of the Temporal and Spatial Convolutions, computing only a single column with the Temporal Convolution, followed by computing a single output element of the Spatial Convolution. For the intermediate result, we reuse the same memory location.

F. Merging batch normalization

A trained BN layer can be computed in several different ways. For a fixed-point implementation, the most precise results are achieved by adding a bias b and then dividing by a factor f. The BN layer in the Temporal Convolution block of Fig. 1 requires almost 200 000 integer divisions. To reduce the number of divisions, which can be very costly on low-power embedded platforms, we exploit the linearity of the convolution operation. More specifically, the division of the first BN is moved after the depth-wise convolution and combined with the BN in the Spatial Convolution block, reducing the number of divisions by more than a factor $10\times$. This can be expressed as:

$$y = \frac{(x \star w_T + b_1)/f_1 \star w_S + b_2}{f_2} = \frac{(x * w_T + b_1) \star w_S + f_1 b_2}{f_1 f_2},$$

where $\star$ is the convolution operation, x the input feature map, w_T the network weights in the Temporal Convolution block, and w_S the weights in the Spatial Convolution block. b_1 and f_1 represent the bias and the normalization factor of the first BN layer, similarly, b_2 and f_2 for the second BN layer.

Note that after the Temporal Convolution, the resulting features are in 32-bit representation using the 1D convolution of PULP-DSP library. Since the complexity of the depth-wise convolution in the Spatial Convolution block is orders of magnitude lower than the Temporal Convolution, we do not requantize the activations and execute the depth-wise convolution in 32-bit with negligible impact on the overall performance. This also reduces the error introduced by the requantization.

G. Layer reordering

In both the Spatial and Separable Convolution blocks, the convolution is followed by a BN, a ReLU, and a pooling layer. However, it is beneficial first to execute the pooling layer to reduce the feature maps, and then apply the other layers, which decreases the overall number of operations. In contrast to the

```
lp.setup 0,a2,end
    p.lw t4,4(a7!)          // t4 = {w0,w1,w2,w3}
    p.lw t0,4(a8!)          // t0 = {x0,x1,x2,x3}
    lw t3,a4                // t3 = {x4,x5,x6,x7}
    mv t1,t0
    mv t2,t0
    pv.shuffle2.b t1,t3,a9  // t1 = {x1,x2,x3,x4}
    pv.shuffle2.b t2,t3,a10 // t2 = {x2,x3,x4,x5}
    pv.shuffle2.b t3,t0,a11 // t3 = {x3,x4,x5,x6}
    pv.sdotsp.b t0,t4,a3    // a3 += w0*x0+w1*x1+w2*x2+w3*x3
    pv.sdotsp.b t1,t4,a4    // a4 += w0*x1+w1*x2+w2*x3+w3*x4
    pv.sdotsp.b t2,t4,a5    // a5 += w0*x2+w1*x3+w2*x4+w3*x5
end:pv.sdotsp.b t3,t4,a6    // a6 += w0*x3+w1*x4+w2*x5+w3*x6
```

Listing 1. Cross-correlation with shuffle. The pointer to the weights is stored in register a7, and the pointer to the data in register a8. Registers a9, a10 and a11 contain the appropriate shuffle mask.

```
lp.setup 0,a2,end
    p.lw t4,4(a7!)          // t4 = {w0,w1,w2,w3}
    p.lw t0,4(a8!)          // t0 = {x0,x1,x2,x3}
    p.lw t1,4(a9!)          // t1 = {x1,x2,x3,x4}
    p.lw t2,4(a10!)         // t2 = {x2,x3,x4,x5}
    p.lw t3,4(a11!)         // t3 = {x3,x4,x5,x6}
    pv.sdotsp.b t0,t4,a3    // a3 += w0*x0+w1*x1+w2*x2+w3*x3
    pv.sdotsp.b t1,t4,a4    // a4 += w0*x1+w1*x2+w2*x3+w3*x4
    pv.sdotsp.b t2,t4,a5    // a5 += w0*x2+w1*x3+w2*x4+w3*x5
end:pv.sdotsp.b t3,t4,a6    // a6 += w0*x3+w1*x4+w2*x5+w3*x6
```

Listing 2. Cross-correlation with data replication. The pointer to the weights is stored in register a7, while the pointer to the 4 copies of the data (shifted by 1 Byte) are stored in registers a8, a9, a10, and a11, respectively.

non-linear ReLU layer, the BN can be computed after the pooling layer, shown as follows:

$$y = \frac{1}{N}\sum_{i=0}^{N-1} \max\left(\frac{x_i + b}{f}, 0\right) = \frac{Nb + \sum \max(x_i, -b)}{Nf},$$

where b and f are respectively the bias and the normalization factor of BN, the $\max(\cdot, 0)$ is the original ReLU activation, and the average summation represents the pooling layer. The new ReLU activation $\max(\cdot, -b)$ is shifted by b, and the BN is combined with the division from the average pooling layer. This reduces the number of divisions by the pooling factor N and the number of additions by $N-1$.

H. Replicate feature maps

In order to use SIMD for computing convolutions, we need to re-shuffle the data whenever the kernel is shifted by 1 Byte, as can be seen in Listing 1, because the packed data access is no longer aligned. However, the additional shuffle instructions can be avoided by replicating feature maps. We use the DMA to copy the feature maps four times to local L1 memory, each of which is shifted by 1 Byte. These DMA transfers add an insignificant overhead, compared to the shuffle instructions they replace. The resulting implementation no longer requires shuffle instructions, as shown in Listing 2, at the cost of more memory usage and accesses. The L1 memory available inside the cluster is not large enough to fit the entire input data, when replicated four times. Hence, we split the data into five similarly sized parts along the time dimension, which allowed us to fit the data into L1 memory and at the same time minimize the number of DMA transfers. The DMA independently transfers the data into the cluster memory while the cores are computing on the previously loaded data, reducing the idle time of the cores.

VI. Experimental Results

To obtain results comparable to literature, we strictly follow the rules of BCI Competition IV-2a for splitting the dataset, as explained in Section II-A. Table I compares the classification accuracy of the original EEGNet, the adapted EEGNet using ReLU activations, and the quantized Q-EEGNet. The training and testing procedures are implemented in PyTorch and are repeated 50 times for each subject to determine the variance in accuracy among different runs. Table I reports the average accuracy and the standard deviation over the runs. The network modifications (i.e., using ReLU instead of ELU) had an negligible impact of -0.1% on the classification accuracy compared to the original EEGNet. Moreover, the quantization to 8-bit fixed-point yields a negligible accuracy loss of 0.3%.

TABLE I
Classification Accuracy (% Avg. ± Std. Dev. Over 50 Runs) on 4-Class BCI Competition IV-2a in Full Precision and Quantized

	EEGNet		Q-EEGNet
Activation	ELU	ReLU	ReLU
Quantization	none	none	8 bits
Subject 1	81.0 ± 2.4	81.1 ± 2.2	81.0 ± 2.3
Subject 2	57.6 ± 4.5	52.2 ± 4.4	53.1 ± 4.2
Subject 3	87.9 ± 2.7	91.3 ± 2.1	91.2 ± 2.4
Subject 4	61.6 ± 3.4	59.1 ± 4.0	58.1 ± 4.0
Subject 5	70.6 ± 2.3	68.6 ± 3.2	68.4 ± 2.6
Subject 6	53.4 ± 3.2	52.0 ± 3.8	50.1 ± 4.3
Subject 7	75.7 ± 6.9	76.8 ± 5.2	75.2 ± 5.0
Subject 8	77.4 ± 4.2	80.0 ± 2.1	81.2 ± 1.9
Subject 9	76.7 ± 4.3	79.3 ± 3.1	79.7 ± 2.9
Mean	**71.3 ± 1.3**	**71.2 ± 1.4**	**70.9 ± 1.3**
Std. dev. sub*	11.5	14.0	14.3

*Standard deviation across average accuracies per subject.

TABLE II
Optimizations for Q-EEGNet on Mr. Wolf at 50 MHz. Letters A–H refer to sections V-A–V-H.

	baseline	A+B	C	D	E+F+G	H
Temp. Conv. ms]	1653.82	331.37	42.67	40.50	—	—
Spat. Conv. [ms]	58.00	42.31	6.74	6.74	—	—
Temp.+Spat. [ms]	—	—	—	—	31.08	26.16
Sep. Conv. [ms]	11.90	6.88	0.94	0.94	0.81	0.81
FC [ms]	0.07	0.07	0.07	0.07	0.07	0.07
Complete [ms]	1732.01	380.27	50.33	48.04	31.98	27.06
Speedup	—	4.55×	7.56×	1.05×	1.50×	1.18×
Tot. speedup	—	4.55×	34.4×	36.1×	54.2×	64.0×
Memory [kB]	230.29	248.87	248.87	248.87	35.41	68.15
MACs/cycle	0.15	0.69	5.18	5.42	8.15	9.63
insn/cycle	0.79	0.69	0.66	0.65	0.89	0.83

Table II shows the performance improvements for the optimizations on Q-EEGNet presented in Section V, and compares the computation time of the different parts on Mr. Wolf, executed at 50 MHz. One can notice that the execution time of the complete inference, executing all the layers at once, does not correspond precisely to the sum of each layer. This is due to the variability introduced by the measurement framework; however, the difference is negligible. From the table, we can see that with PULP-DSP library, the execution is accelerated by 4.55× using SIMD, loop unrolling, and transposing feature maps (A + B). Furthermore, the 7.56× speedup demonstrates that Q-EEGNet can be parallelized very well over eight cores using concurrent execution (C). With the substitution of cross-correlation instead of convolution (D), we gain another 5% speedup. When combining our novel optimizations (E–H), the speedup is additionally improved by 78%, resulting in an overall speedup of 64× with respect to the baseline

TABLE III
COMPARISON BETWEEN Q-EEGNET ON MR. WOLF AND EEGNET ON ARM CORTEX M7.

Platform	Mr. Wolf (ours)		Cortex M7 [14]
Input size	22×1125		38×80
MACs	12 984 432		1 509 220
Memory	68.15 kB		146.32 kB
	@50 MHz	@350 MHz	@216 MHz
Power [mW]	11.75	107.87	413.06
Time/inference [ms]	28.67	5.82	43.81
Energy/inference [mJ]	0.337	0.627	18.1
Throughput [MMAC/s]	458	2258	34
En. eff. [GMAC/s/W]	38.990	20.957	0.082

implementation, highlighting the effectiveness of our proposed optimizations. Looking more closely, the interleaved computation of the Temporal and Spatial Convolution (E) reduces the memory footprint of Q-EEGNET by almost 85% from 230.29 kB to 35.41 kB, making it applicable to embedded devices with very limited memory availability. Moreover, merging (F) and reordering (G) the BN layers reduces the total number of divisions by 98%. This fact also explains the higher number of instructions per cycle. Finally, the replication of the feature maps (H) eliminates the re-shuffling instructions and gives the highest performance of 9.63 MACs/cycle. However, it introduces more memory usage, but still 70% less than the baseline implementation. It is left then to the discretion of the user whether to include this last optimization step, depending on the available resources.

For comparison, we implement the most compute-intensive block of Q-EEGNET—the Temporal Convolution block contributing over 95% to the overall MAC operations—on Mr. Wolf using the PULP-NN library [11]. The measured result shows that it takes 76.13 ms to complete, achieving only 3.28 MACs per cycle. This is $3\times$ slower than our implementation, which at the same time, also computes all remaining layers and includes all DMA transfers. The reason can be mostly attributed to the focused optimization of PULP-NN on 2D convolutions instead of 1D convolutions, which are often encountered in CNNs for image classification.

Finally, we perform power measurements to assess the energy consumption of our implementation. Table III shows the measurements on Mr. Wolf, including the startup time of the compute cluster, in two different configurations: 50 MHz @ 0.8 V and 350 MHz @ 1.2 V. The former consumes the least amount of energy per inference (0.337 mJ) at highest energy efficiency (38.99 GMAC/s/W), while the latter provides the highest performance (2258 MMAC/s) by executing one inference in only 5.82 ms at energy efficiency of 20.97 GMAC/s/W. Comparing to [14], which has also implemented EEGNET at maximum clock frequency of 216 MHz on an ARM Cortex-M7 (STM32F756ZG) using CUBE.AI, our implementation is approximately $256\times$ more energy-efficient.

VII. CONCLUSION

This paper presents Q-EEGNET, a modified and 8-bit quantized EEGNET, which enables energy-efficient inference on resource-limited low-power edge devices at negligible accuracy loss. With the proposed optimizations, which can be adopted by other similar CNN architectures, we achieve a runtime speedup of up to $64\times$ relative to the baseline implementation on Mr. Wolf, yielding only 5.82 ms and 0.627 mJ per inference. Due to its specialization, our implementation surpasses the energy-efficiency of general CNN libraries, like PULP-NN and CUBE.AI. This work shows that MI-BMI can be operated directly on the edge, exclusively using fixed-point operations, on a low-power embedded platform. In the future, the proposed technique of interleaved layers can be included in automatic code generators/compilers for deep learning inference, such as Apache TVM [20], Google MLIR [21], or DORY [22], to overcome the layer-by-layer implementation paradigm. Moreover, even lower bit representations and mixed-precision inference can be further explored.

REFERENCES

[1] A. A. Frolov, O. Mokienko *et al.*, "Post-stroke Rehabilitation Training with a Motor-Imagery-Based Brain-Computer Interface (BCI)-Controlled Hand Exoskeleton: A Randomized Controlled Multicenter Trial." *Frontiers in neuroscience*, vol. 11, p. 400, 2017.

[2] N. Kobayashi and M. Nakagawa, "BCI-based control of electric wheelchair using fractal characteristics of EEG," *IEEJ Tran. on Electrical and Electronic Engineering*, vol. 13, no. 12, pp. 1795–1803, 2018.

[3] K. K. Ang, Z. Y. Chin *et al.*, "Filter bank common spatial pattern (FBCSP) in brain-computer interface," in *Proc. IEEE IJCNN*, 2008.

[4] M. Hersche, T. Rellstab *et al.*, "Fast and Accurate Multiclass Inference for MI-BCIs Using Large Multiscale Temporal and Spectral Features," in *Proc. IEEE EUSIPCO*, 2018, pp. 1690–1694.

[5] R. T. Schirrmeister, J. T. Springenberg *et al.*, "Deep learning with convolutional neural networks for eeg decoding and visualization," *Human brain mapping*, vol. 38, no. 11, pp. 5391–5420, 2017.

[6] F. Lotte, L. Bougrain *et al.*, "A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update," *Journal of Neural Engineering*, vol. 15, no. 3, p. 031005, 2018.

[7] V. J. Lawhern, A. J. Solon *et al.*, "EEGNet: a compact convolutional neural network for eeg-based brain–computer interfaces," *Journal of neural engineering*, vol. 15, no. 5, p. 056013, 2018.

[8] D. Li, J. Wang *et al.*, "Densely Feature Fusion Based on Convolutional Neural Networks for Motor Imagery EEG Classification," *IEEE Access*, vol. 7, pp. 132 720–132 730, 2019.

[9] P. D. Schiavone, F. Conti *et al.*, "Slow and steady wins the race? a comparison of ultra-low-power risc-v cores for internet-of-things applications," in *Proc. IEEE PATMOS*, 2017, pp. 1–8.

[10] M. Gautschi, P. D. Schiavone *et al.*, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Transactions on VLSI Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[11] A. Garofalo, M. Rusci *et al.*, "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, 2020.

[12] X. Wang, M. Magno *et al.*, "FANN-on-MCU: An Open-Source Toolkit for Energy-Efficient Neural Network Inference at the Edge of the Internet of Things," *IEEE Internet of Things Journal*, 2020.

[13] A. Pullini, D. Rossi *et al.*, "Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[14] X. Wang, M. Hersche *et al.*, "An Accurate EEGNet-based Motor-Imagery Brain-Computer Interface for Low-Power Edge Computing," *arXiv:2004.00077*, 2020.

[15] L. Cavigelli and L. Benini, "RPR: Random Partition Relaxation for Training; Binary and Ternary Weight Neural Networks," *arXiv:2001.01091*, 2020.

[16] C. Brunner, R. Leeb *et al.*, "BCI competition 2008 - Graz data set A," http://bnci-horizon-2020.eu/database/data-sets.

[17] M. S. Louis, Z. Azad *et al.*, "Towards Deep Learning using TensorFlow Lite on RISC-V," in *Proc. ACM CARRV*, 2019.

[18] B. Jacob, S. Kligys *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE CVPR*, 2018, pp. 2704–2713.

[19] X. Wang, "DSP library for PULP," https://github.com/pulp-platform/pulp-dsp, 2019.

[20] T. Chen, T. Moreau *et al.*, "TVM: end-to-end optimization stack for deep learning," *arXiv:1802.04799*, 2018.

[21] C. Lattner, M. Amini *et al.*, "MLIR: A Compiler Infrastructure for the End of Moore's Law," *arXiv:2002.11054*, 2020.

[22] A. Burrello, F. Conti *et al.*, "Work-in-progress: Dory: lightweight memory hierarchy management for deep nn inference on iot endnodes," in *Proc. IEEE CODES+ISSS*, 2019.