# Bandwidth-efficient distributed neural network architectures with application to neuro-sensor networks

Thomas Strypsteen and Alexander Bertrand, *Senior Member, IEEE*

***Abstract*—In this paper, we describe a conceptual design methodology to design distributed neural network architectures that can perform efficient inference within sensor networks with communication bandwidth constraints. The different sensor channels are distributed across multiple sensor devices, which have to exchange data over bandwidth-limited communication channels to solve a classification task. Our design methodology starts from a user-defined centralized neural network and transforms it into a distributed architecture in which the channels are distributed over different nodes. The distributed network consists of two parallel branches, whose outputs are fused at the fusion center. The first branch collects classification results from local, node-specific classifiers while the second branch compresses each node's signal and then reconstructs the multi-channel time series for classification at the fusion center. We further improve bandwidth gains by dynamically activating the compression path when the local classifications do not suffice. We validate this method on a motor execution task in an emulated EEG sensor network and analyze the resulting bandwidth-accuracy trade-offs. Our experiments show that the proposed framework enables up to a factor 20 in bandwidth reduction and factor 9 in power reduction with minimal loss (up to 2%) in classification accuracy compared to the centralized baseline on the demonstrated task. The proposed method offers a way to smoothly transform a centralized architecture to a distributed, bandwidth-efficient network amenable for low-power sensor networks. While the application focus of this paper is on wearable brain-computer interfaces, the proposed methodology can be applied in other sensor network-like applications as well.**

***Index Terms*—Deep neural networks, Distributed deep neural networks, EEG, Wireless EEG sensor networks, Body sensor Networks**

## I. Introduction

### A. Context and contributions

In the last few years, technological advances such as miniaturization of microprocessors and energy-efficient batteries have increasingly enabled the usage of wearable, physiological sensors for ambulant health monitoring. Many applications however, will require recording of different data modalities or multiple channels of the same data type at different locations to extract meaningful patterns. This naturally leads to the concept of a body-sensor network (BSN), where the different sensors wirelessly share their data and solve a given task in a distributed fashion. Well-known applications include microphone arrays to detect heart and lung body sounds [1], electroencephalography (EEG) sensor networks [2,3] and other distributed or modular neuro-sensor platforms [4]. A major constraint in the design of these networks is that they should be energy-efficient, enabling a maximal battery lifetime. In BSNs, the typical energy bottleneck will be the wireless transmission of the data between the sensors and/or a fusion center [2,5,6]. Simply offloading all the recorded data to the cloud where they can be jointly processed will thus severely hamper the battery lifetime, presenting the need for different, bandwidth-efficient solutions [2,7]. In this paper, we will present a framework to design deep neural network (DNN) architectures that deal with such bandwidth constraints. The framework is generic, in the sense that we make no prior assumptions on the DNN architecture itself. We start from a user-defined centralized neural network model for inference from multi-sensor input, and explain how this initial model can be used to build a distributed model that solves the same inference task. This conceptual methodology is then illustrated and analyzed in an EEG-based brain-computer interface task, which acts as a driver application driver throughout this paper.

EEG is a widely used, noninvasive way to measure the electrical activity of the brain. These signals can be harnessed for various purposes, including the monitoring and analysis of sleeping patterns [8], epileptic seizure detection [9], the study of brain disorders after injuries [10] and brain-computer interfaces (BCI), which allows for direct communication between the human brain and external machines [11]–[13]. Traditional EEG requires patients to wear a bulky EEG cap with many wires that are connected to the acquisition device. This means that monitoring the patient's EEG can typically only be done in a hospital or laboratory environment. These limitations of classical EEG have led to a growing desire for ambulatory EEG, allowing for continuous neuromonitoring in daily life [14]. A major enabler for these purposes is the development of mini-EEG devices: concealable, lightweight, miniaturized devices that are deployed behind or in the ear [15]–[17] or attached to the scalp [18,19]. A single device

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 802895). The authors also acknowledge the financial support of the FWO (Research Foundation Flanders) for project G.0A49.18N, and the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

T. Strypsteen and A. Bertrand are with KU Leuven, Department of Electrical Engineering (ESAT), STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics and with Leuven.AI - KU Leuven institute for AI, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium (e-mail: thomas.strypsteen@kuleuven.be, alexander.bertrand@kuleuven.be).

would only be able to record one or a few EEG channels from its local area, hampering the performance in many of the previously mentioned applications. To mitigate this, multiple mini-EEG devices at different locations can be organized in a so-called wireless EEG sensor network (WESN) [2,19,20]. Each device can then perform some local processing on its own channels, before sharing its information with the other devices to perform the original, centralized EEG tasks in a distributed manner. This shift from one EEG cap towards a network of wireless, miniaturized devices affects the design of the machine learning models we use to perform these tasks in two major ways. Firstly, to guarantee a comfortable user experience, we are only able to use a limited number of devices. We thus first need to solve an EEG channel selection task, determining *how many* and *where* these devices should be placed, minimizing the amount of devices, while maximizing the performance of the desired EEG task [20,21]. Secondly, the recorded channels are now stored on separate devices, meaning we cannot perform multi-channel processing without sharing the recorded data across the devices first. Simply transmitting the full, raw channels to a fusion center would incur enormous energy costs and severely hamper the battery life of the mini-EEG devices [2,5]. To achieve a viable battery life, we will thus need to limit the amount of data each device in the WESN can share and take this bandwidth constraint into account during the model design.

Recently, deep learning or DNN models have become more and more popular in the processing and analysis of physiological signals, including EEG [22]. This trend in combination with the shift towards low-power wearables cultivates a need to redesign such DNNs towards distributed architectures that can operate in modular sensor platforms, such as WESNs and other body-sensor networks. While a generic methodology for the first problem (i.e., channel selection and sensor placement for DNN-based inference) has been proposed in [21], generic methodologies for the second problem (i.e. translating DNNs to bandwidth-efficient modular architectures) are still largely lacking. In this paper, we study how we can adapt existing *centralized* DNN architectures to make them amenable for use in distributed settings with communication bandwidth constraints such as in body-sensor networks (and WESNs in particular). In the resulting distributed architecture, the sensor nodes learn to locally process the data, compress them to a desired degree and transmit them and finally fuse the compressed data to solve the desired task. To validate the applicability of this method, we study its performance on a motor execution EEG task and analyze the bandwidth-versus-performance tradeoff.

The main contributions of this paper are:

- We introduce a design framework that maps a given centralized neural network architecture to a distributed architecture that is able to run efficiently on a bandwidth-constrained sensor network.
- We combine this framework with the early exit mechanism of [23] to further decrease the bandwidth by deciding on a per-sample basis how much data needs to be transmitted to the fusion center.
- We demonstrate the usage of our method by taking a centralized neural network architecture solving a given motor execution EEG classification task and decentralizing it. We analyze the resulting bandwidth-accuracy trade-offs of the resulting distributed architecture and demonstrate that with only small performance losses compared to the centralized baseline, substantial bandwidth gains can be achieved.

### *B. Distributed deep learning: related work*

The literature of distributed deep learning is diverse and covers many different topics. A first class of methods distributes networks across multiple compute nodes, either to enable training of a single very large network that would otherwise not fit in memory on multiple standard CPU's or GPU's [24] or to accelerate training by training a network on multiple devices in parallel and aggregating the gradient updates on each device [25]. A second line of research aims to map centralized models to a number of hardware devices to perform efficient inference. For instance, Bhardwaj et al. [26] employ multiple model compression techniques to map a single network to a number of smaller student networks with a limited memory footprint, while also minimizing the inter-device communication cost. Stahl et al. [27] have a similar goal, but instead employs layer partitioning to perform the exact same operations as in the original network, but spread these out across devices.

All the previous work has one major factor in common, which makes them not applicable to our problem statement. They share the assumption that either all devices have access to all the input data or all the input data are generated in a central location and the energy of communicating this data to the worker nodes is not a constraint. The literature on deep learning where different channels or modalities of the input data itself is split across different devices is quite limited. The closest work to ours in this regard is the distributed deep neural network (DDNN) framework of Teerapittayanon et al. [23]. Similarly to our setting, the input data is distributed across devices. The local classifications of each device are aggregated and the confidence in this prediction is estimated with the normalized entropy of the resulting class probability distribution. If the confidence is high enough, this result - which only required the transmission of a classification vector of each node - is taken as the final result. Otherwise, a processed version of the data of each local device is forwarded to the cloud, thereby requiring a larger bandwidth. The main idea is thus to reduce bandwidth by only forwarding difficult samples that can't be correctly classified locally. Designing the optimal bandwidth-performance trade-off for a given application is then done by setting the desired confidence threshold of the local classification, with higher required confidence resulting

in more data streamed to the cloud, but fewer misclassifications. In contrast, the main focus of this work will be to reduce bandwidth by designing an efficient architecture that compresses the data on our nodes to the desired degree, as will be described in the next section. However, both approaches are orthogonal to each other and we will ultimately combine them to gain even greater bandwidth gains without losing too much accuracy.

### C. Paper Outline

The paper is organized as follows. In section II we introduce our framework and show how to combine it with the early exiting mechanism of [23]. Section III presents the WESN use case, providing an overview of the used EEG dataset, how we emulate the environment of a WESN and design the neural network architecture for this specific use case. We then present our experimental results in section IV and finish with some conclusion in section V.

## II. Proposed Method

In this section, we will conceptually describe the proposed bandwidth-efficient distributed architecture. We will first give a conceptual overview of the proposed architecture in Subsection II-A, while in Subsection II-B, we will explain in more detail how a centralized neural network can be cast to this distributed architecture and how it is trained. In Section III and IV, we will then apply this architecture design framework to a specific EEG inference task.

### A. Proposed architecture

To build an architecture that minimizes the communication cost in a wireless sensor network, we propose a scheme where each local sensor (henceforth referred to as a node) compresses its recorded data as much as possible before sending it to a fusion center, where the final processing and inference will take place. The idea is to design an architecture that is able to interpolate between the two extreme cases of minimal and maximal communication, which also corresponds to minimal and maximal task accuracy. As the point of minimal communication, we take the setting where each node only transmits its local classification, as this would reasonably be the most condensed task-relevant information it could share. This setting, represented by the *ClassFuse* branch (orange path in Figure 1), will serve as the basis of our architecture, with the other modules serving to trade extra bandwidth for an improved performance compared to this minimum-communication baseline. The point of maximal communication corresponds to each node simply transmitting its full recorded data. This would allow the fusion center to perform the same multi-channel processing as in the centralized case and achieve maximal accuracy. However, to achieve a trade-off between bandwidth and performance, we compress each signal at the local node, after which they are reconstructed at the fusion center. This is the task of the *CompressFuse* branch (blue path in Figure 1). This *CompressFuse* branch essentially mimics the original centralized network, although it operates on data that is distorted through the compression-reconstruction scheme. Finally, the results of the two branches are fused to provide a final output. A high-level schematic of such an architecture is illustrated in Figure 1. Another way to look at this architecture is as a combination of *early* and *late fusion*, respectively represented by the *CompressFuse* and *ClassFuse* branches. We will now delve deeper into the design of these modules for our WESN case and how they are trained.

### B. Design of the modules

In this subsection, we will delve deeper into how we can use this framework to transform a given centralized architecture that has access to all input channels simultaneously, into a decentralized version that performs the same task.

*1) ClassFuse:* The task of the *ClassFuse* branch (orange path in Figure 1 is to let each node perform local classification and optimally fuse them together at the fusion center. The local classifications are performed with the original centralized architecture,where the input dimensions are reduced with respect to the number of local channels at each node. Each node then outputs the class scores as a log-probability vector (i.e. the classification output before applying softmax) to the fusion center. At the fusion center, these probability vectors are fused into a final class probability vector. This fusion is performed with a simple multilayer perceptron (MLP) with 1 hidden layer and Rectified Linear Unit (ReLU) nonlinearities on the concatenated outputs of the nodes. We take advantage of the modular nature of this network to train it in two stages. First, the weights of the local classifiers are pre-trained with a single-channel classification task. Then, the full *ClassFuse* branch is trained end-to-end, with the weights of the local classifiers having a lower learning rate due to previously being pre-trained.

*2) CompressFuse:* The task of the *CompressFuse* branch (blue path in Figure 1 is to compress each local recording, reconstruct the full multi-channel signal at the fusion center and classify this with the original, centralized neural network. To compress the local sensor channels at each node, we use two strided convolutional layers, with the value of the strides together determining the amount of compression (e.g. a stride of 2 and a stride of 3 resulting in a downsampling with a factor 6). Note that the downsampling could also be performed by a single strided convolutional layer, but we empirically found that using two two layers resulted in more stable training. We then upsample each channel separately with two transposed, strided convolutional layers, mirroring the strides of the compression layers. Another possibility would be to omit the reconstruction step (i.e. upsampling) alltogether and directly classify the compressed, downsampled signal, since the reconstructed signal will not contain more useful information than the compressed signal. However, when employing an existing neural network for classification, hyperparameters such as the length of the kernels have been tuned assuming a specific length of the time window at the

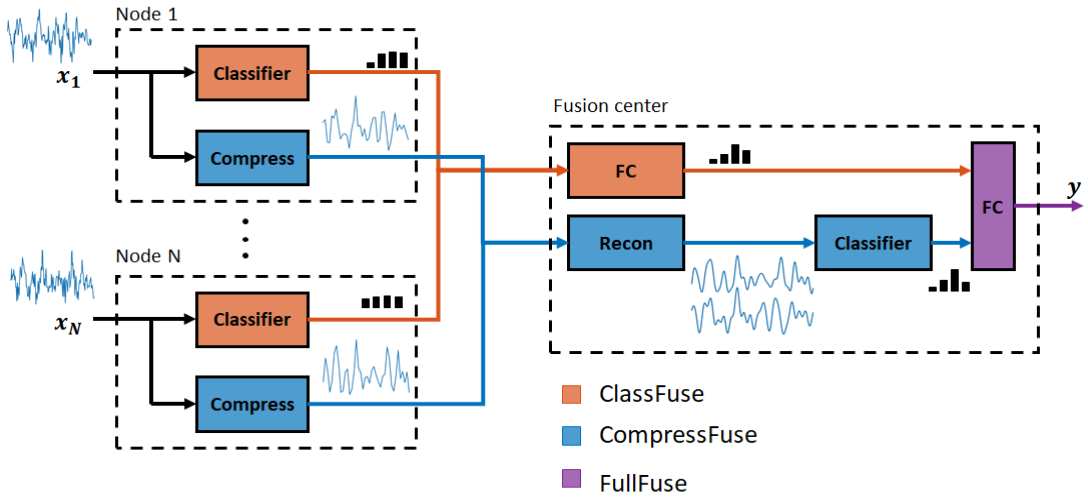


Fig. 1: Illustration of the distributed neural network architecture and its modules. The orange *ClassFuse* branch lets each node perform its own local classification with a single-channel neural network ('Classifier' block) and combines these at the fusion center with fully connected layers ('FC'). This branch only transmits a class probability vector. The blue *CompressFuse* branch compresses each channel locally ('Compress') and reconstructs the full multi-channel signal at the fusion center ('Recon'), only having to transmit a downsampled version of the original signal. The full reconstructed signal is then classified by the multi-channel neural network ('Classifier'). The purple *FullFuse* module combines these two branches and performs the final classification with another pair of fully connected layers ('FC'). By tuning the compression rate in the *CompressFuse* branch, we can reduce the amount of data to be transmitted, while the *ClassFuse* branch helps boost the accuray at very little cost in bandwidth.

input and a certain desired receptive field. Not employing reconstruction would break these assumptions and force us to redesign the original centralized classification network, which we aim to avoid as much as possible. Similary to the *ClassFuse* branch, it is possible to perform the training of this branch in multiple stages. We could, for instance, first pre-train the compression-reconstruction layers as an auto-encoder by minimizing the mean squared error (MSE) between the reconstructed output and the input and then train the full *CompressFuse* end-to-end. Whether this two-step training will be necessary, will largely depend on the size of the compression network compared to the classification network.

*3) FullFuse:* The *FullFuse* module combines the classifications of the *ClassFuse* and the *CompressFuse* branches to perform the final classification. Similarly to the *ClassFuse*, we simply use an MLP with 1 hidden layer and ReLU nonlinearity for this task. We train this module jointly with the previously trained *ClassFuse* and *CompressFuse*, once again employing a lower learning rate for the latter two. In Section III, we will demonstrate that the output of *FullFuse* obtains a higher accuracy than both the *ClassFuse* and *CompressFuse* branch separately.

In summary, the training of the network is thus comprised of the following steps:

1) Train the single-channel local classifiers of each node.
2) Train the full *ClassFuse* branch, combining the local classifications and fine-tune the local classifier weights.
3) Train the *CompressFuse* branch, which compresses, reconstructs and classifies the node signals jointly.
4) Train the entire network end-to-end, including the *FullFuse* module, which combines the two previous branches to perform the final classification.

In Subsection IV-C, we will demonstrate the importance of using this piece-wise pre-training scheme, by comparing it to a direct end-to-end training from scratch.

### C. Early exiting

The *ClassFuse* branch allows us to reach a certain, basis classifcation accuracy with minimal communication, while the *CompressFuse* allows us to send additional information to boost this accuracy further. However, when the *ClassFuse* is able to already correctly classify a substantial fraction of the samples on its own, this implies that for many of these samples, the extra information of the *CompressFuse* branch is not necessary for a correct classification. Thus, we can save additional bandwidth by only transmitting the data for the *CompressFuse* when we are not confident that the *ClassFuse* has already successfully predicted the label of the current sample. This idea of allowing samples to exit the network early has previously been employed to reduce inference time [28] and in the Distributed Deep Neural Network (DDNN) framework of [23] to decide whether a sample is processed locally or in the cloud. A common metric for classification confidence in this line of work, which we will employ here as well, is the normalized entropy of the softmaxed classification vector, defined as

$$H(\boldsymbol{x}) = -\frac{1}{\log|C|}\sum_{i=1}^{|C|} x_i \log(x_i). \tag{1}$$

with $|C|$ the number of classes and $\boldsymbol{x}$ a probability vector, which in this case is the softmaxed output vector of the

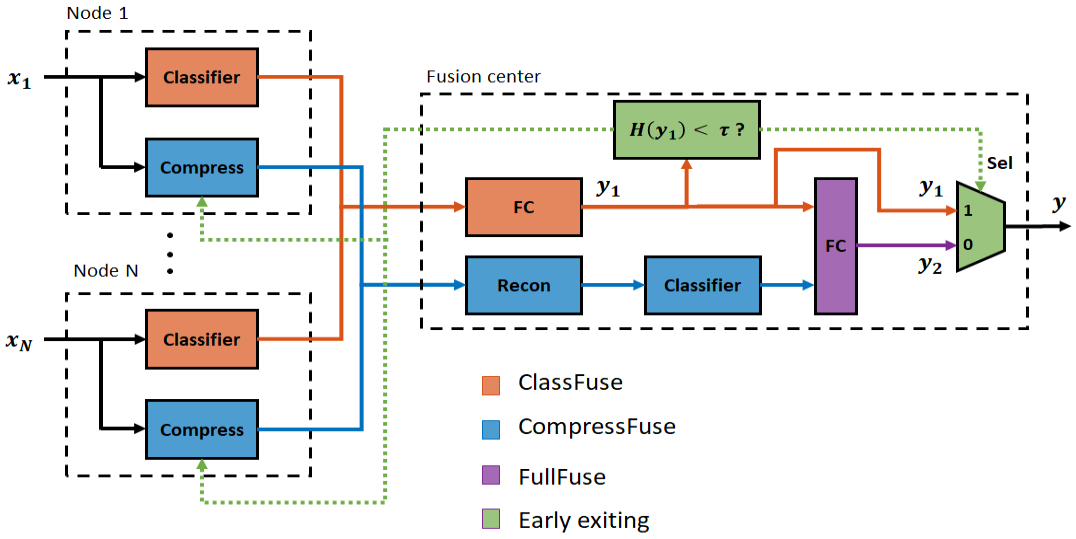


Fig. 2: Illustration of the distributed neural network architecture extended with early exiting. First, the *ClassFuse* delivers a first prediction $y_1$, requiring a negligible amount of bandwidth. If the entropy of the class distribution of $y_1$ is low enough, i.e., the confidence of the *ClassFuse* is high, $y_1$ is taken as the final classification output. If the *ClassFuse* is insufficiently confident, the nodes are signaled to enable the *CompressFuse* branch as well, whose prediction is fused with with $y_1$ to create a more accurate prediction $y_2$, at the cost of more bandwidth being consumed.

*ClassFuse* branch. Similar to [23], we thus first perform classification using the *ClassFuse* branch and measure the entropy of the current sample's output. If the entropy is lower than a certain threshold, we keep this output. If the entropy threshold is exceeded, we activate the *CompressFuse* and combine this with the *ClassFuse* to perform inference on the full network. The value of this threshold introduces a trade-off which can easily be tuned after network training: the higher we put this threshold, the less frequently the *CompressFuse* branch will be activated, thereby saving more bandwidth at the cost of a reduced classification performance. Figure 2 shows a schematic overview of this mechanism in our distributed architecture. A major advantage of combining early exiting with our bandwidth-efficient architecture design is that, in contrast to the compression factor of our *CompressFuse* branch, the confidence threshold is a continuous parameter, thus allowing us to perform the bandwidth-accuracy trade-off in a continuous manner rather than a discrete one. The efficient architecture design on the other hand, allows us to start this trade-off from a more favorable point than we could otherwise.

## III. Case study: Wireless EEG Sensor networks

In this section, we investigate the use of our distributed architecture in the context of a BCI task in a wireless EEG sensor network. We use data from a motor execution classification task, which is a well-known EEG-BCI paradigm for which large data sets as well as mature deep neural network architectures are available.

### A. Data set

Motor execution is a widely used paradigm in the field of BCI. Real or intended body movement typically goes hand in hand with neuronal activity in certain motorsensory areas of the brain. The goal of motor execution is then to derive from these signals which movement was performed. In this work, we will employ the High Gamma Dataset [29], containing a training set of 880 trials of executed movement following a visual cue, for each of the 14 subjects. We employ 80% of these for training and 20% as a validation set for early stopping. The dataset also contains a separate test set of 160 trials per subject, which we use to validate our results. The movements to be decoded are divided in 4 classes: left hand, right hand, feet and rest. While originally 128 channels were recorded for this dataset, we follow the approach of [29] and perform our experiments using only the 44 channels covering the motor cortex. The rest of our preprocessing procedure also follows the work of [29]:

- Resampling to 250 Hz
- Highpass-filtering at 4 Hz
- Standardizing mean and variance per channel to 0 and 1
- Epoching in segments of 4.5 seconds, consisting of the 4 seconds after the visual cue and the 0.5 before.

The neural network architecture we employ for classification - and the one we will convert to a distributed architecture for our WESN - is the multiscale parallel filter bank convolutional neural network (MSFBCNN) proposed in [30]. For completeness, a detailed summary of this network in table format can be found in Appendix A.

### B. WESN node emulation

In traditional EEG caps, a channel is usually measured as the potential between an electrode at a given location and a common reference, typically the mastoid or Cz electrode. However, in the case of mini-EEG devices, we can only

measure a local potential between two proximate electrodes belonging to the same device. To emulate this setting based on a standard cap-EEG recording, we follow the approach of [20]. In this setting, each pair of electrodes within a preset maximum inter-electrode distance from each other is a candidate electrode pair or node we could measure. The signal this node records is then emulated by subtracting one of the channels from the other, thus removing the common (far-distance) reference in the process. We applied this method with a distance threshold of 3 cm to our dataset, converting the 44 channels in 286 candidate electrode pairs or nodes. The resulting set of nodes had an average inter-electrode distance of 1.98 cm with a standard deviation of 0.59 cm.

### C. Node selection

Since we are only able to use a limited number of mini-EEG devices, we will first perform a channel/node selection step to select the most relevant sensor nodes from the pool of 286 candidate nodes. To this end, we employ the regularized Gumbel-softmax method described in [21]. This method allows us to learn the $M$ optimal nodes for a given task and neural network by training said network jointly with a special selection layer that is able to learn the discrete variables involved in feature selection through simple backpropagation. The value of $M$ will also be varied throughout our experiments. We jointly train this selection layer of size $M$ with the centralized MSFBCNN architecture using the training data from all subjects in the data set and selecting the model weights and candidate nodes that reach the best performance on the validation set. This results in a subject-independent set of $M$ mini-EEG nodes that are optimally placed to solve the motor execution task. The $M$ selected nodes are then used to design a distributed version of the MSFBCNN network as explained next. For more details on the exact procedure of the channel selection step, we refer the reader to [21], where the same procedure is applied to channels instead of candidate nodes.

### D. Distributed architecture design

We build our distributed network by taking the MSFBCNN architecture as our centralized baseline. Thus, we employ a single-channel version of the MSFBCNN as our classifier on the local nodes and the multi-channel version as our classifier in the fusion center in the *CompressFuse* branch (this corresponds to all the blocks denoted as 'classifier' in Figure 1). The MLP that fuses our local classifications in the *ClassFuse* branch (the first orange FC block in Figure 1), consists of a simple MLP with one hidden layer of size 50 and ReLU nonlinearity in between. We use the same MLP architecture to fuse the output of the *ClassFuse* and *CompressFuse* (i.e. the purple FC block in Figure 1). The 'Compress' block consists of two convolutional layers, each consisting of a single kernel with strides to match a desired compression factor (e.g. one stride of 2 and one of 3 to achieve a compression factor 6). The 'Recon' block is built symmetrically to the 'Compress' block, with transposed convolutions replacing the normal convolutions. Since the reconstruction happens at the fusion center, it would also be possible to jointly reconstruct the channels using spatiotemporal filters instead, though our experiments indicated no advantage in this approach for our application.

The distributed network is trained using the data of all subjects jointly, using the procedure described in Section II-B. Training is performed with the Adam optimzer [31], a learning rate of 0.001 and a batchsize of 64 for 50 epochs. Early stopping when the validation loss does not decrease for 5 epochs is employed to prevent overfitting. As soon as a layer has been trained for the first time, all subsequent fine-tuning of said layer will use a learning rate of only $10^{-4}$, a tenth of the original learning rate. When the full network has been trained, subject-dependent decoders are obtained by fine-tuning the full network end-to-end with subject-specific data. The performance of each of these subject-dependent decoders is then measured on the test set of the corresponding subject and we report the average performance across the subjects.

## IV. Experimental Results

### A. Impact of short-distance nodes

First, we take a look on how much using short-distance nodes instead of channels built from far-distance electrodes (with a common reference) impacts the accuracy of our motor execution task in the centralized case. Figure 3 compares the subject-dependent accuracy of training the centralized baseline on the $M$ optimal mini-EEG nodes and the $M$ optimal Cz-referenced channels. Clearly, using electrodes only 2 to 3 centimeters apart from each other significantly affects the motor execution accuracy. These performance drops have also previously been observed in the field of auditory attention decoding [32], though only when the average distance between the electrodes becomes smaller than 3 cm. As observed in Figure 3, the difference between short-distance electrode pairs (nodes) and the original cap-EEG data tends to decrease when using more nodes, which is consistent with the observations in [20,32]. We do not consider networks consisting of more than 8 nodes because the model's accuracy does not strongly increase beyond this point and employing a neuro-sensor network with more nodes would become impractical.

### B. Distributed architecture

Next, we compare the performance of the proposed distributed architecture using different compression factors to the centralized baseline and investigate the individual and combined contribution of both branches. The results are summarized in Figure 4. A first observation is that, while the *ClassFuse* is clearly less accurate than the centralized baseline, it still achieves reasonable accuracy considering it only requires the nodes to transmit a probability vector of size 4 (due to the 4-class task) compared to a full window of size 1125 (4.5 seconds sampled at 250Hz). A second observation is that the fusion of the *ClassFuse* and *CompressFuse* branches

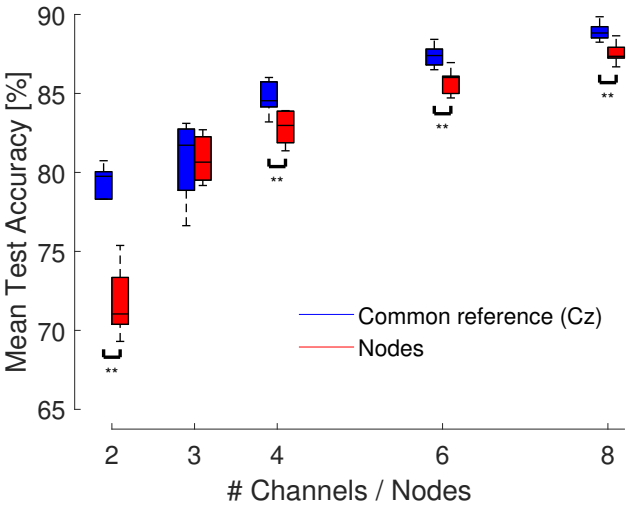


Fig. 3: Comparison of the subject-dependent centralized motor execution accuracy when using $M$ short-distance nodes and $M$ Cz-referenced channels. Mean test accuracies across the subjects are plotted as a function the number of channels/nodes. The displayed boxplots are computed over 10 runs and compared with independent samples t-test (no correction for multiple comparison) after confirming the normality of the data with a Kolmogorov-Smirnov test . $**$ indicates statistically significant difference with $p < 0.005$.

consistently and significantly outperforms the two separate branches, resulting in a *FullFuse* that is competitive with the centralized baseline despite its much lower bandwidth usage. When moving to higher compression factors such as 16 however, the *CompressFuse* has more and more trouble reconstructing the original EEG signal, especially at a lower number of nodes. At this point, its performance even drops below the *ClassFuse* performance, while consuming more bandwidth. Remarkably, even at this stage it is still beneficial to fuse the two branches, suggesting that the information provided by the two branches is complementary. Thirdly, using more nodes results in a slowly increasing gap between the centralized baseline and the distributed architecture, since the unconstrained baseline is naturally more able to exploit the spatial correlations across the nodes. Finally, in terms of actual bandwidth gains, the efficient architecture design allows us to reach similar performance as the original network at 11% of the original bandwidth and even with 6% bandwidth, accuracy merely drops from 87% to 82% in the worst-case scenario.

### C. Impact of pre-training

To demonstrate the importance of the proposed training scheme, we also compare the performance of our network modules with and without this pre-training. As illustrated in Figure 5, the network accuracy severely drops for the *ClassFuse* and especially for the *FullFuse* when training from scratch. This implies that the increased complexity of the distributed architecture indeed necessitates a custom training scheme taking advantage of its modular nature to train the network piece-wise. Though not shown Figure 5, it should be noted that the *CompressFuse* branch on the other hand, does not require pre-training at all, due to the small amount of parameters in the currently used compression-reconstruction layers. However, it stands to reason that pre-training in this branch might become necessary as well when deeper and more complex architectures are used in this branch.

### D. Early exiting

Now that we have a more bandwidth-efficient architecture, we employ early exiting to let the network decide which samples are processed by the bandwidth-friendly *ClassFuse* only and which by the complete *FullFuse* network. By tuning the required confidence threshold between 0 (all samples are handled by the full network) and 1 (all samples are processed by *ClassFuse* only) we can explore the accuracy-bandwidth trade-off in a continuous manner (instead of being confined to discrete non-prime compression factors) and find Pareto-optimal points, i.e., points where we cannot improve bandwidth or accuracy without sacrificing the other. We perform this trade-off for our distributed architecture with varying compression factors in Figure 6. Each point in this plot corresponds to a network with $M$ nodes, compression factor $D$ and local exit confidence threshold $T$ (varied from 0 to 1 with a step size of 0.01), which in turn corresponds to a percentage of samples handled by the *ClassFuse* alone, denoted by $\lambda(T)$. We compute the per-node bandwidth of this point, relative to the bandwidth required to run the centralized network (i.e. continuously transmitting the full recorded data window of length $L$ at each node). This relative per-node bandwidth B can be computed as:

$$B(T) = \frac{1}{L}\left(|C| + (1-\lambda(T))\frac{L}{D}\right). \tag{2}$$

with $|C|$ the amount of classes, i.e. the size of the class probability vector (in this case 4).

A first observation to be made from the bandwidth-accuracy curves is that often, bandwidth can be reduced up to 50% without any loss in accuracy. Interesting to note is that the deflection point at which the accuracy starts decreasing tends to shift more to the left, the more nodes we employ. This is not surprising, since more nodes implies a higher accuracy of the *ClassFuse* branch, thus less samples for which the full network has to be activated. The advantage of using multiple nodes is thus twofold: it increases accuracy due to the higher amount of recorded data (see Figure 3), but also allows us to save more bandwidth *per node* by requiring samples to pass through the whole network less often. Thus, instead of using nodes for increased accuracy, we can also employ them to save per-node bandwidth for the same accuracy. For instance, while using 3 nodes allows us to reach 80% accuracy at 11% of the original bandwidth, using 6 nodes allows us to do so at 1.3% of the original bandwidth. A second observation is that, when requiring low bandwidths, starting from a more bandwidth-efficient network with higher compression factors in the *CompressFuse* branch and applying early exiting generally outperforms applying early exiting to a network with smaller

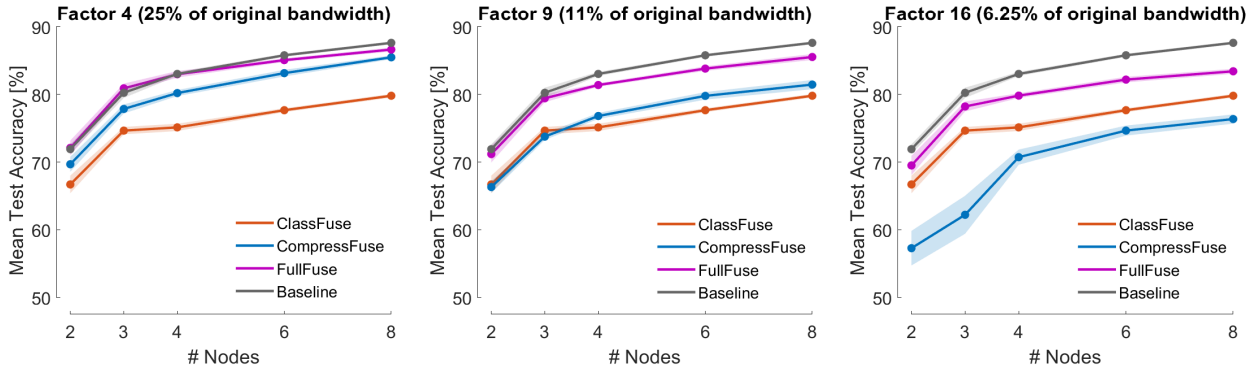


Fig. 4: Comparison of the distributed architecture with different compression factors in the *CompressFuse* branch. Each compression factor was obtained by two strided convolutions, with each stride equal to the square root of the compression factor. Mean test accuracies across the subjects are plotted as a function the number of nodes and averaged over 10 runs. Shades indicate standard error of the mean.

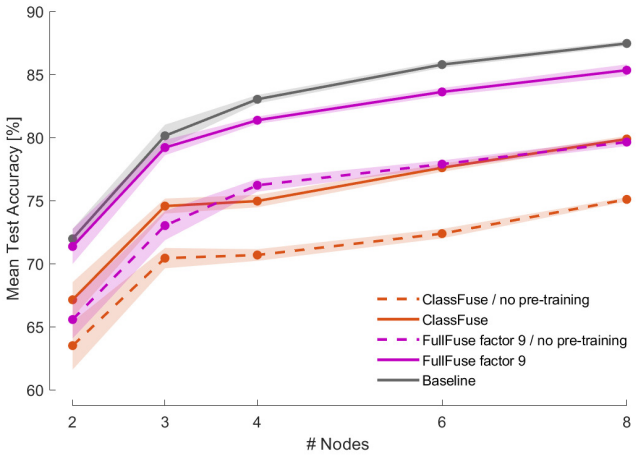


Fig. 5: Effect of pre-training on the *ClassFuse* and *FullFuse* with compression factor 9 in the underlying *CompressFuse* branch. Mean test accuracies across the subjects are plotted as a function the number of nodes and averaged over 10 runs. Shades indicate standard error of the mean.

compression factors. This is especially salient when comparing using *CompressFuse* branches with compression (yellow,green and red in Figure 6) to the version without compression (in blue), which corresponds to transmitting the raw sensor data and using the centralized baseline when the *ClassFuse* is not confident enough. It is noted that these gains tend to decrease as we increase the compression factor,yet the non-compressive version remains outperformed by the compressive versions of *CompressFuse*. Finally, we can observe that the bandwidth gains of combining the efficient architecture design and the early exiting are substantial, allowing the network to operate at only 5% of the original bandwidth, while never losing more than 2% accuracy at that point. This demonstrates how the gains obtained by our proposed distributed architecture and those obtained by early exiting are complementary.

### *E. Impact on power consumption*

Finally, we will analyze the impact of these bandwidth-reduction schemes on the total power consumption on the nodes, which will consist of the sum of two major parts. Firstly, the reduced bandwidth results in a decreased transmission power $P_T$ compared to simply offloading the raw data. Secondly however, the presence of the per-node classifier and the compression that enable this reduced bandwidth also results in an increase in on-node computing power consumption $P_C$. We estimate $P_C$ as:

$$P_C = \frac{N_{ops}/T_w}{P} \tag{3}$$

with $N_{ops}$ the amount of floating point operations (FLOPs) to perform inference on one input using the neural networks on each node, $T_w$ the length of one window in seconds and $P$ the efficiency of the processor, a technology parameter expressed in GOPS/W[1]. $P_T$ meanwhile is estimated as:

$$P_T = \frac{N_b BLC}{T_w} \tag{4}$$

with $N_b$ the number of bits per sample, $L$ the number of samples within a single input window, $B$ the relative per-node bandwidth savings achieved by the distributed architecture (defined in Eq. (2)) and $C$ the efficiency of the transmitter, a technology parameter expressed in nJ/bit. Table I summarizes the values employed in our simulations.

TABLE I: Constants employed for the power estimation

| | |
|---|---|
| $N_b$ | 32 bit |
| $N_{ops}$ | $\approx 4.10^6$ FLOPs |
| $T_w$ | 4.5s |
| $L$ | 1125 |
| $C$ | 1.9 nJ/bit [33] |

We fix the value of $C$ to 1.9 nJ/bit [33] and investigate how efficient our on-node processing has to be to make sure the reduced transmission power compensates for the increased computing power. To do this, we compare the total reduction in power compared to offloading all the raw data

[1]Note that we follow the convention that FLOPs indicate the absolute amount of floating point operations, while FLOPS (all capitals) indicate floating point operations per second.

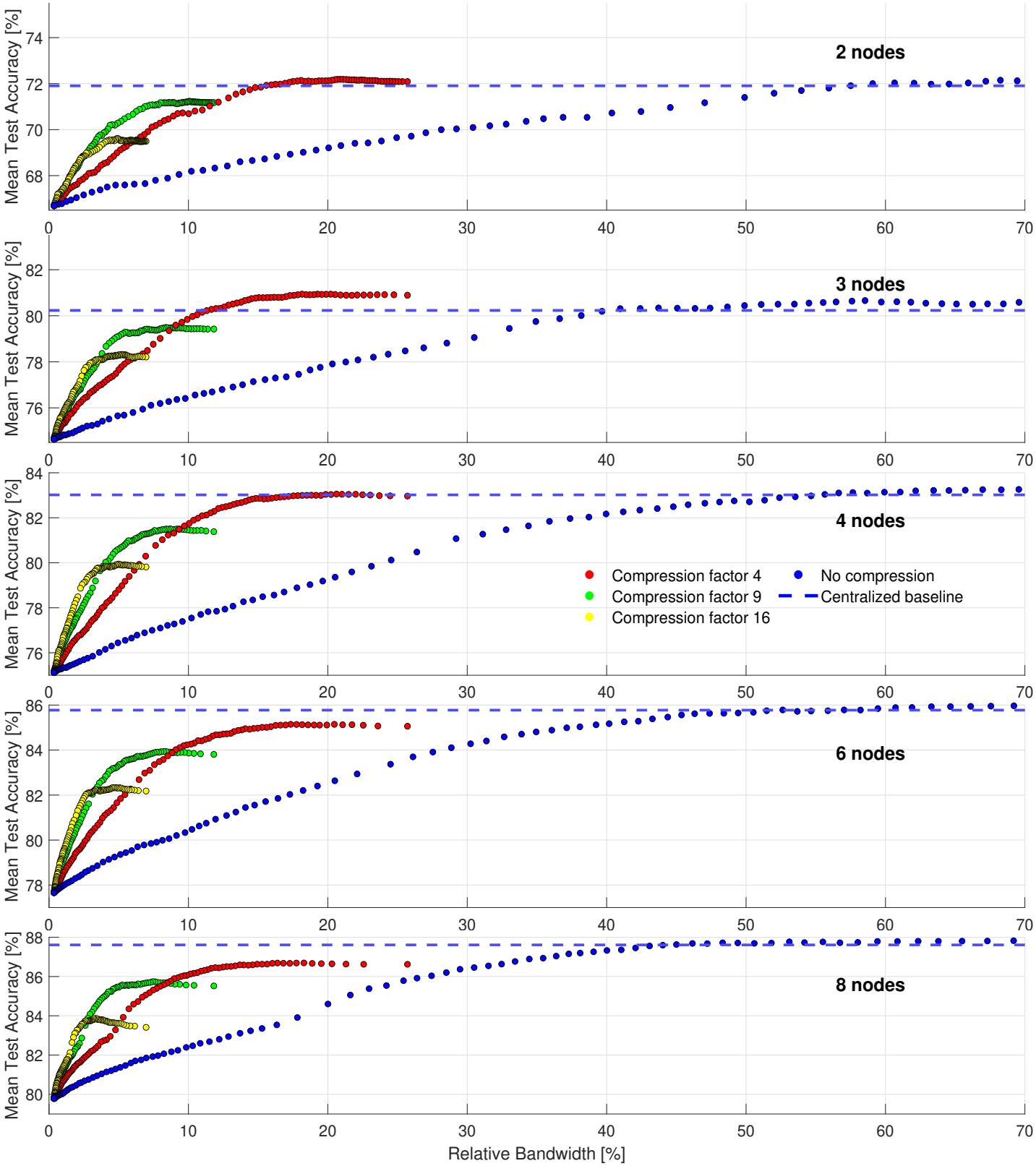


Fig. 6: Bandwidth-accuracy trade-offs when applying early exiting to the distributed architecture with different compression rates of the *CompressFuse* branch and for a different number of nodes. Bandwidth is measured as the average size of the data vector transmitted by each node relative to the full window size of each epoch, as determined by Eq. (2). Mean test accuracies across the subjects are averaged over 10 runs. Dashed lines indicate centralized accuracy.

(which is equivalent to Eq. (4) with $B = 1$) for different bandwidth reductions (as shown in Eq. (2) and Figure 6) and different processor efficiencies $P$ in Figure 7. We see that for very inefficient processors, the extra overhead of the on-node computations outweigh the gains achieved by the reduced bandwidth and no power reduction is achieved. Conversely, in the case of efficient processors, the computing power is negligible and the total power reduction in power is equal to the reduction in bandwidth. The more bandwidth-efficient a network is, the more efficient a processor needs to be to fully realize these power savings, since transmission will become less and less of a bottleneck. Realistic neural network accelerator ASICs today are known to be capable of reaching 1 TOPS/W and higher [34][2]. This means that for instance in our use case, working with a bandwidth reduction of a factor 20 (i.e, $B = 1/20$ in Eq. (2), corresponding to the region of 5% relative bandwidth in Figure 6), corresponds to a total power reduction factor of about 9 (see Figure 7 with $B = 1/20$), showing that despite the computational overhead, significant power gains can still be reached. Despite this strong compression and power reduction, the accuracy drops with at most 2% for any number of nodes (see Figure 6).

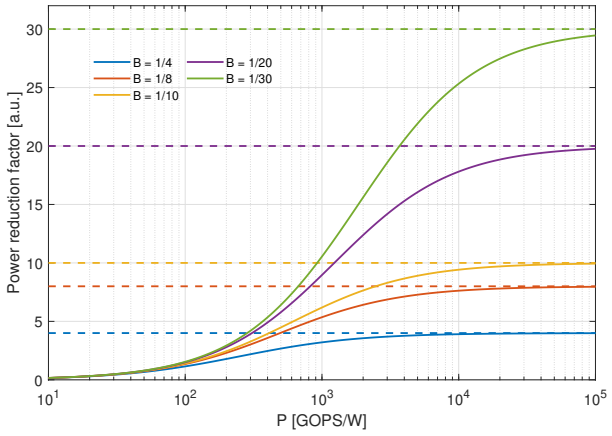


Fig. 7: Total reduction in power consumption per node for *FullFuse* networks with different bandwidth reductions as obtained by Eq. (2) and different processor efficiencies $P$. A power reduction factor $F$ indicates each node in the the network consumes $\frac{1}{F}$ of the power a node consumes when it would directly offload all the raw data. Dotted lines indicate power reduction when on-node compute power is neglected and only power savings in transmission costs are considered. The more efficient the processor, the lower the power cost of the computational overhead and the closer the power gains are to the bandwidth gains.

## V. Conclusion and Future Outlook

We have proposed a novel distributed neural network architecture design framework that can straightforwardly be mapped on a wireless sensor network and perform inference in this setting in a bandwidth-efficient manner. While we have applied it to the specific case of BCI in WESNs, the nature of this architecture is generic. The architecture consists of two parallel branches. The *ClassFuse* branch lets each node in the network perform its own local classification and then aggregates these in a fusion center. The purpose of this late fusion procedure is to produce reasonable classifications while consuming the minimal amount of communication energy. To then be able to perform a trade-off between bandwidth and performance, the *CompressFuse* branch compresses the recorded sensor signal of each node to a desired level and then approximately reconstructs the full multi-channel signal at the fusion center, where it can then be classified by a centralized network. These outputs of these two branches are then fused to perform the final classification.

The *FullFuse* requires two rather mild assumptions on the settings to which it can be applied, which will generally be met in a practical neuro-sensor network or WESN context. Firstly, the individual nodes should be able to obtain some reasonable classification above chance level when only using local data in order for the *ClassFuse* branch to provide relevant data to the fusion center. Secondly, some useful inter-node correlations have to be present in order for the *CompressFuse* branch to provide additional information compared to the *ClassFuse* branch via the early fusion principle.

To train the *FullFuse* architecture, we have proposed a step-by-step procedure, taking advantage of the modular structure of the architecture to first pre-train every block separately. We have experimentally demonstrated both the need and the advantage of training the network in this way. We have then combined the resulting network with the early exiting mechanism of [23] to decide on a per-sample basis whether to use the full network or the very bandwidth-friendly *ClassFuse* to process the current input. We have shown that the introduction of the *CompressFuse* branch allows to substantially push the Pareto-front upwards, in particular in low-bandwidth regimes.

We have validated the performance of our architecture on an emulated WESN solving a motor execution EEG task. We have used our architecture to obtain accuracy-bandwidth curves for this task, showing that for a realistic amount of nodes, we could save a factor 20 in bandwidth and a factor 9 in power consumption at the cost of 2% mean test accuracy proving that good motor execution performances can be reached with both a low number of channels and a high reduction in the amount of data that needs to be transmitted from the nodes. An important observation in our experiments is the advantage of using multiple nodes in the sensor network. Not only does using more nodes increase accuracy, it also leads to a more favorable bandwidth-vs-accuracy trade-off, which in the case of WESNs implies an increased battery life. In the future, we will explore ways to reach even higher reductions by using more sophisticated architectures for the *CompressFuse* branch, which is currently a very simple model consisting of strided convolutions. We will also explore the generality of our findings on other EEG tasks, such as epileptic seizure detection [9] and other distributed platforms than WESNs.

[2]Though they often require 8-bit precision to achieve this, neural networks are also known to perform similarly in 8-bit precision as in full 32-bit precision [35].

## REFERENCES


[1] J. Kirchner, S. Souilem, and G. Fischer, "Wearable system for measurement of thoracic sounds with a microphone array," in *2017 IEEE SENSORS*. IEEE, 2017, pp. 1–3.

[2] A. Bertrand, "Distributed signal processing for wireless EEG sensor networks," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 23, no. 6, pp. 923–935, 2015.

[3] D. Seo, J. M. Carmena, J. M. Rabaey, E. Alon, and M. M. Maharbiz, "Neural dust: An ultrasonic, low power solution for chronic brain-machine interfaces," *arXiv preprint arXiv:1307.2196*, 2013.

[4] J. Lee, V. Leung, A.-H. Lee, J. Huang, P. Asbeck, P. P. Mercier, S. Shellhammer, L. Larson, F. Laiwalla, and A. Nurmikko, "Neural recording and stimulation using wireless networks of microimplants," *Nature Electronics*, vol. 4, no. 8, pp. 604–614, 2021.

[5] E. Reusens, W. Joseph, B. Latré, B. Braem, G. Vermeeren, E. Tanghe, L. Martens, I. Moerman, and C. Blondia, "Characterization of on-body communication channel and energy efficient topology design for wireless body area networks," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 6, pp. 933–945, 2009.

[6] F. Chen, A. P. Chandrakasan, and V. M. Stojanovic, "Design and analysis of a hardware-efficient compressed sensing architecture for data compression in wireless sensors," *IEEE Journal of Solid-State Circuits*, vol. 47, no. 3, pp. 744–756, 2012.

[7] B. Somers and A. Bertrand, "Removal of eye blink artifacts in wireless eeg sensor networks using reduced-bandwidth canonical correlation analysis," *Journal of neural engineering*, vol. 13, no. 6, p. 066008, 2016.

[8] O. De Wel, M. Lavanga, A. C. Dorado, K. Jansen, A. Dereymaeker, G. Naulaers, and S. Van Huffel, "Complexity analysis of neonatal EEG using multiscale entropy: applications in brain maturation and sleep stage classification," *Entropy*, vol. 19, no. 10, p. 516, 2017.

[9] A. H. Ansari, P. J. Cherian, A. Caicedo, G. Naulaers, M. De Vos, and S. Van Huffel, "Neonatal seizure detection using deep convolutional neural networks," *International journal of neural systems*, vol. 29, no. 04, p. 1850011, 2019.

[10] J. T. Giacino, J. J. Fins, S. Laureys, and N. D. Schiff, "Disorders of consciousness after acquired brain injury: the state of the science," *Nature Reviews Neurology*, vol. 10, no. 2, p. 99, 2014.

[11] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "EEGNet: a compact convolutional neural network for EEG-based brain–computer interfaces," *Journal of neural engineering*, vol. 15, no. 5, p. 056013, 2018.

[12] B. Z. Allison, E. W. Wolpaw, and J. R. Wolpaw, "Brain–computer interface systems: progress and prospects," *Expert review of medical devices*, vol. 4, no. 4, pp. 463–474, 2007.

[13] F. Lotte, L. Bougrain, A. Cichocki, M. Clerc, M. Congedo, A. Rakotomamonjy, and F. Yger, "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.

[14] A. J. Casson, D. C. Yates, S. J. Smith, J. S. Duncan, and E. Rodriguez-Villegas, "Wearable electroencephalography," *IEEE engineering in medicine and biology magazine*, vol. 29, no. 3, pp. 44–56, 2010.

[15] B. Mirkovic, M. G. Bleichner, M. De Vos, and S. Debener, "Target speaker detection with concealed eeg around the ear," *Frontiers in neuroscience*, vol. 10, p. 349, 2016.

[16] K. B. Mikkelsen, S. L. Kappel, D. P. Mandic, and P. Kidmose, "Eeg recorded from the ear: characterizing the ear-eeg method," *Frontiers in neuroscience*, vol. 9, p. 438, 2015.

[17] M. G. Bleichner, B. Mirkovic, and S. Debener, "Identifying auditory attention with ear-eeg: ceegrid versus high-density cap-eeg comparison," *Journal of neural engineering*, vol. 13, no. 6, p. 066004, 2016.

[18] M. Baijot, A. M. Narayanan, M. B. A. Rosa, J. Dan, A. Bertrand, and M. Kraft, "A miniature eeg node for synchronized wireless eeg sensor networks," 2021.

[19] T. Tang, L. Yan, J. H. Park, H. Wu, L. Zhang, H. Y. B. Lee, and J. Yoo, "34.6 EEG dust: A BCC-based wireless concurrent recording/transmitting concentric electrode," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 516–518.

[20] A. M. Narayanan and A. Bertrand, "Analysis of miniaturization effects and channel selection strategies for EEG sensor networks with application to auditory attention detection," *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 1, pp. 234–244, 2019.

[21] T. Strypsteen and A. Bertrand, "End-to-end learnable EEG channel selection for deep neural networks with gumbel-softmax," *Journal of Neural Engineering*, vol. 18, no. 4, p. 0460a9, 2021.

[22] Y. Roy, H. Banville, I. Albuquerque, A. Gramfort, T. H. Falk, and J. Faubert, "Deep learning-based electroencephalography analysis: a systematic review," *Journal of neural engineering*, vol. 16, no. 5, p. 051001, 2019.

[23] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[24] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker *et al.*, "Large scale distributed deep networks," 2012.

[25] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[26] K. Bhardwaj, C.-Y. Lin, A. Sartor, and R. Marculescu, "Memory-and communication-aware model compression for distributed deep learning inference on iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–22, 2019.

[27] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *International Conference on Embedded Computer Systems*. Springer, 2019, pp. 77–90.

[28] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[29] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball, "Deep learning with convolutional neural networks for EEG decoding and visualization," *Human brain mapping*, vol. 38, no. 11, pp. 5391–5420, 2017.

[30] H. Wu, F. Li, Y. Li, B. Fu, G. Shi, M. Dong, and Y. Niu, "A parallel multiscale filter bank convolutional neural networks for motor imagery EEG classification," *Frontiers in Neuroscience*, vol. 13, p. 1275, 2019.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[32] A. M. Narayanan, R. Zink, and A. Bertrand, "Eeg miniaturization limits for stimulus decoding with eeg sensor networks," *Journal of Neural Engineering*, vol. 18, no. 5, p. 056042, 2021.

[33] Y.-H. Liu, X. Huang, M. Vidojkovic, A. Ba, P. Harpe, G. Dolmans, and H. de Groot, "A 1.9 nj/b 2.4 ghz multistandard (bluetooth low energy/zigbee/ieee802. 15.6) transceiver for personal/body-area networks," in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*. IEEE, 2013, pp. 446–447.

[34] K. Guo, W. Li, K. Zhong, Z. Zhu, S. Zeng, S. Han, Y. Xie, P. Debacker, M. Verhelst, Y. Wang. "Neural Network Accelerator Comparison" [Online]. Available: https://nicsefc.ee.tsinghua.edu.cn/projects/neural-network-accelerator/.

[35] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

## APPENDIX A
## MSFBCNN ARCHITECTURE

| Layer | # Filters | Kernel | Stride | # Params | Output | Activation | Padding |
|---|---|---|---|---|---|---|---|
| Input | | | | | (C,T) | | |
| Reshape | | | | | $(1, T, C)$ | | |
| Timeconv1 | $F_T$ | $(64, 1)$ | $(1, 1)$ | $64F_T$ | $(F_T, T, C)$ | Linear | Same |
| Timeconv2 | $F_T$ | $(40, 1)$ | $(1, 1)$ | $40F_T$ | $(F_T, T, C)$ | Linear | Same |
| Timeconv3 | $F_T$ | $(26, 1)$ | $(1, 1)$ | $26F_T$ | $(F_T, T, C)$ | Linear | Same |
| Timeconv4 | $F_T$ | $(16, 1)$ | $(1, 1)$ | $16F_T$ | $(F_T, T, C)$ | Linear | Same |
| Concatenate | | | | | $(4F_T, T, C)$ | | |
| BatchNorm | | | | $2F_T$ | $(4F_T, T, C)$ | | |
| Spatialconv | $F_S$ | $(1, C)$ | $(1, 1)$ | $4CF_TF_S$ | $(F_S, T, 1)$ | Linear | Valid |
| BatchNorm | | | | $2F_S$ | $(F_S, T, 1)$ | | |
| Non-linear | | | | | $(F_S, T, 1)$ | Square | |
| AveragePool | | $(75, 1)$ | $(15, 1)$ | | $(F_S, T/15, 1)$ | | Valid |
| Non-linear | | | | | $(F_S, T/15, 1)$ | Log | |
| Dropout | | | | | $(F_S, T/15, 1)$ | | |
| Dense | $N_C$ | $(T/15, 1)$ | $(1, 1)$ | $F_S(T/15)N_C$ | $N_C$ | Linear | Valid |

TABLE II: Architecture of the MSFBCNN used for motor execution classification (the 'Classifier' blocks in Figure 1). In the model we use $T = 1125$, $F_T = 10$, $F_S = 10$ and $N_C = 4$. Each node operates a single-channel version version of this network where $C = 1$ for the *ClassFuse* and the fusion center contains a multi-channel version for the *CompressFuse*, where $C$ is the number of nodes. This table is cited from [30].