Mazen Bouchur and Andreas Reinhardt. "Efficient Neural Network Representations for Energy Data Analytics on Embedded Systems." In: *Proceedings of the 13th ACM International Conference on Future Energy Systems (e-Energy)*. 2022, pp. 81–92.

Efficient Neural Network Representations for Energy Data Analytics on Embedded Systems

Mazen Bouchur
TU Clausthal
Germany
mazen.bouchur@tu-clausthal.de

Andreas Reinhardt
TU Clausthal
Germany
reinhardt@ieee.org

ABSTRACT

Electrical energy consumption data contain a great wealth of information. Currently, however, only limited insights are possible into the data that are collected by millions of smart meters everyday. A major reason for this limitation is the high computational and memory demand of corresponding analysis methods. This makes their execution on embedded processing systems virtually impossible. For example, Non-Intrusive Load Monitoring (NILM) methods strive to extract the individual power demand of an appliance from the aggregated power data of a household. Recent NILM solutions are based on highly memory-intensive neural networks. This limits their application to powerful hardware systems and strongly hampers their wide practical adoption. In this work, we demonstrate how the application of neural network model compression techniques can be applied to make state-of-the-art NILM solutions operable on systems with limited resources. Through comparatively analyzing the impact of size reduction techniques on the models, we show how a balance between neural network size and NILM accuracy can be found. We moreover verify the operability on a real embedded hardware system by means of a practical evaluation. Model compression enables NILM methods to be executed on a vastly greater number of devices, at similar levels of performance as compared to unmodified algorithms.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; • **Computing methodologies → Neural networks**.

KEYWORDS

Non-Intrusive Load Monitoring, Neural networks, Edge computing

ACM Reference Format:
Mazen Bouchur and Andreas Reinhardt. 2022. Efficient Neural Network Representations for Energy Data Analytics on Embedded Systems. In *The Thirteenth ACM International Conference on Future Energy Systems (e-Energy '22), June 28-July 1, 2022, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3538637.3538842

1 INTRODUCTION

Non-Intrusive Load Monitoring follows the objective to estimate the energy consumption of individual household appliances, based only on the total energy demand of a household. This disaggregation process is a major driver to exploit the wealth of information contained in smart meter data, and provides benefits to energy producers and consumers alike. Utility companies can benefit from a better understanding of their consumers' habits. For instance, they can categorize their customers by their consumption patterns (i.e., their daily routines) and exploit this knowledge to adapt their generation strategies. On the consumer side, NILM can reveal energy consumption habits and the obtained data can be used to generate personal recommendations [2]. This feedback can take the form of a direct report of current energy usage of each appliance, but can also be distilled further to provide services like anomaly and failure detection, thus avoiding great costs from faulty appliances.

NILM refers to a family of solutions for the load disaggregation problem. The common objective is to analyze a temporal sequence of readings (the *input window*) of the total energy consumption, in order to identify the contributions of individual appliances. Since its original description in [20, 21], multiple NILM algorithms have been proposed, including approaches based on Hidden Markov models [43] as well as Machine Learning-based approaches like Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs) and Denoising Autoencoders (DAEs) [31, 35, 41, 49]. The performance of these algorithms is highly variable and depends on multiple factors, such as the quality of the available data [24, 45], their sampling frequency [25], and the intended purposes [12, 30]. Early NILM methods focused on detecting the on-off states of each appliance, without predicting their exact power consumption. Today, the focus has shifted towards disaggregating the total power consumption data of a dwelling into exact predictions of each appliance's power demand. Good success has been reported for the usage of methods based on deep neural networks [24, 31, 51]. By way of example, Fig. 1 shows the actual and disaggregated (predicted) power consumption of four electrical appliances.

As a result of being based on deep neural models, state-of-the-art NILM methods exhibit a growing need for memory and computational power. This hinders their wide application on the customers' premises, e.g., by executing them on smart meters. Local data processing and on-device inference, however, has several advantages, such as eliminating the need for transferring all readings to a central processing system, thus inherently preserving user privacy better. However, moving inference to embedded systems based on Microcontroller Units (MCUs) requires a substantial reduction in the memory footprint and computational requirements of such models. This can be achieved through a compression of the neural network

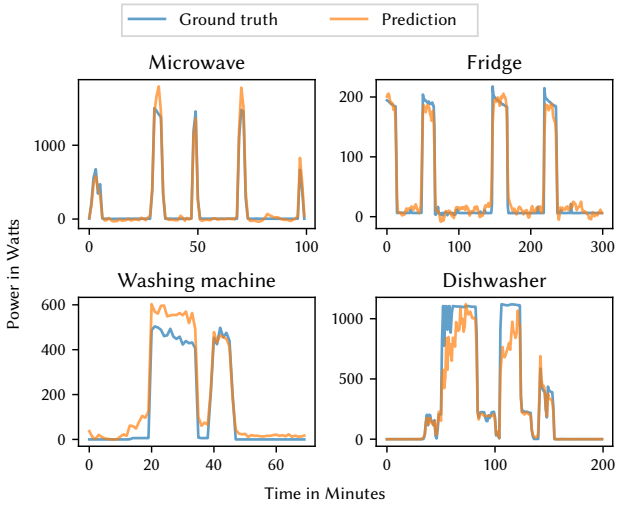

Figure 1: Comparison of the actual and disaggregated power consumptions of four different electrical appliances.

models. We hence comparatively study six different techniques to reduce the size and computational requirements of NILM models and verify their operability on embedded devices.

We survey previous approaches to compress neural network models for energy disaggregation in Sec. 2. Subsequently, we introduce the Sequence-to-Point (S2P) model for NILM in Sec. 3, which will serve as the basis for our experiments. Through an analysis of S2P's internal structure, we identify suitable locations to apply model compression. In Sec. 4, we present the considered compression methods along with the modifications to the S2P model that are required to apply them in practice. Our evaluation setup is described in Sec. 5, and results for the size and accuracy of each compression technique are presented in Sec. 6. We distill the results of each experiment into a new model that combines the best configurations in each compression method and test it on a microcontroller system based on the ESP32 MCU in Sec. 7, before summarizing our findings in Sec. 8.

2 RELATED WORK

The tremendous success of Deep Neural Networks (DNNs) has motivated researchers to apply these methods to the analysis of energy data [17]. Kelly and Knottenbelt were among the first researchers to propose and evaluate the use of DNN architectures in the context of NILM [31], especially focusing on models that rely on convolutional networks. Zhang et al. improved the performance of previous models by introducing the Sequence-to-Point (S2P) architecture for NILM in [51]. Thanks to their remarkable performance, sequence-based models enjoy a great popularity for NILM. At the same time, however, the underlying DNNs are notorious for their high computational requirements, which limit their operability on resource-constrained systems. Consequently, several compression techniques have emerged in recent years to achieve a trade-off between processing efficiency, memory demand, and achievable accuracy [14]. As follows, we review previous studies on compressing convolutional networks in the context of energy data disaggregation.

In [5], the effect of pruning techniques, i.e., selectively disabling and removing parts of the neural network, has been tested on the S2P model. The authors show that a simple low-magnitude pruning does not only lead to a reduction of the neural network's size, but also improves its performance at the same time. Two more approaches to apply magnitude-based pruning for S2P were also examined in [36], differing in whether the pruning step is applied once or repeatedly.

Quantization, i.e., the reduction of the resolution of numerical values in stored models, is yet another approach to reduce a model's size. In [1], the main elements from MobileNets [22] are carried over to NILM, targeting to assess its operability on resource-constrained edge devices. After some modifications to the original MobileNet model, including a change of the activation function, the authors assessed the impact of lowering the precision of the learnable model parameters by means of quantization. Measurable size reductions could be accomplished across three different NILM datasets, yet also a 10 % degradation in accuracy was incurred [1]. It is a general observation that deploying computationally demanding NILM models on devices with computational and memory constraints is a challenging task, especially when real-time operation is desired [3, 4].

Brewitt and Goddard present a network architecture based only on convolutional layers, known as a Fully Convolutional Network (FCN), for load disaggregation [8]. The authors' rationale for their design is that the majority of the parameters in a typical Convolutional Neural Network (CNN) structure are concentrated in the dense part of the network. Removing this part of the network eliminates a large fraction of the trainable parameters, thus reducing the model size and increasing inference speed. However, to make up for the lost dense part in FCNs, an increase in the density of the input field of the network is required. The authors of [8] employed the dilated convolution concept from [42, 48], which enlarges the receptive fields exponentially with the number of layers. They showed that their FCN architecture achieves better accuracy and faster training times with an around 72 × smaller model size than S2P.

The application of Multi-task Learning (MTL) is another idea for reducing the resource demand that was introduced in [36]. While in the large majority of NILM studies, individual models are being trained for each electrical appliance whose operation shall be identified in aggregate load data, MTL applies the concept of transfer learning to eliminate this requirement [9]. In transfer learning, generic models are being trained (allowing them to identify appliances of a large range of models and types) rather than focusing on the creating highly optimized models for each device separately. The application of transfer learning for S2P was reviewed in [15], showing that the many features in the neural network are in fact invariant to the selected appliances and datasets. It needs to be noted, however, that MTL only leads to size savings when the operation of multiple appliances shall be detected in aggregated data. The prevalent use case in literature, where the power demand of only one appliance is being disaggreated, does not benefit from MTL.

Our survey of related work has shown that several approaches have been made to reduce the model size, some of which can even be

Table 1: The S2P model architecture as described in [51].

Part	Layer		Total number of
	Type	Configuration	trainable parameters
	Input	Window of 599 readings	
Convolutional	Conv1D	30 filters of size 10	330
	Conv1D	30 filters of size 8	7,230
	Conv1D	40 filters of size 6	7,240
	Conv1D	50 filters of size 5	10,050
	Dropout	With 20 % rate	
	Conv1D	50 filters of size 5	12,550
	Dropout	With 20 % rate	
	Flatten		
Dense	Dense	1024 units	30,669,824
	Dropout	With 20 % rate	
	Dense	A single output unit	1,205

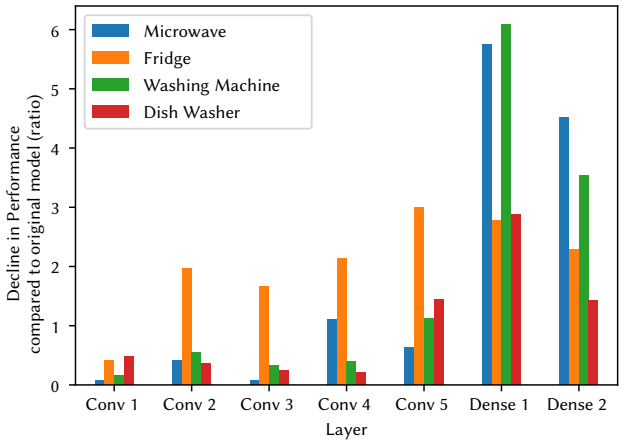

Figure 2: Layer-wise decline in performance as measured by MAE for baseline S2P models trained for different appliances. The decline is measured based on the re-randomization robustness and normalized by the number of parameters in the corresponding layer.

combined with each other. For a direct comparability between the aforementioned methods, however, a coherent evaluation methodology is needed. Moreover, the operability on embedded systems has not been within the focus of most existing works, despite the advent of edge computing and the resulting trend towards analyzing data locally on smart meters. Rather than reviewing a single compression method only, we thus comparatively study the applicability of six methods to reduce the computational and size requirements of NILM neural networks, with custom modifications and a specific focus on the possibility to execute them on resource-contrained systems.

3 NILM BASELINE MODEL

To cater for the comparability of our results, we base our work on the S2P architecture presented in [51], which is one of the most successful deep learning-based models to solve the NILM problem. This model consistently ranks among the top-performing NILM methods and is consequently part of many evaluation studies [24].

3.1 Model Architecture

Motivated by the development of sequence-based models in the areas of speech recognition and image classification, the S2P architecture relies on an input data window of fixed length, and outputs the disaggregated power demand of the appliance under consideration for a single point in the middle of this window [51]. The task of the model is therefore to reduce the input window to a single prediction. It benefits from information that spans the history as well as the future in the current input window, which is assumed to have a high correlation with the mid-point of the current appliance usage. The model is based on a one-dimensional convolutional network architecture with fully-connected layers at the model output. By design, the model is configured to learn and disaggregate traces of a single appliance, hence a new model must be trained for each target appliance.

The structure of the model is shown in Table 1, where the input window is applied at the top and the output sample is returned by the layer on the bottom of the table. The S2P architecture is composed of five one dimensional convolutional layers with increasing numbers of hidden neurons and simultaneously decreasing filter sizes. These choices are based on typical theoretical bases for CNNs and sequence data, which intend to learn patterns in the input at different positions and make them recognizable. The last convolutional layer is, after a 20 % dropout, followed by a flatten layer which prepares the data to be provided to the first dense fully connected layer with 1024 neurons. This transformation from convoluted to flattened data is responsible for the largest number of parameters in the network, which makes it an interesting and relevant target for optimizing the network size, as we will show later in Secs. 4.2, 4.3 and 4.5. Finally, the last dense layer with its singular neuron produces the final model output. Three dropout layers are present in the lower half of the neural network to reduce the risks of overfitting as the number of network parameters increases.

3.2 Layer Importance Assessment

Five different types of parameters can be identified in any neural network, and categorized into two groups based on their learnability: (1) Static and/or learned parameters, i.e., weights and biases, and (2) Variable parameters, i.e., activations, inputs, and outputs. While the values of static parameters stay constant after training, the variable parameters realize the final prediction of the model during the inference phase. Both types of parameters can be valid targets for compression. Especially the reduction of the number of static parameters leads to reductions in the final trained model size and an increased inference speed. Beside the number and type of parameters in each layer, the layer location plays an important role in the final model performance of most neural architectures [50]. The achievable compression ratio thus strongly depends on the location of the modified layer within the neural network.

We have thus estimated the importance of each layer following the procedure documented in [50], i.e., by resetting or randomizing the weights of a single layer and considering the decline in performance of the overall model. Fig. 2 provides results for this estimation for the used S2P model, which has been individually trained to recognize one of four appliance types (microwave oven, refrigerator, washing machine, dish washer). While the last convolutional layer has a greater impact on the overall performance than previous layers, the dense layers clearly have the greatest effect on the final model performance with more than 99 % of the parameters. This difference is due to their type and location: Each neuron in the first dense layer is connected to all neurons in the last convolutional layer. The impression that alterations of the dense layers lead to the greatest decline in performance needs to be put into perspective, however. On the one hand, the first dense layer has the largest number of trainable parameters by far. On the other hand, the convolutional layers cannot be considered in isolation, but will generally compensate for changes applied to only one of the layers. The results of our preliminary study have demonstrated that accuracy reductions have the greatest impact in the layers buried deep within the neural network, while the first layers are comparably insensitive to variations. We exploit this knowledge in our selection of suitable locations for applying compression techniques.

4 NEURAL NETWORK COMPRESSION METHODS

Six candidate methods for compressing S2P models are introduced in this section. Fig. 3 shows a simplified overview of the general mode of operation of these methods, as well as pointing to the section where the corresponding method is described in more detail. In real models, methods can also be applied to a subset of the model only. Moreover, a combination of multiple methods can also be used in tandem. As a general observation, however, the purpose of each method is to either reduce the number the model parameters or the amount of memory required for their storage.

4.1 Quantization

Quantization is the process of mapping an input value from larger (or more precise) value range into smaller one, which necessarily entails a loss of precision. In the context of neural networks, quantization refers to the reduction of the resolution of parameters (e.g., weights) by lowering the number of bits required to represent them (bitwidth). For instance, moving the realm of arithmetic operations from the floating point space into integer only arithmetic through quantization allows for the use of embedded hardware devices without Floating-Point Units (FPUs). Thus, quantization can improve both memory size and computational efficiency at the cost of accuracy.

Quantization can be applied selectively to any of the network parameters. Subsequently, different quantization approaches for neural networks have emerged, as shown in [18, 28, 37]. Based on recommendations from [28, 34], we examine the following ways of applying quantization in the model:

Weights Quantization (`weights8`*).* All weights and biases of the model are quantized into 8-bit integers, while preserving other parameters at their original precision. Activations are quantized into 8-bit values on-the-fly during inference to perform the required matrix multiplication operations with the weights and biases. This mode promises up to 4 × reduction in model size if the original parameters have been stored as 32-bit floating-point numbers.

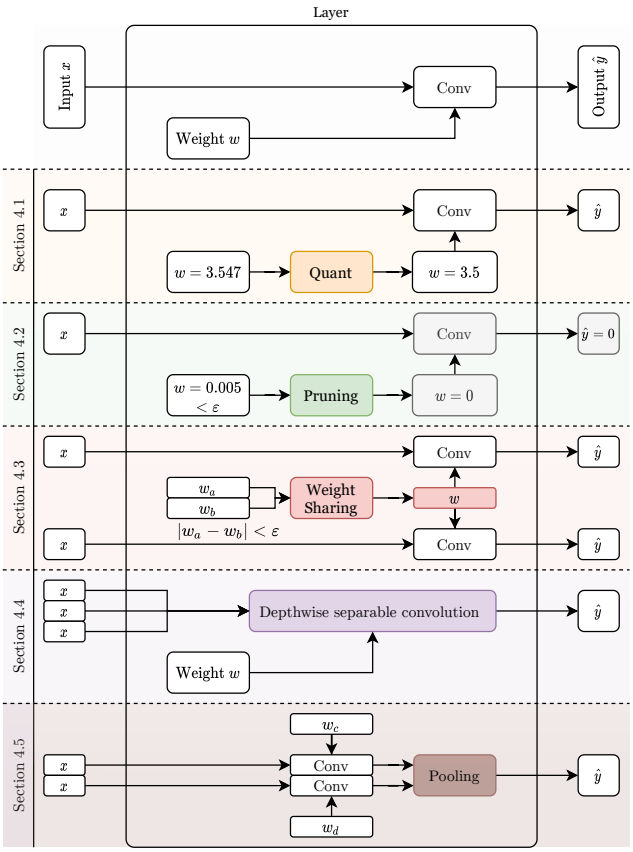

Figure 3: Simplified overview of the compression methods as operators on the network weights, where x is a single input sample, $\hat{y}$ is the corresponding model prediction, w_i is the weight value of the i-th neuron in the layer, and ε is a small positive threshold value that must be chosen appropriately depending on the corresponding method.

Mixed Precision Quantization (`mixed8` *and* `mixed16`*).* While the most commonly used data types are `float32` (32-bit floating point values) and `int8` (8-bit integer values), other options can also be considered. If only a minimal reduction in size is required, weights can also be quantized to the half-precision floating-point format (`float16`) while keeping all other parameters at their original precision. This reduces the model size almost by half. One may also apply different precision degrees for each type of parameters based on their importance for the inference (biases being more important than activations, which in turn are more important than weights [28]). Thus, a scheme for integer quantization that compresses biases and activations with slightly higher precision while still achieving full integer quantization, can have large benefits and avoid any floating-point arithmetic to achieve acceptable accuracy with high

computational efficiency. Two options have been examined for this mode: Quantization of weights from `float32` to `float16` (referred to as `mixed16`) and quantization of activations to `int16`, weights to `int8`, and biases to `int64` (referred to as `mixed8`).

Full Quantization (`full8`). As the name suggests, all network parameters (including inputs, activations, and outputs) are quantized post-training to `int8` values in this mode. Since this mode also includes the variable parameters, a priori knowledge of the dataset characteristics through representative samples is essential to calibrate the quantization processes and prepare it for the type of incoming data.

4.2 Pruning

Pruning essentially comprises of two steps: An evaluation of parameter importance, followed by a subsequent elimination of unimportant parameters. The first step typically results in a threshold value that allows to distinguish the model values by importance in the second step. As the weights generally contribute most to a model's final memory demand, they are the typical target for pruning. Since the largest weights have the greatest impact on a neural network's operation [19], it is a natural choice to estimate the importance of a weight by its magnitude. The default pruning strategy only masks the network weights (without actually removing the corresponding cells from the neural network), thus it does not lead to an automatic reduction in model size. For a fair comparison of the reduced memory demand of pruned neural networks, we thus consider both regular pruning, in which weights of pruned elements are set to zero, as well as structural pruning, in which pruned elements are completely removed from the neural network.

Pruning can be applied as a single step after training the network by removing weights below the pruning threshold. Alternatively, iterative pruning consists of multiple iterations of retraining, each of which is followed by a partial pruning step with increased value of pruning threshold, until the required target sparsity is reached [19]. This approach requires a further configuration parameter (beside the target sparsity) which defines the *frequency* at which pruning is applied, in terms of the number of training steps. Since the weights of the neural network are often initialized with random values, it is generally preferred to train the network first, such that it can learn the initial importance of its parameters before starting to apply pruning. Similarly, neural networks generally need some training iterations to adapt themselves to the new structure that results from pruned weights. Thus, we apply pruning only for a fraction of the training phase, configurable by its *start* and *end* training iteration. It is also possible to distribute the pruning effect unevenly between the start and stop steps by applying a non-linear pruning scheme as a function of the current training step, which increases the strength of pruning by changing the threshold in each step in non-linear fashion that depends on a *power* parameter [53]. We have confined our study the following set of settings due to their popularity in the literature [7]:

`const`. Constant sparsity scheme with target sparsity of 50 % that starts at 20 % of the total number of training steps and ends at 80 %, applied twice per epoch.

`poly`. Polynomial sparsity scheme with the same parameters as the constant sparsity scheme, but using a power of 2.

`s-const`. Structurally pruned variant of the `const` pruning scheme.

`s-poly`. Structurally pruned variant of the `poly` pruning scheme.

4.3 Weight Sharing

Weight sharing (also known as weight clustering) aims to group weights together based on a metric of similarity. A clear advantage of weight sharing is to reduce the total number of unique weights in the model, and thus the model size. However, if the required operations for inference support custom mode for clustered data, then an acceleration of the inference speed is also possible [18]. The objective of the similarity metric is to quantify the amount of association of each weight to each group. To achieve this, a clustering algorithm can be used to identify similar weights. A certain number of centroids (clusters' centers) must be chosen in advance (as a hyperparameter), which is also the number of the weight groups. Weights in the neural network are then grouped by their distance to these centroids. After grouping, each weight keeps a reference to its centroid instead of its original value. The idea was originally proposed in [40] and further employed in other studies, such as [18] and [11].

The grouping of the weights can be done across the whole network; however, weights tend to have different value ranges across different layer types. Thus, it is more appropriate to group weights by layers. This also opens the opportunity to apply weight sharing only to some layers of the network. As shown in Sec. 3.2, different layers within the S2P model have a different impact on the total model performance. For this reason, applying weights sharing selectively to the dense part of the network (due to its higher parameters' density) can achieve a high compression rate without sacrificing accuracy. Based on the centroid selection method and the layers to which weight sharing is applied, we consider the following models in this work:

`all/density` *and* `dense/density`. Centroids are chosen such that weights are distributed evenly among the groups (density-based) in either the full network (`all/density`) or only in the two dense layers (`dense/density`).

`all/linear` *and* `dense/linear`. Centroids are chosen uniformly between the minimum and maximum weights values in each layer in the full network (`all/linear`) or only in the dense layers (`dense/linear`).

`all/random` *and* `dense/random`. Centroids are chosen randomly, by sampling the required number of groups from the weights' values in the full network (`all/random`) or only in the dense layers (`dense/random`).

`all/kmeans` *and* `dense/kmeans`. Centroids are chosen by the k-means++ algorithm, which is used to find initial clusterings for the k-means algorithm and provides a degree of optimality to avoid arbitrary poor clustering in the full network (`all/kmeans`) or only in the dense layers (`dense/kmeans`).

4.4 Depthwise Separable Convolution

All of the compression techniques discussed above are targeting to optimize neural networks with a given, fixed architecture. However, the architecture of a neural network has a direct effect on its performance for a given application. Many attempts with varying degrees of success have been made trying to bring large successful networks to low-end devices by altering and especially reducing their architecture, e.g., [13, 22, 23, 26, 52] (resulting in so called micro-networks). One successful micro-network has been presented in [22]. Its core idea is to separate the convolution operations that occur in the CNN layers into two components. This approach, known as depthwise separable convolution, divides the convolution operation occurring in convolutional layers into depth- and pointwise convolution operations. This, however, requires the used kernels to be separable[1]. This is not the case for most convolutional layers within CNNs. While the first component (depthwise) operation performs a standard convolution and can be separated, a separation is not possible across the last dimension of the convolution matrix (i.e. the channels) and thus a single filter must be applied per channel. Because this results in a loss of the combined convolution across these channels, a second pointwise component is required to combine the outputs of the depthwise convolution again. Depthwise separable convolution has been already deployed in many models such as Flattened networks [29], Xception networks [13], and Inception models [27].

The number of parameters in S2P models is often prohibitively large for embedded applications. Single-kernel depthwise convolution offers a new structural aspect to reduce the model architecture rather than compressing it. In this work, we propose to integrate the success ingredient from MobileNets [46] and other architectures of micro-networks with the successful S2P model by replacing the convolutional layers in the initial part of the network with depthwise separable convolution operations.

`dwsc`. We integrate a separation of the convolution step into the S2P model by applying a single-kernel depthwise convolution in the convolutional part of the network, which greatly reduces the number of parameters. Unlike [1], however, we adapt this concept for the S2P model instead of replicating the MobileNets network architecture for NILM.

4.5 Pooling

Pooling is an established approach to reduce the dimensionality of convolutional layers output to improve computational and memory efficiency while simultaneously lowering the overfitting risks. Global average pooling in a CNN as proposed in [38] averages the output of the final convolutional layer across each feature map before feeding into the dense part of the network. Pooling thus reduces the model size and overfitting risks simultaneously.

As the convolutional-dense transition in the baseline model results in 99 % of the model parameters (cf. Table 1), applying pooling to this part of the network can greatly reduce the number of parameters and model size. Furthermore, different appliances exhibit diverse recurring patterns with variable lengths that can be exploited at the convoluted part of the network. Motivated by this fact, we explore a new pooling approach by decomposing the average pooling into smaller chunks, thus averaging each fractional length of the feature maps (i.e., the filters) labeled by their dimensions:

`p-`x`,`y. Represents the new models with the configurable pooling layer. The last axes in the final convolution layer is expanded to 2 new axes with sizes of x and y, and as before the averaging is only applied to the last axis, which includes y units. Thus, the output of the flatten layer is of size y.

4.6 Multitask Learning

Multi-task Learning (MTL) is an application of transfer learning, which has been successfully applied and tested on CNNs in multiple domains [10, 47]. It exploits the fact that many tasks in a single domain share the same fundamental properties. The complexity of these properties is reflected in the order of layers of a neural network: Layers closer to the output are generally capable of capturing and representing more complex concepts. Instead of training a new network to disaggregate the power traces for each appliance separately, the same network is used for all but the final layer, and different final (prediction) layers are being provided for the NILM-based disaggregation of different electrical appliances. Thus, the initial (shared) layers of the neural network can capture common features like the different activation cycles of each appliance or simple on/off state transitions. Conversely, the output of this structure is realized by different output layers, one for each appliance.

Results reported on its usability within the scope of NILM [36] have encouraged us to further explore this idea for S2P models. Especially that our intended application domain in memory-restricted systems can greatly benefit for reducing the initial part of multiple models (one for each appliance) into single shared one. We run our experiments on two S2P-based MTL models:

`mtl`. The structure of this model diverges after the first dense layer into a new output layer for each appliance.

`mtl/dwsc`. This model uses the same structure, but combines it with depthwise separable convolution for further size reduction.

5 EVALUATION SETUP

5.1 Input Data

The quality of models correlates directly with the quality of the data, as the models' parameters are learned from the data. Thus, it is important to make informed choices regarding the input data, to cater to a fair and representative evaluation. For our tests, we selected three publicly available datasets (UKDALE [32], REFIT [39], and REDD [33]) that provide data for the total power consumption as well as for individual appliances as the ground truth. They cover an adequate time period to capture periodicities and patterns in appliances operation cycles and human behavior. The conscious decision to include different datasets was made to ensure the transferability and validity of the learned models across different datasets captured in different countries, therefore, including different kinds of appliances and consumer behavior. We expect this

[1] A separable kernel is characterized by the fact that applying single one-dimensional computations in each direction achieve the same effect as applying the transformation to the whole kernel at once, usually in the form of a matrix multiplication.

to improve the resilience of the models and provide more real-live settings for evaluation, as models employed in practice have no prior knowledge about occupants' habits or locations.

Four target appliances that provide different electrical power consumption patterns have been selected for analysis. In general, two kinds of power patterns can be identified: simple and quick on/off state switches and more complex patterns (such as the diverse cycles of washing machines and dishwashers) [24]. We included both types to ensure the possibility to generalize the findings of the research. Furthermore, we favored appliances with high load demand, as the effect of appliances with negligible load on the aggregate power demand tends to vanish in the noise [31] and moreover promise less energy savings in domestic settings. Based on these criteria, the following appliances were selected:

(1) Microwave
(2) Fridge
(3) Washing machine
(4) Dish washer

All houses in UKDALE [32] and REDD [33] that contain the selected appliances were used for training and the houses that contain these appliances from REFIT [39] were used for cross-dataset testing.

A preprocessing step was applied to all datasets. This step ensures that the raw measurement are evenly spaced chronologically by applying resampling, forward-filling, and clipping techniques to remove any jitter in the timestamps and ensure that the data is synchronized and aligned consistently.

5.2 Evaluation Metrics

Given that only few of the compression techniques introduced in Sec. 4 have been evaluated in conjunction with energy data, and no comparative study of all methods using identical input data are known in the literature, it is important to identify the relevant metrics for a fair evaluation of the results presented in Sec. 6.

Two types of metrics capture the aspects of interest regarding our research. First, the performance metrics that quantify the accuracy of the model, i.e. how much the model prediction differs from the ground truth readings. Second, we quantify the compression gain, as reducing the model size to fit on resource-constrained embedded hardware is one of our main goals. For the first category, the MAE metric was chosen due to its usage in virtually all previous literature on sequence models in NILM [31, 36, 51] and for comparability and benchmarking against the baseline model. The MAE is computed as follows:

$$MAE = \sum_{i=0}^{N} \frac{|y_i - \widehat{y_i}|}{N}$$

where N is the number of samples, y_i is the i-th measured reading of the appliance (ground truth), and $\widehat{y_i}$ is the model prediction for y_i. As the network input is normalized, the output of the network is denormalized accordingly, using the same parameters that have been used during normalization, before computing the loss.

The second type of metrics attempts to measure the effectiveness and gains of the applied model compression techniques. A clear choice is the trained *model size*, as it also gives a good indication of the expected network performance and correlates with the number of model parameters.

Finally, some compression operations (e.g., unstructured pruning) do not affect the model size, as they only set specific network parameters to zero (effectively performing parameters masking) without actually removing them from the model. We thus also consider a model's *sparsity* by defining a near-zero threshold ε and reporting the number of parameters in the network that are less than ε over the total number of parameters. Thus, identical models in terms of size and the number of parameters may have different sparsity levels and be more amenable to pruning and other compression techniques. For the reported results, a value of 10^{-6} was used for ε. This metric is mentioned in the results when its value was of interest and relevant to the corresponding method.

Table 2: An overview of the parameters utilized in our experiments.

Parameter	Value
Batch size	1000 samples
Number of epochs	100 epochs
Resampling Rate	1 min
Validation split	20 % of the training data
Training houses from UKDALE	{1, 2, 5}
Training houses from REDD	{1, 2, 3, 5}
Testing houses from REFIT	{11, 18}

5.3 Model Parameters

In this section, we give a brief overview of further hyperparameters that are not directly related to the compression methods but required to configure the model. The input layer in S2P accepts a sliding window of the mains power data. Since the output is always restricted to a single point, distilling a large amount of information at one time may lead to worse performance. On the other hand, the ideal window size has been shown to be appliance-specific [44]. In order to retrieve comparable results, the default window size as proposed in [51] and used in further works (such as [6]) of 599 samples has been used across all experiments. Input data supplied to the model has moreover been normalized to avoid excessively high weight parameters during training and allow the model to converge, as the actual magnitude of the data is irrelevant for the learning process. The normalization parameters were computed from the entire set of training data, and consequently applied to all input windows. Table 2 provides an overview of other hyperparameters and the values used in the experiments.

5.4 Models with Constant Output

In order to develop a better understanding of the limits of model performance and identify poor models, we include the results of three trivial models: `zero`, `min`, and `avg`. These models do not perform an actual disaggregation operation, but always predict a constant value regardless of the input. As the name suggests, the first model always predicts 0 W. For the other models, the training data is used to compute the minimum and average power consumption values, respectively, and then these values are used for their constant value

predictions. Although these models are not emitting any meaningful disaggregated data, they serve as an indication of the MAE when no actual disaggregation takes place, i.e., they provide a baseline for understanding MAE values better. If MAE values equal to or greater than those of the constant output models are observed, this is indicative of NILM no longer functioning because the applied compression steps have removed crucial parts of the neural network.

6 RESULTS

The results of applying each of the compression methods described in Sec. 4 and Sec. 5 are shown in Table 3. Moreover, the table shows results for a combined model which is compiled from the other results and introduced in Sec. 6.7.

6.1 Quantization

Applying `mixed16` quantization, a reduction of the original baseline model size of 368.56 MB to 61.42 MB is possible. When using any of the other three quantization methods, a further size reduction to 30.71 MB is achieved. The results in Table 3 show a minor degradation in performance compared to the `baseline` model for all models except for `full8`, which results in significantly higher MAE values that partially even exceed the constant baseline models. This demonstrates that the quantization operation is too excessive for this model and hinders learning, so the predictions resemble random shapes rather than appliance activations and have higher MAE values than the `zero` model. In contrast to this, the `weights8` and `mixed8` models achieve better results with the same model size. This results is aligned with our previous discussions of the importance of selective quantization based on the parameter types in Sec. 3.2 and Sec. 4.1. Based on these results, we can conclude that 8 bits are not enough to pass learned features through the these parameters, especially neuron activations as they pass the information between the layers during inference, and a higher precision is required. The results also confirm that the model weights should be the first choice for quantization, as they have the highest effect on the model size and the lowest impact on performance. For reference, Fig. 4 shows the prediction of the `mixed16` model against the baseline model for a single input window. It becomes apparent that the limited value precision acts as a regularizer and restricts the model to simpler disaggregated consumption shapes. Considering the best tradeoff between model size and MAE increase, `weights8` has emerged as the best choice.

6.2 Pruning

As a general observation, pruned models consistently achieve comparable or even better MAE results than the baseline model. This is an indicator of the fact that the baseline models are highly overparameterized, and pruning was able to exploit this to improve the model's accuracy [16]. The model trained with polynomial scheduler always achieved better or comparable results to the model trained with the constant pruning scheduler. Polynomial pruning may better correlate with the Learning Rate (LR) decay of the optimizer, which plays an important role for pruning [53]. The stripped down models, which labels are prepended with `s-`, have also comparable results to their full counterparts. The benefit of these models becomes clearer when we look at their sizes. `s-const` was able to reduce the size of `const` by 47 % and `s-poly` reduced the size of `poly` by 33 %, while maintaining an almost identical performance.

The training loss of the `const` models shows a clear drop after the *start* pruning step (cf. Sec. 4.2) which is a clear indicator of how

Table 3: Experiments result of model accuracy and size for appliances microwave (MW), fridge (F), washing machine (WM), dishwasher (DW), and the arithmetic mean of all four.

	Mean Average Error [W]					Size
	MW	F	WM	DW	Mean	[MB]
Baseline and constant-value models						
`baseline`	11.68	16.13	12.05	14.23	13.52	368.56
`zero`	27.68	74.33	112.42	21.80	59.06	0.0093
`min`	57.12	66.40	158.63	64.96	86.78	0.0093
`avg`	36.50	74.22	110.16	21.01	60.47	0.0093
Quantized models						
`weights8`	23.61	15.86	15.41	14.51	17.34	30.71
`full8`	15.02	75.17	186.19	34.85	77.81	30.71
`mixed16`	14.87	11.29	14.92	14.53	13.93	61.42
`mixed8`	15.01	48.15	15.81	14.96	23.48	30.71
Pruned models						
`const`	11.71	10.03	7.34	11.89	10.24	122.86
`s-const`	11.70	10.03	7.35	11.89	10.24	65.72
`poly`	9.86	9.61	6.65	11.94	9.51	122.86
`s-poly`	9.86	9.61	6.84	11.94	9.56	82.73
Models with shared weights						
`all/density`	11.28	18.66	29.54	14.19	18.41	117.18
`all/kmeans`	9.67	12.71	13.46	15.27	12.78	117.18
`all/linear`	9.72	12.94	11.67	15.51	12.46	117.18
`all/random`	10.16	12.20	12.95	14.92	12.56	117.18
`all average`	10.21	14.13	16.91	14.97	14.05	117.18
`dense/density`	11.30	18.31	14.62	13.08	14.32	122.87
`dense/kmeans`	10.29	12.80	9.90	14.64	11.91	122.87
`dense/linear`	10.21	12.88	13.01	14.16	12.56	122.87
`dense/random`	14.93	13.88	19.93	12.52	15.31	122.87
`dense average`	11.68	14.47	14.37	13.60	13.53	122.87
Model with depthwise separable convolution						
`dwsc`	12.46	18.10	15.26	15.72	15.38	1.01
Models with pooling layer						
`p-2,14975`	11.72	16.22	11.89	14.33	13.54	184.56
`p-5,5990`	14.79	19.48	11.12	13.39	14.67	74.15
`p-10,2995`	17.81	24.22	10.87	14.51	16.85	37.35
`p-25,1198`	21.56	33.32	12.73	14.87	20.62	15.27
`p-50,599`	22.80	13.71	14.44	15.34	22.07	7.91
Single model for all appliances (MTL)						
`mtl`	12.24	17.86	14.88	13.77	14.68	368.63
`mtl/dwsc`	16.99	18.13	18.36	14.66	17.03	3.37
Combined model (described in Sec. 6.7)						
`comb`	8.68	13.88	10.86	12.06	11.36	0.59

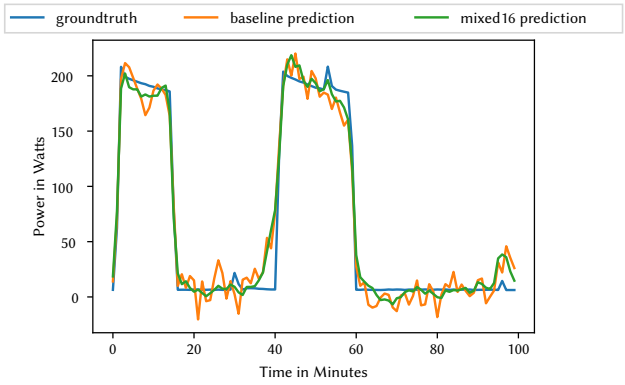

Figure 4: Predictions of `mixed16` and baseline models for a fridge trace.

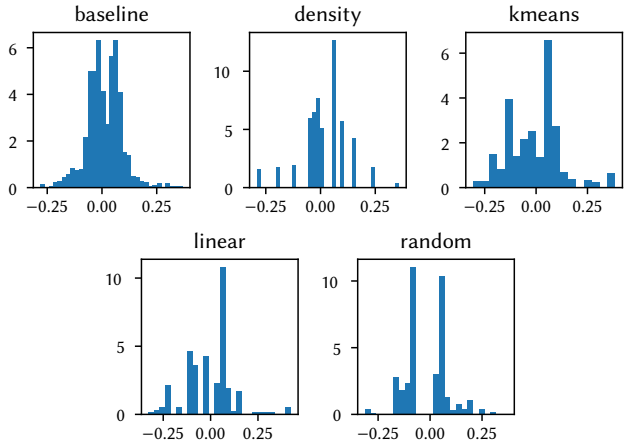

Figure 5: Weights distribution in the second dense layer of S2P models trained to disaggregate dishwasher traces with different weight sharing strategies, as described in Sec. 4.3 and Sec. 6.3. The x-axis represents the weights value and the y-axis represents their density.

iterative pruning affects the learning process during training and how the model recovers and improves its loss after passing this initial pruning step, which emphasizes the importance of having margins towards the beginning and the end of the pruning process.

6.3 Weight Sharing

Weight sharing consistently results in models that are about a third of the baseline model size. Models compressed with the `density` method, however, lead to slightly higher MAE scores than when using other options, which is likely due to the vanishing representation of large weights, as explained in [18]. The poor results of the models compressed with the random centroid select method are easy to explain by the arbitrary choice of centroids, as depicted in Fig. 5. In contrast to this, centroids selected using any of the other methods capture the actual range of values better, which results in a better overall performance.

Applying clustering only to the dense part of the network is even better, as the convolution part is free from constraints and can maintain its full expressive power this way. This again confirms that the dense part of the network is more prone to overparameterization. Some clustering techniques, like k-means++ , produce models with sparse weights values, which happened to be the result of the selection of a zero-centered group. This may make these models more prone to other compression techniques.

6.4 Depthwise Separable Convolution

Although the MAE of the `dwsc` model is 2 W worse than the baseline model on average, the model achieves these results with a size of 1.01 MB, which is 365 × smaller than the original model. This model thus effectively increases the information content of its parameters. This is also confirmed by the reduced sparsity (cf. Sec. 5.2): Only 1.99 % of the parameters are considered relevant when applying `dwsc`, as compared to 4.67 % in the `baseline` model. Considering the size/performance ratio of this model, it is has demonstrated the superior performance so far, and it is also faster to train and test due to its very small size.

6.5 Pooling

The MAE values of best performing `p-`x`,`y models are shown in Table 3. While we have evaluate other configurations with different x, y values, these exhibited poor results (up to an average MAE of more than 45 W) and were thus eliminated for clarity. For these poor-performing models, the pooling operation hindered learning. The best model we found is `p-2,14975`, which possesses about half the number of the original model's parameters. It is the only pooling-based model which was able to achieve results comparable to the `baseline` model for the microwave and fridge. For the washing machine and dishwasher, many pooled models were able to achieve good accuracy. There is a clear correlation between the performance of these models and their sizes, which is reasonable as larger models contain more parameters and have more expressive power. Overall, the `p-50,599` model provides a good balance between size and performance, as the model is less than 8 MB and still able to achieve an average MAE of less than 23 W, which is the same MAE of the `mixed8` model but with a size of 7.91 MB instead of 23.48 MB.

6.6 Multitask Learning

Both MTL-models achieve comparable results to the baseline models for all appliances, cf. Table 3. The poor results of the `dwsc`-version are due to the enormous reduction in the number of parameters, thus they could be expected. However, the size of this model is only 3.37 MB. A separate evaluation of this model's level of sparsity has moreover shown it to be only 0.31 % higher than `dwsc`. Since this model can be used instead of four `dwsc` models trained for a single appliance, it provides a good trade-off when the disaggregation of mutiple appliances is required for the application use case.

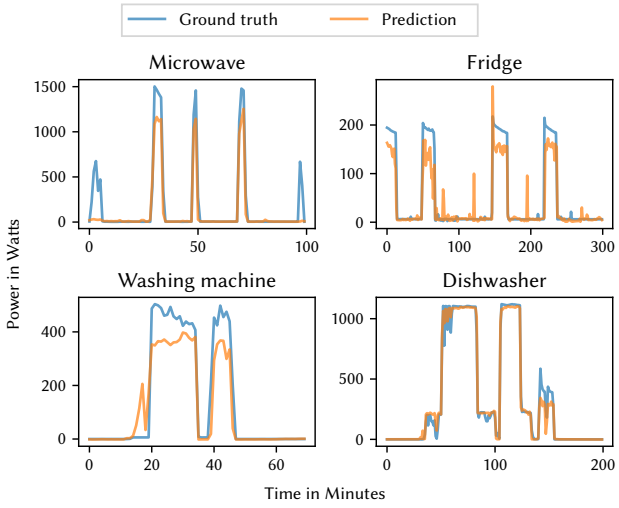

Figure 6: An example of the real power consumption traces along with the disaggregation results for each appliance as generated by the `comb` model.

6.7 Combined Model

Based on the previous results and the size/performance relation, the best configurations among the tested parameter combination for each compression method were identified and chosen to parameterize a final model `comb`. Since the gains from structural pruning were higher, average pooling was not considered for this combined model. MTL was also disregarded, as depthwise convolution on its own produced small models with better performance for each appliance.

The final model is configured as follows:

- Parameter quantization that follows the same technique as the `mixed16` model.
- Pruning with polynomial scheduler. After training, the model is also structurally pruned to eliminate neurons with negligible contribution to the model performance.
- Weight sharing with k-means++ based centroid selection algorithm applied only to the dense part of the network.
- Depthwise separable convolutional for the huge benefits regarding size-performance ratio.

The predictions shown in Fig. 6 were generated using this model. From this figure, it can be seen that the model is occasionally emits wrong predictions: Some activations are missing (in the case of the microwave), or activations are predicted where no actual activations exist (in the case of the fridge and washing machine). Due to the limited expressive power of this model in terms of the number of parameters and their precision, the predictions also tend to have either simpler shapes, like straight lines in the case of the dishwasher, or more noisy patterns as in the case of the fridge, as compared to the `baseline` model. Overall, however, the MAE for the analysis of the entire input dataset, as reported in Table 3, demonstrates a performance that is superior to the baseline, despite its reduced memory demand.

6.8 Discussion and Insights

For comparison, we have plotted the average and best case results for all types of techniques in Fig. 7, showing the tradeoff between achievable MAE and model size (on a logarithmic axis). Moreover, we show the relationship between size and MAE for each model in Fig. 8. `comb` stands out as the best performing model among the smallest. Moreover, we can see that pruning and pooling with the highest number of units are the best methods to improve performance at the expense of size. On the other hand, quantization, depthwise convolution, and smaller pooling models achieve the opposite by producing small models at cost of MAE performance.

Applying depthwise convolution seems to be a key idea for achieving extreme size reductions while maintaining acceptable performance, which we have proven useful for MTL settings as well. Global pooling has been proven effective to average information along different dimensions and reduce the number of parameters. An advantage of pooling is the fact, that it can be applied along different dimensions as necessary to achieve the desired trade-off between size and performance, as shown in Fig. 8. Pruning as a training-aware process can be used to mark unnecessary weights and improve performance (by acting as a regularizer). Combined with a subsequent algorithm to strip out the pruned weights, pruning can greatly reduce the number of parameters without sacrificing accuracy. Finally, weight sharing can be applied for further reductions in model size and possible improvements in performance, as the model is restricted to use fewer weights to better represent the patterns in the data.

The effectiveness of each of these methods depends on the final desired application domain of the model. For instance, for applications on smartphones, where power and size constraints are less strict than on the embedded systems integrated into smart meters, a depthwise convolution may be overly aggressive in removing parameters in a situation where other, less aggressive methods would have sufficed. In power-restricted mobile applications, where sufficient memory is available but the processing power is restricted, pruning and weight sharing can be applied to reduce inference times and computational load without hurting performance. Finally, quantization and MTL can be applied alongside pooling to produce tiny models for embedded applications at generally satisfactory accuracy levels. In fact, techniques like structural pruning, weight sharing, and quantization can even be applied to reduce existing over-parameterized models without the need for re-training.

7 OPERABILITY ON EMBEDDED SYSTEMS

Being able to operate NILM methods on resource-constrained systems, such as the processing devices embedded in smart meters, is crucial for their deployment at large. Verifying the operability of the S2P method with a compressed model is thus of crucial importance. We have thus prepared an S2P model according to the results of our parameter study, documented in Sec. 6.7.

To validate the implemented model for real-live application, each `comb` model for each appliance is exported in the TensorFlow Lite format and transformed to the 4 MB flash memory of an ESP32 system, which is the lowest possible configuration for this MCU module. Models for all four electrical appliances can fit in the embedded flash memory, as their total cumulative size is only 592.9 kB The

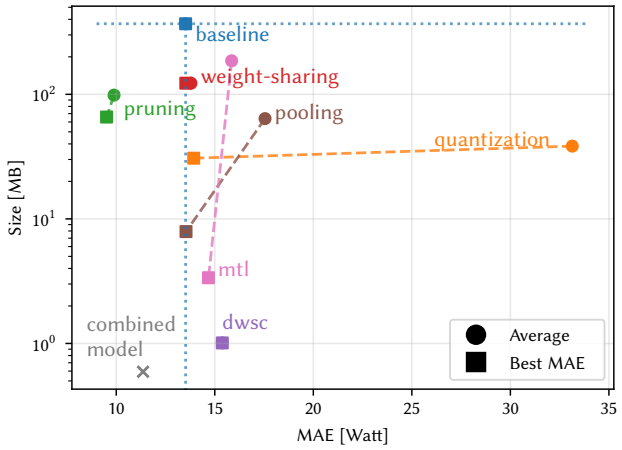

Figure 7: Results of all models grouped by the compression method and expressed by the model accuracy (represented by the MAE) vs. model size.

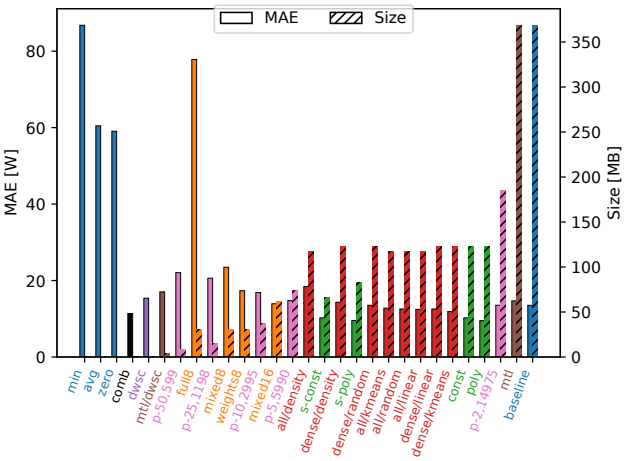

Figure 8: Size and performance of all models sorted along the x-axis by the performance-size ratio.

program loads all models into memory and starts an applications that accepts power readings, executes each model, and transforms back the results. The results are shown in the last row of Table 3.

The model shows a drop in sparsity of more than 3.5 % as compared to the baseline model Sec. 3, which is a good sign that the weights are utilized appropriately. More importantly, the model possesses only 78,019 parameters compared to the 30 million parameter in the baseline model. Thus, it succeeded at achieving *better performance* than the baseline model with an average size of 148.2 kB for each appliance, rendering it *more than* 99 × *smaller* than the baseline.

8 CONCLUSION

In this work, we have explored the possibility of reducing the size of neural network-based NILM-models, in order to execute them on embedded hardware with restricted memory and computation capabilities. Through a comparison of six techniques, it not only became clear that size reductions are possible, but also that the applied techniques can in some cases be used to improve the model accuracy. A fusion of the best-performing methods into a new *combined* model could not only be demonstrated to achieve better performance than the original model at a fraction of the size, but also easily fits on the flash memory of a contemporary MCU. The final model was 99 × smaller than the original model, with a 2 W reduction of the MAE (i.e., an improvement of 15.9 % over the baseline) on average.

Research on neural network compression also opens the opportunity to better comprehend these models, as smaller models are more accessible and easier to understand and interrupt, due to the limited number of variables per layer. A more profound understanding of the relation between the number and location of network parameters can help in making informed choices regarding network size and structure, thus making general recommendations regarding size-performance trade-off.

REFERENCES

[1] Shamim Ahmed and Marc Bons. 2020. Edge Computed NILM: A Phone-Based Implementation Using MobileNet Compressed by Tensorflow Lite. (2020), 5.

[2] K. Carrie Armel, Abhay Gupta, Gireesh Shrimali, and Adrian Albert. 2013. Is Disaggregation the Holy Grail of Energy Efficiency? The Case of Electricity. *Energy Policy* 52, 1 (2013).

[3] Christos Athanasiadis, Dimitrios Doukas, Theofilos Papadopoulos, and Antonios Chrysopoulos. 2021. A Scalable Real-Time Non-Intrusive Load Monitoring System for the Estimation of Household Appliance Power Consumption. *Energies* 14, 3 (Jan. 2021), 767. https://doi.org/10.3390/en14030767

[4] Christos L. Athanasiadis, Theofilos A. Papadopoulos, and Dimitrios I. Doukas. 2021. Real-Time Non-Intrusive Load Monitoring: A Light-Weight and Scalable Approach. *Energy and Buildings* 253 (Dec. 2021), 111523. https://doi.org/10.1016/j.enbuild.2021.111523

[5] Jack Barber, Heriberto Cuayáhuitl, Mingjun Zhong, and Wenpeng Luan. 2020. Lightweight Non-Intrusive Load Monitoring Employing Pruned Sequence-to-Point Learning. In *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring (NILM'20)*. Association for Computing Machinery, New York, NY, USA, 11–15. https://doi.org/10.1145/3427771.3427845

[6] Nipun Batra, Rithwik Kukunuri, Ayush Pandey, Raktim Malakar, Rajat Kumar, Odysseas Krystalakos, Mingjun Zhong, Paulo Meira, and Oliver Parson. 2019. Towards Reproducible State-of-the-Art Energy Disaggregation. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '19)*. Association for Computing Machinery, New York, NY, USA, 193–202. https://doi.org/10.1145/3360322.3360844

[7] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. 2020. What Is the State of Neural Network Pruning? *arXiv:2003.03033 [cs, stat]* (March 2020). arXiv:2003.03033 [cs, stat]

[8] Cillian Brewitt and Nigel Goddard. 2018. Non-Intrusive Load Monitoring with Fully Convolutional Networks. *arXiv:1812.03915 [cs, stat]* (Dec. 2018). arXiv:1812.03915 [cs, stat]

[9] Rich Caruana. 1997. Multitask Learning. *Machine Learning* 28, 1 (July 1997), 41–75. https://doi.org/10.1023/A:1007379606734

[10] Eva Cetinic, Tomislav Lipic, and Sonja Grgic. 2018. Fine-Tuning Convolutional Neural Networks for Fine Art Classification. *Expert Systems with Applications* 114 (Dec. 2018), 107–118. https://doi.org/10.1016/j.eswa.2018.07.026

[11] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. 2015. Compressing Neural Networks with the Hashing Trick. In *International Conference on Machine Learning*. PMLR, 2285–2294.

[12] Xinlei Chen, Moosa Moghimi Haji, and Omid Ardakanian. 2021. Disaggregating Solar Generation Using Smart Meter Data and Proxy Measurements from Neighbouring Sites. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (E-Energy '21)*. Association for Computing Machinery, New York, NY, USA, 225–230. https://doi.org/10.1145/3447555.3464856

[13] Francois Chollet. 2017. Xception: Deep Learning With Depthwise Separable Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1251–1258.

[14] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey. *Proc. IEEE* 108, 4 (April 2020), 485–532. https://doi.org/10.1109/JPROC.2020.2976475

[15] Michele D'Incecco, Stefano Squartini, and Mingjun Zhong. 2019. Transfer Learning for Non-Intrusive Load Monitoring. *arXiv:1902.08835 [cs, stat]* (Sept. 2019). arXiv:1902.08835 [cs, stat]

[16] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. (March 2018).

[17] Hamed Ghoddusi, Germán G. Creamer, and Nima Rafizadeh. 2019. Machine Learning in Energy Economics and Finance: A Review. *Energy Economics* 81 (June 2019), 709–727. https://doi.org/10.1016/j.eneco.2019.05.006

[18] Song Han, Huizi Mao, and William J. Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]* (Feb. 2016). arXiv:1510.00149 [cs]

[19] Song Han, Jeff Pool, John Tran, and William J. Dally. 2015. Learning Both Weights and Connections for Efficient Neural Networks. *arXiv:1506.02626 [cs]* (Oct. 2015). arXiv:1506.02626 [cs]

[20] George W. Hart. 1985. *Prototype Nonintrusive Appliance Load Monitor*. Technical Report. MIT Energy Laboratory and Electric Power Research Institute.

[21] George W. Hart. 1992. Nonintrusive Appliance Load Monitoring. *Proc. IEEE* 80, 12 (Dec. 1992), 1870–1891. https://doi.org/10.1109/5.192069

[22] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]* (April 2017). arXiv:1704.04861 [cs]

[23] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]* (Jan. 2018). arXiv:1608.06993 [cs]

[24] Patrick Huber, Alberto Calatroni, Andreas Rumsch, and Andrew Paice. 2021. Review on Deep Neural Networks Applied to Low-Frequency NILM. *Energies* 14, 9 (Jan. 2021), 2390. https://doi.org/10.3390/en14092390

[25] Jana Huchtkoetter and Andreas Reinhardt. 2020. On the Impact of Temporal Data Resolution on the Accuracy of Non-Intrusive Load Monitoring. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '20)*. Association for Computing Machinery, New York, NY, USA, 270–273. https://doi.org/10.1145/3408308.3427974

[26] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5MB Model Size. *arXiv:1602.07360 [cs]* (Nov. 2016). arXiv:1602.07360 [cs]

[27] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. PMLR, 448–456.

[28] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2017. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *arXiv:1712.05877 [cs, stat]* (Dec. 2017). arXiv:1712.05877 [cs, stat]

[29] Jonghoon Jin, Aysegul Dundar, and Eugenio Culurciello. 2015. Flattened Convolutional Neural Networks for Feedforward Acceleration. *arXiv:1412.5474 [cs]* (Nov. 2015). arXiv:1412.5474 [cs]

[30] Florian Kalinke, Pawel Bielski, Snigdha Singh, Edouard Fouché, and Klemens Böhm. 2021. An Evaluation of NILM Approaches on Industrial Energy-Consumption Data. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems (E-Energy '21)*. Association for Computing Machinery, New York, NY, USA, 239–243. https://doi.org/10.1145/3447555.3464863

[31] Jack Kelly and William Knottenbelt. 2015. Neural NILM: Deep Neural Networks Applied to Energy Disaggregation. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments (BuildSys '15)*. Association for Computing Machinery, New York, NY, USA, 55–64. https://doi.org/10.1145/2821650.2821672

[32] Jack Kelly and William Knottenbelt. 2015. The UK-DALE Dataset, Domestic Appliance-Level Electricity Demand and Whole-House Demand from Five UK Homes. *Scientific Data* 2, 1 (March 2015), 150007. https://doi.org/10.1038/sdata.2015.7

[33] J Zico Kolter and Matthew J Johnson. 2011. REDD: A public data set for energy disaggregation research. In *Workshop on data mining applications in sustainability (SIGKDD), San Diego, CA*, Vol. 25. 59–62.

[34] Raghuraman Krishnamoorthi. 2018. Quantizing Deep Convolutional Networks for Efficient Inference: A Whitepaper. *arXiv:1806.08342 [cs, stat]* (June 2018). arXiv:1806.08342 [cs, stat]

[35] Odysseas Krystalakos, Christoforos Nalmpantis, and Dimitris Vrakas. 2018. Sliding Window Approach for Online Energy Disaggregation Using Artificial Neural Networks. In *Proceedings of the 10th Hellenic Conference on Artificial Intelligence (SETN '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3200947.3201011

[36] Rithwik Kukunuri, Anup Aglawe, Jainish Chauhan, Kratika Bhagtani, Rohan Patil, Sumit Walia, and Nipun Batra. 2020. EdgeNILM: Towards NILM on Edge Devices. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation (BuildSys '20)*. Association for Computing Machinery, New York, NY, USA, 90–99. https://doi.org/10.1145/3408308.3427977

[37] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed Point Quantization of Deep Convolutional Networks. In *International Conference on Machine Learning*. PMLR, 2849–2858.

[38] Min Lin, Qiang Chen, and Shuicheng Yan. 2014. Network In Network. *arXiv:1312.4400 [cs]* (March 2014). arXiv:1312.4400 [cs]

[39] David Murray, Lina Stankovic, and Vladimir Stankovic. 2017. REFIT: An Electrical Load Measurements Dataset of United Kingdom Households from a Two-Year Longitudinal Study. *Scientific Data* 4, 1 (Jan. 2017), 160122. https://doi.org/10.1038/sdata.2016.122

[40] Steven J. Nowlan and Geoffrey E. Hinton. 1992. Simplifying Neural Networks by Soft Weight-Sharing. *Neural Computation* 4, 4 (July 1992), 473–493. https://doi.org/10.1162/neco.1992.4.4.473

[41] Yungang Pan, Ke Liu, Zhaoyan Shen, Xiaojun Cai, and Zhiping Jia. 2020. Sequence-To-Subsequence Learning With Conditional Gan For Power Disaggregation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 3202–3206. https://doi.org/10.1109/ICASSP40776.2020.9053947

[42] Wei Ping, Kainan Peng, Kexin Zhao, and Zhao Song. 2020. WaveFlow: A Compact Flow-based Model for Raw Audio. In *International Conference on Machine Learning*. PMLR, 7706–7716.

[43] Gautam A. Raiker, Subba B. Reddy, L. Umanand, Aman Yadav, and Mujeefa M. Shaikh. 2018. Approach to Non-Intrusive Load Monitoring Using Factorial Hidden Markov Model. In *2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS)*. 381–386. https://doi.org/10.1109/ICIINFS.2018.8721436

[44] Andreas Reinhardt and Mazen Bouchur. 2020. On the Impact of the Sequence Length on Sequence-to-Sequence and Sequence-to-Point Learning for NILM. In *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*. ACM, Virtual Event Japan, 75–78. https://doi.org/10.1145/3427771.3427857

[45] Andreas Reinhardt and Christoph Klemenjak. 2020. How Does Load Disaggregation Performance Depend on Data Characteristics? Insights from a Benchmarking Study. In *Proceedings of the Eleventh ACM International Conference on Future Energy Systems (E-Energy '20)*. Association for Computing Machinery, New York, NY, USA, 167–177. https://doi.org/10.1145/3396851.3397691

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4510–4520.

[47] Nima Tajbakhsh, Jae Y. Shin, Suryakanth R. Gurudu, R. Todd Hurst, Christopher B. Kendall, Michael B. Gotway, and Jianming Liang. 2016. Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE Transactions on Medical Imaging* 35, 5 (May 2016), 1299–1312. https://doi.org/10.1109/TMI.2016.2535302

[48] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. *arXiv:1609.03499 [cs]* (Sept. 2016). arXiv:1609.03499 [cs]

[49] Zhenrui Yue, Camilo Requena Witzig, Daniel Jorde, and Hans-Arno Jacobsen. 2020. BERT4NILM: A Bidirectional Transformer Model for Non-Intrusive Load Monitoring. In *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*. Association for Computing Machinery, New York, NY, USA, 89–93.

[50] Chiyuan Zhang, Samy Bengio, and Yoram Singer. 2019. Are All Layers Created Equal? *arXiv:1902.01996 [cs, stat]* (May 2019). arXiv:1902.01996 [cs, stat]

[51] Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton. 2017. Sequence-to-Point Learning with Neural Networks for Nonintrusive Load Monitoring. *arXiv:1612.09106 [cs, stat]* (Sept. 2017). arXiv:1612.09106 [cs, stat]

[52] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2017. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *arXiv:1707.01083 [cs]* (Dec. 2017). arXiv:1707.01083 [cs]

[53] Michael Zhu and Suyog Gupta. 2017. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. *arXiv:1710.01878 [cs, stat]* (Nov. 2017). arXiv:1710.01878 [cs, stat]