Inference Time Reduction of Deep Neural Networks on Embedded Devices: A Case Study

Isma-Ilou Sadou
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
sadou@kth.se

Seyed Morteza Nabavinejad
School of Computer Science
Institute for Research in Fundamental Sciences (IPM)
Tehran, Iran
nabavinejad@ipm.ir

Zhonghai Lu
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
zhonghai@kth.se

Masoumeh Ebrahimi
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
mebr@kth.se

***Abstract*—From object detection to semantic segmentation, deep learning has achieved many groundbreaking results in recent years. However, due to the increasing complexity, the execution of neural networks on embedded platforms is greatly hindered. This has motivated the development of several neural network minimisation techniques, amongst which pruning has gained a lot of focus. In this work, we perform a case study on a series of methods with the goal of finding a small model that could run fast on embedded devices. First, we suggest a simple, but effective, ranking criterion for filter pruning called Mean Weight. Then, we combine this new criterion with a threshold-aware layer-sensitive filter pruning method, called T-sensitive pruning, to gain high accuracy. Further, the pruning algorithm follows a structured filter pruning approach that removes all selected filters and their dependencies from the DNN model, leading to less computations, and thus low inference time in lower-end CPUs. To validate the effectiveness of the proposed method, we perform experiments on three different datasets (with 3, 101, and 1000 classes) and two different deep neural networks (i.e., SICK-Net and MobileNet V1). We have obtained speedups of up to 13x on lower-end CPUs (Armv8) with less than 1% drop in accuracy. This satisfies the goal of transferring deep neural networks to embedded hardware while attaining a good trade-off between inference time and accuracy.**

***Index Terms*—Deep neural network optimisation, embedded deep learning, embedded devices, edge AI, pruning.**

I. Introduction

Machine learning has been rapidly advancing over the past couple of decades and is now a common tool used in many tasks that require extracting information from a large set of data [1]. One of the main reasons behind this revolution in the applications of machine learning is the advancement and introduction of deep learning [2].

With the advancement of neural networks over the last decade, the prediction error has been continuously dropping. This improvement is, however, accompanied by deeper neural networks, which results in bigger networks, higher computation costs, and longer inference times. A complicated model such as Inception-ResNetv2 [3] contains 55.8 million parameters and requires 13 billion operations for a single inference (208.8 ms on Intel i7-8700k or 20.6 ms on Nvidia 1080ti). Due to this complexity, the execution of deep neural networks usually has to be deployed on GPUs, leading to high acquisition and energy costs [4]–[6].

The need for smart sensor solutions is also on the rise and it is not feasible to have GPUs in all sensors as they are bulky and power hungry. The sensors are expected to run simple classification or detection algorithms without needing additional industrial PCs or servers. An alternative to using energy-hungry GPUs in Edge AI is hardware and FPGA-based accelerators [7]–[9], but they result in a loss of flexibility compared with the general purpose CPUs.

To exploit the success achieved by deep learning in industrial and embedded applications, ways need to be found to port them to commercially available (lower-end) CPUs, while ensuring that the fast reaction time and accuracy requirements (maximum 1% loss) are also met. Many model minimisation techniques such as pruning [10] and quantization [11], knowledge distillation [12], etc.), inference optimisations (vector merging [13], deep reuse [14], etc.), and new architectures (Squeezenet [15], MobileNet [16], etc.) have been proposed to address this problem. However, the expected results from many of these techniques, aiming at reduction in size and computation, can usually not be easily obtained without specialised software and/or hardware. In addition, many of the proposed methods only reported results on GPUs and higher-end CPUs.

Building on top of the previous works, this paper proposes a Mean Weight ranking criterion combined with a threshold-aware layer-sensitive filter pruning strategy with the goal of transferring deep neural networks to embedded hardware while attaining a good trade-off between inference time and accuracy. The proposed criterion is simple to compute and targets filters (rather than individual weights) to eliminate the need for specialised hardware or software to exploit the resulting sparsity while the layer-sensitive pruning strategy adds data awareness to the pruning strategy. We assume the accepted accuracy drop of maximum 1% when investigating different trade-offs.

To validate the effectiveness of different approaches, we first evaluate it on an industrial classification scenario with 3 classes (Wood dataset) and an already optimised neural

network (SICK-Net). For scalability purposes, the procedure is then applied to a not typical problem with 101 classes [17] and the prevailing ImageNet dataset [18] with 1000 classes and a slightly larger MobileNet-V1 model [16]. The main contributions of this work are as follows:

- We evaluate different filter ranking criteria while proposing a simple but effective filter ranking criterion called Mean Weight. This criterion identifies the least important filters for pruning based on the average value of filters' weights.
- We suggest a variant of the layer-sensitive pruning strategy, called T-sensitive pruning, and combine Mean Weight with it to accelerate DNN inference on resource-constraint devices with minimum accuracy loss.
- We introduce a structured filter removal algorithm to prune filters and all subsequent dependent filters from the DNN model to reduce computation and speed up the inference. This can include removing an entire layer.
- We conduct extensive experiments to show the applicability of our approach on different higher-end and lower-end hardware devices, industrial and conventional DNNs with different amount of computational complexities, and datasets with different number of classes. We show how a simple approach can surpass previous works in terms of inference latency and accuracy.

The organization of the rest of the paper is as follows: We discuss the ideas presented in previous works in Section II. The background needed to follow the paper is presented in Section III. The design and implementation of our proposed approach is explained in Section IV. The experimental results are presented in Section V, and the paper is concluded in Section VI.

II. Previous work

A large body of research has focused on enabling real-time power/energy-efficient DNN inference on embedded devices [19]–[21]. Since the computing capacity and maximum available power of embedded devices is less than conventional CPUs, GPUs, and FPGAs, it is challenging to simply deploy the DNNs designed for such devices on embedded devices. Thus, various techniques such as quantization [21], [22] and pruning [19], [20] are employed to reduce the resource demand of DNNs, and hence, improve their performance on embedded devices.

The quantization technique is widely used to improve the latency of DNN inference [23]–[26]. In this technique, the DNN parameters are quantized from high-precision floating point (e.g., 32-bit floating point) to lower precision floating point (e.g., 16-bit floating point) or integer (e.g., 8/4-bit integer). Since the low-precision floating point or integer arithmetic operations can be executed faster than high-precision floating point ones, this technique can improve the latency of inference, while negatively affecting the accuracy. The improvement can be achieved by quantization technique provided that the target hardware processor is equipped with an exclusive instruction set that supports the reduced-precision operations (e.g., 4-bit integer).

The pruning strategies employed for compressing the DNNs can be mainly divided into two categories: unstructured and structured pruning. The most popular unstructured pruning strategies, called *weight pruning* [27]–[29], leverage the weight redundancy in the DNNs and try to achieve model compression with slight accuracy loss. Since the weights are only zeroed-out, these strategies usually obtain low (or even no) latency improvement. The most popular structured pruning strategies on the other hand, called *filter pruning*, consider pruning the filters in convolution layers instead of weights [10], [30], [31]. Unlike weight pruning, filter pruning completely removes the selected filters from the model. The complete removal of filters can help gain performance improvement, in the form of decreased inference time. An example for weight and filter pruning is illustrated in Figure 1 (a) and (b), respectively.

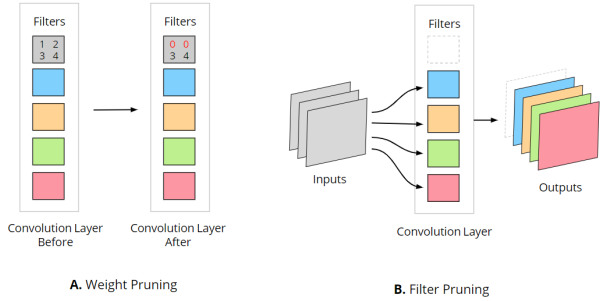

Fig. 1. Weight vs Filter pruning

Different criteria have been proposed to identify the perfect candidate filters for removal such as L1-norm [10] or using Taylor expansion [31] to approximate the impact of each filter on the loss function of the DNN. These criteria are either too complex, imposing significant ranking overhead, and/or mainly target GPUs. To address these challenges, our investigation focuses on a simpler criterion which is more effective in identifying and removing the least important filters, and could achieve higher performance improvement (i.e., faster inference with acceptable level of accuracy loss) on embedded devices.

III. Background

In this section, we introduce the basic operations of a convolutional neural network (CNN) before talking about the optimisation works we deemed most relevant. A typical neural network consists of multiple interconnected layers. Each of these layers is made up weight tensors which are used in processing inputs. In the case of a convolutional layer, these weight tensors are grouped into entities called filters.

As shown in Fig. 2, filter pruning, unlike weight pruning, requires the removal of complete filters from the DNN model in question instead of replacing them with zeros. In this figure green and yellow filters and their dependencies are completely removed from the entire model. This eliminates the need for specialised hardware/software to benefit from the resulting sparsity.

The three main prerequisites of filter pruning are: 1) A criterion to rank the filters by importance (Subsection IV-A);

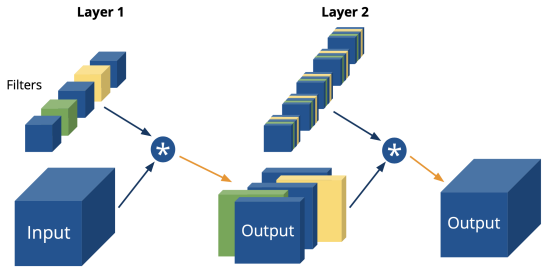

Fig. 2. Pruning dependencies when the green and yellow filters in Layer 1 are removed

2) A strategy to remove filters either layer-by-layer or globally (Subsection IV-B); and 3) A method to remove subsequent dependent filters (all dependencies) from the next layers (Subsection IV-C). The ultimate goal is to reach a model that could fit and run fast on lower-end CPUs.

IV. Preparing DNN for Execution on Embedded Devices

This paper aims at studying and proposing light-weight solutions to accelerate DNN models for deployment on embedded devices by removing as many filters as possible with an acceptable loss in accuracy (1% in this work). To achieve this goal, we need to solve the problems mentioned in the previous sections. We investigate the saliency of weights and their extendibility to structured weight pruning. By considering actual weight values in determining the importance of filters, we propose a simple ranking criterion that exploits the proven relationship between weight and model accuracy [32]–[35], while having the same benefits from the lack of sparsity.

This section first describes well-known criterion to rank the filters, including L1-Norm and Taylor expansion. Then we suggest a simpler and more effective criterion called Mean Weight ranking. After that, we combine all criteria with two different pruning strategies to find out the best solution. One strategy rank all filters in the model and remove the least important ones while the other one remove filters layer by layer. Finally, a structured algorithm is introduced which removes the selected filters and their dependencies from the whole model for faster inference (see Fig. 2).

A. Criterion to rank filters

Different works use different methods and criterion to rank filters by their importance [10], [31], [36]. In AutoML, the authors propose training a reinforcement learning agent to predict actions and give structured sparsity [36]. However, DNNs are mathematical and it is technically possible to measure the exact importance of parameters by removing them one by one and evaluating the changes in different performance metrics (FLOPS, size, etc.). This method is called Oracle and is the most optimal ranking criterion, albeit too expensive [31].

In the following we draw comparison with L1-norm and Tailor expansion ranking criteria.

*1) **L1-norm Ranking**:* In [10], Hao Li et. al. proposed a layer-wise pruning acceleration method that uses the L1-norm (absolute sum of weights) of filters to rank them. This simple criterion gives an estimate of the filter's magnitude and hence their saliency. For each filter, its magnitude is calculated by Eq. (1):

$$Magnitude = \sum_{i=1}^{n} |W_i| \tag{1}$$

where W_i is a weight in the filter and n is the number of weights in the filter. An example of filter selection based on this criterion is shown in Fig. 3 (a).

*2) **Taylor expansion Ranking**:* Concurrently with the work above, NVIDIA proposed an iterative global filter pruning approach that interleaves greedy criteria-based pruning with fine-tuning by back-propagation [31]. Rephrasing pruning as an optimisation problem (see Eq. (2)), they propose a new criterion based on a first order Taylor expansion of the network cost function to approximate the change in the loss function when removing a particular parameter.

$$|\Delta C(W_i)| = |C(D, W_i = 0) - C(D, W_i)| \tag{2}$$

where $C(D, W_i = 0)$ is the cost model of D if filter W_i is pruned. Fig. 3 (b) shows an example of filter selection based on this criterion.

It is prohibitively costly to compute Oracle criterion which empirically evaluates each parameter. The iterative pruning strategy they employ was introduced by early works in the 1990s [37], [38] for weights. This approach involves pruning k% of the least important filters, and retraining the DNN model to recover the lost accuracy. This process is repeated until the desired optimisation or the accuracy loss constraint is reached. This work achieves up to 1.7x speedup for the VGG-16 with the ImageNet dataset on a higher-end Intel i7-5930K CPU and up to 2.5x on an NVIDIA Jetson TX1 GPU with 2.3% accuracy loss. These results are pretty impressive considering all the feautures in the 1000 classes. However, the Taylor criterion is too complex. A point of interest is to see if such complex filter ranking criteria can be outperformed by simpler criteria. This work also mainly targeted higher-end CPUs and GPUs.

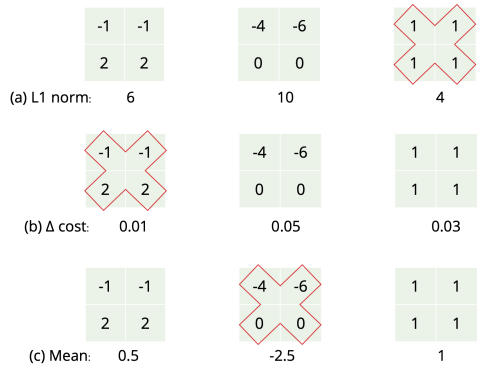

Fig. 3. Least important filter: a) L1-norm ranking, b) Taylor ranking, c) The proposed Weight Mean ranking

*3) **Mean Weight Ranking**:* We propose to extend magnitude-based weight pruning criterion to filters. Instead of sorting filters based on the sum of absolute values as in L1-norm (Eq. (1)), we propose to take an average sum (Eq (3)). This criterion takes a more conservative approach in removing filters. Weight Mean considers the importance of all weights in the filter by taking into account their positive and negative values. This is in contrast with L1-norm where both negative and positive value increases the importance of a filter rather than negating each other. In Weight Mean, by using the average weight of a filter as its magnitude, this saliency criterion has both the benefits of filter pruning as well as the performance and simplicity of magnitude-based weight ranking. This reduces the complexity of Taylor ranking and is more flexible than the L1-norm criterion. The magnitude of a filter with this criterion is given by Eq (3):

$$Magnitude = \frac{1}{|n|}\sum_{i=1}^{n} W_i \tag{3}$$

where W_i is a weight in the filter and n is the number of weights in the filter. Fig. 3 (c) shows an example of filter selection based on this criterion. Generally, Fig. 3 highlights the fact that different criteria result in different filter removal.

B. Strategy on removing filters

After defining the filter ranking criterion, the next step is to define a threshold line to remove the ranked filters. In the following, two pruning strategies we evaluated. The first strategy is based on ranking all filters in the entire model and then remove the k% least important filters, retraining the model, and repeating this process, thereby the strategy is called *iterative global* pruning. In the second strategy, a global threshold is defined and then based on the sensitivity of a layer, a percentage of the ranked filters are removed in that layer. After pruning filters in each layer, the whole model is retrained only once. This strategy is called T-sensitive pruning.

*1) **Iterative global pruning**:* This strategy involves ranking all the parameters of the model, pruning k% of the least important ones, and retraining to recover the lost accuracy. The process is repeated until the desired speedup or the accuracy loss constraint is reached. The amount of filters pruned in each iteration is usually determined through experimentation. In this pruning strategy, we propose to prune 8% of the least important filters in the model in each iteration. This allows the model to be retrained frequently and its accuracy to stay as close to the original one as possible. This, however, results in many more iterations and retraining which is a costly process. Different pruning rates (8%, 16%, etc.) were evaluated and the rate of 8% per iteration was determined was found to recover quality faster when fine-tuning.

*2) **Layer-sensitive pruning**:* Although iterative global pruning targets the least important filters in the model, it ignores the relationship between different layers, and also between filters in the same layer. Layer-wise pruning, however, considers this relationship. Layer-sensitive pruning, as a type of layer-wise pruning strategy, was introduced in [10] and takes into consideration the sensitivity of each layer in the model. Each layer is pruned with increments, and the accuracies are extracted without retraining. This represents the sensitivity of each layer and determines how much of each layer to prune. In this strategy, the model is pruned and retrained only once. Fig. 4 shows the sensitivity of 11 layers in the SICK-Net with pruning increments of 10%. This means, for instance, that 50% of both the Conv_1 and Conv_11 layers can be pruned without losing any accuracy. It should be noted that this is the accuracy after pruning, and some of it can be recovered by retraining the pruned model. In this method, the number of filters pruned per layer is decided empirically according to the sensitivity of each layer to pruning. They achieve about 30% reduction in FLOPs for the VGGNet (on CIFAR-10) and deep ResNets without significant loss in the original accuracy [10]. This work provides a great foundation for filter pruning but it does not consider the benefits of ranking filters globally and performing pruning in the entire model. Although the concept of layer sensitivity introduces some level of globality, ranking filters layer-wise only shows their level of importance in that layer and not in the entire model. Actual inference time and real speed up are also not reported.

*3) **Threshold-based Layer-sensitive pruning**:* To determine and automate what percentage of filters to prune in each layer, we extend this pruning strategy with the concept of pruning threshold. The pruning threshold is the accuracy expected of each layer before retraining. This is represented by the horizontal line in the sensitivity graph on Fig. 4 (60%), and where it intersects with each layer represents the percentage of filters to prune. For example, considering the threshold of 60% in Fig. 4, around 90% of filters will be pruned in Conv_8. In the case of Conv_6, where it does not intersect with the pruning threshold, all filters may be pruned. The percentage of filters pruned in each layer is denoted by s(layer)% in Alg. 1. We name this enhancement strategy, T-sensitive pruning.

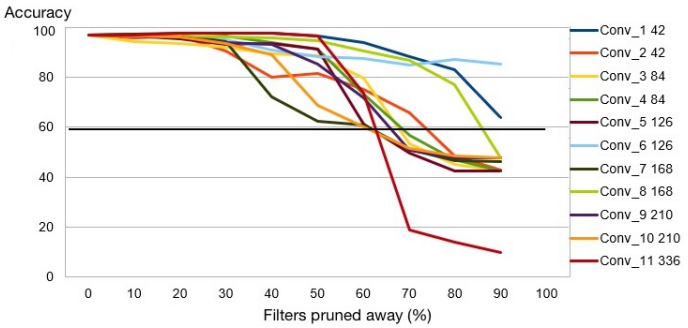

Fig. 4. Sensitivity diagram of the SICK-net on the Wood dataset

C. Method to remove dependencies

In this section, we describe how the selected filters and their dependencies to next layers are actually removed from the whole model for faster inference. After determining the least important filters with the proposed ranking criterion, the model complexity is reduced for execution on lower-end CPUs. Algorithm 1 shows the three main functions used to determine

the filters to be pruned and how they are eventually cut out of the network, reducing model complexity.

Algorithm 1: Filter Removal Algorithm

```
Function Rank_Filters (Model, Criterion, Pruning_Strategy):
    Evaluate the saliency of every filter in Model according to
      Criterion;
    if Pruning_Strategy == Iterative global pruning then
        Add smallest k% filters (and corresponding layers) in the
          whole model to To_be_pruned, where k is determined
          experimentally;
        return To_be_pruned
    Else if Pruning_Strategy == T-sensitive pruning then
        for each layer in Model do
            Add smallest s(layer)% filters in layer to To_be_pruned,
              where s is the layer sensitivity threshold;
        return To_be_pruned
-----------------------------------------------------------------
Function Clone_Model (Model, To_be_pruned):
    for each layer in Model do
        if any To_be_pruned in layer then
            Remove s(layer)% or k% filters from that particular
              layer depending on the selected strategy;
    return Model
-----------------------------------------------------------------
Function Prune_Filters (Model, To_be_pruned):
    New_model = Clone_Model (Model, To_be_pruned);
    Pruned_layer = [];
    for each layer in Model do
        if any To_be_pruned in layer then
            Remove corresponding output channels from weights
              dimensions;
            Append layer to Pruned_layer set;
        if layer is data dependent on any Pruned_layer then
            Remove corresponding input channels from weights
              dimensions;
        Set weights to corresponding layer of the New_model;
    return New_model
```

The *Rank_Filters* function takes as input the *Model*, ranking *Criterion*, and *Pruning_Strategy*. It first evaluates the importance of each filter based on the *Criterion*. It then ranks them globally or layer-wise depending on the *Pruning_Strategy* and returns the smallest filters to be pruned and their corresponding layers in a dictionary called *To_be_pruned*. Layer sensitivity should be calculated prior as shown in Fig. 4 in the case of the *T-sensitive* pruning strategy. This ranking can be done multiple times for the *iterative global* pruning strategy.

The *Prune_Filters* function takes as input the *Model* and *To_be_pruned*, the dictionary of the filters to be removed. The function starts by cloning the current model using the *Clone_Model* function, removing the filters in *To_be_pruned* parameter from the *Model* configuration. It then loads the weights from the old model into the pruned model, layer by layer. This involves deleting the weights corresponding to the removed filters and any of their dependencies. This data dependency can be seen in Fig. 2 where the green and yellow filters are removed from the first layer and all the subsequent dependent layers.

V. Evaluation

We empirically study and compare the pruning criteria and procedure detailed in Section IV for three experiments: 1) a 3-class wood flipper application on the SICK-Net; 2) a 101-class food recognition experiment on the MobileNetv1; and 3) An Imagenet dataset on the MobileNetv1. Training and pruning are performed on the first two experiments, and results are reported for the AMD Ryzen 7 workstation, the Armv8 Raspberry Pi 4. The best joint solution (i.e., the highest and most stable speedup) is then evaluated on an embedded camera.

A. Experimental Setup

TABLE I
DNN models benchmark

	SICK-Net	**MobileNetv1**
Dataset	Wood	Food & ImageNet
Accuracy	96.85%	72.24%
Inference Ryzen 7	4.84 ms	36.32 ms
Inference Armv8	33.14 ms	142.50 ms
# of layers	42	100
# of parameters	2.15 M	4.38 M

As listed in Table I, the SICK-Net and MobileNetv1 [16] CNN models are considered in this work. SICK-Net is a protected industrial DNN and cannot be disclosed beyond the configurations shown in Table I. MobileNetv1 is a small-size CNN model for image classification and is widely used for resource-constrained devices. The PyTorch and Tensorflow packages are used for neural network processing, and the trained DNNs are then sent for optimisation experiments. The optimisation procedures are implemented in Python.

Three datasets are used to evaluate the scalability of the solution: 1) Wood dataset which is an internal dataset for wood classification with three classes and 9409 images; 2) Food [17] with 101 classes and 101,000 images. It contains images of food that are organized by type; and 3) ImageNet [18] with 1000 classes, which is a popular dataset that is widely used in previous works.

TABLE II
Specifications of devices used in the experiments

	AMD Ryzen 7 3700x	**Armv8**	**Armv7**
Device	Workstation	Raspberry Pi 4	Lector621 Camera
CPU type	Higher-end	Lower-end	Embedded Camera
CPU freq.	3.6 GHz	1.5 GHz	1 GHz
RAM	32 GB	4 GB	1 GB
OS	Ubuntu 18.04	Ubuntu 18.04	None (BusyBox)
Use	Benchmark	Benchmark	Validation

We implemented the benchmark framework for comparing the optimisation techniques in C++. The OpenCV package loads the trained or optimised DNNs and records the time it takes for a single inference. The program is cross-compiled with OpenCV DNN as inference back-end and the binaries are executed on two benchmark devices presented in Table II. The employment of three different systems allows to highlight the importance of DNN optimisation on resource constrained devices and the possibility of Edge AI. The embedded camera

is unusable at the start due to the high latency. The benchmark devices are used for evaluation of the proposed approach, and the latter for portability in a real-world setup.

B. Accuracy

Fig. 5 show the accuracy against filters removed for each experiment with the proposed Mean Weight criterion as well as the two other criteria as L1-norm and Taylor. This quantity is either the total number of filters pruned in the model (%) in the case of *iterative global* pruning or the pruning threshold (%) for *T-sensitive* pruning.

Fig. 5 (a) and (c) show how different ranking criteria perform with the *iterative global* pruning strategy on different experiments. As an example, in Fig. 5 (a) when pruning 72% of all filters, the accuracy drops to 90% under Mean Weight criterion that is beyond the acceptable 1% loss. Both L1-norm and Taylor ranking maintain an accuracy loss of around 1% after pruning away 72% of the filters in SICK-Net.

As illustrated in Fig. 5 (b) and (d), the *T-sensitive* pruning strategy introduces data-awareness and globality to the Mean Weight and L1-norm ranking criterion. In Fig. 5 (b), at a pruning threshold of 60%, an accuracy of the Mean Weight criterion drops by -0.82% to 96.38%. In the L1-norm, an accuracy drops by -4.76% to 92.09%. At a pruning threshold of 60%, in the Taylor criterion, the accuracy drops by -0.47% to 96.37%.

C. Parameter Count

Iterative global and *T-sensitive* pruning strategies lead to different amount of pruned parameters. Fig. 6 illustrates how many parameters are removed when either *iterative global* or *T-sensitive* strategies are applied. As can be seen in the figure, *T-sensitive* leads to extensive parameter removal in Fig. 6 (b).

D. Inference Time

In all performed experiments, layer sensitive pruning strategy is replaced with T-sensitive pruning (Subsection IV-B). Moreover, in all examined methods, the filters and all their dependencies are removed from an entire DNN model (Subsection IV-C). In this way, we can fairly compare and analysis different criteria and pruning strategies under different settings as follows:

1) ***SICK-net DNN with the Wood dataset****:* This set up represents our industrial use case. Fig. 7 reports the inference time against the quantity of filters removed for each experiment. As can be seen in the figure, the speedups on both ARMv8 and AMD Ryzen 7 are quite similar.

Fig. 7 (a) show how different ranking criteria perform with the *iterative global* pruning strategy. As an example in Fig. 7 (a), when pruning 72% of all filters, the Mean Weight criterion provides significant speedups of up to 16.05x while the L1-norm criterion leads to the speedups of 3.69x. In Taylor ranking, as most filters are pruned from deeper layers, it provides lesser speedups of up to 2.55x.

Fig. 7 (b) show the results on the *T-sensitive* pruning strategy when Mean Weight and L1-norm ranking criterion are applied. As an example in Fig. 7 (b), at a pruning threshold of 60%, the Mean Weight criterion gives a speedup of 12.99x. The speedup is 17.71x with L1-norm ranking. The Taylor ranking, on the other hand, is already data-aware as it uses normalised gradients and activations to calculate ranking and thus the improvement is minute. At a pruning threshold of 60%, the Taylor criterion gives a speedup of 1.71x. The results obtained from pruning experiments on our industrial case (Wood dataset on an Armv8 sensor) are summarised in Table III, keeping in mind the maximum acceptable accuracy drop of 1%.

The speedups obtained from the Taylor ranking criterion in both pruning strategies and DNN models on the Wood dataset are comparatively quite low, and thus it is discarded from the remaining sets of experiments.

2) ***MobileNet-v1 DNN with the Food-101 dataset****:* Fig. 7 (c) and (d) show the results for the *iterative global* and *T-sensitive* pruning strategies, respectively. As shown in Fig. 7 (c) and (d), Mean Weight and L1-norm ranking criteria lead to similar speedups under both pruning strategies. The speedups, however, are smaller than the lighter model of SICK-Net under the Wood dataset (Fig. 7 (a) and (b)). Looking into all experiments in Fig. 7, we observe that a higher speedup is obtained when the Mean Weight ranking is combined with T-sensitive pruning. Thereby, we dig deeper into this combination in the third set of experiments.

3) ***MobileNetv1 with the ImageNet dataset****:* In this set of experiments, a MobileNetv1 model trained on the ImageNet dataset is analyzed. The Mean Weight ranking criterion is combined with the *T-sensitive* pruning to show the applicability of this combination on the well-known ImageNet dataset. Being an already compact model on the Imagenet dataset, Top-1 accuracy of the small MobileNetv1 is quite low and starts from 65.40%. With T-Sensitive Mean Weight pruning (henceforth TS-MW) Top-1 accuracy drops from 65.40% to 60.05% after a 29.9% reduction in the number of parameters, resulting in a 1.3x speedup as shown in Fig. 9. Because of the compactness of MobileNetv1, many state-of-the-art papers evaluate their results on smaller datasets such as Paupamah *et al.* [23] which gets a 1.6x compression of the MobileNet on CIFAR10 with no accuracy loss compared to our 1.4x compression on the Imagenet with 4.9% accuracy loss. Others rather use larger models such as Molchanov *et al.* [31] which gets a 1.7x speedup of the VGG-16 (Imagenet) on i7-CPU with a 2.3% accuracy loss compared to our 1.3x speedup of the MobileNetv1 (Imagenet) on armv8 with 4.9% accuracy loss. This not only shows the comparability of our simple TS-MW pruning criterion but also its applicability in the embedded world. The accuracy, inference time, and parameter for each pruning threshold is given in Fig. 9. The results are also summarised in Table III.

E. Portability to Embedded Camera

Embedded cameras are highly used in industry automation. Edge inference on the low-energy Lector621 (wood flipper) led to a high latency of 598.1 ms. To reduce the latency, the SICK-Net model is pruned with a similar configuration as in Fig. 5 (a) and (b). The pruned SICK-Net models are then deployed on the camera, and as the result, latency dropped to 55.9 ms with our proposed TS-MW. This satisfies the goal of porting

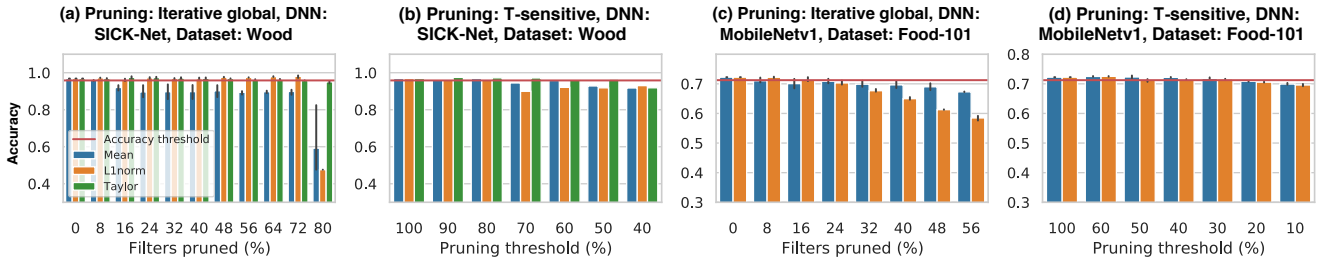

Fig. 5. Evaluating the different strategies and ranking criteria with regard to accuracy

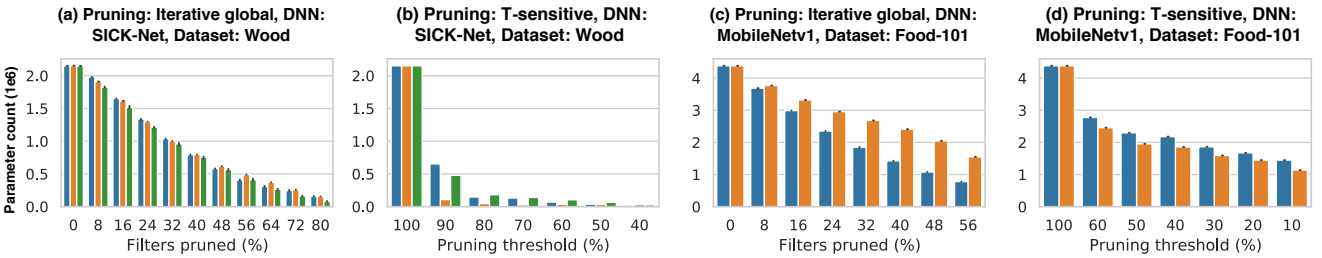

Fig. 6. Evaluating the different strategies and ranking criteria with regard to parameter count

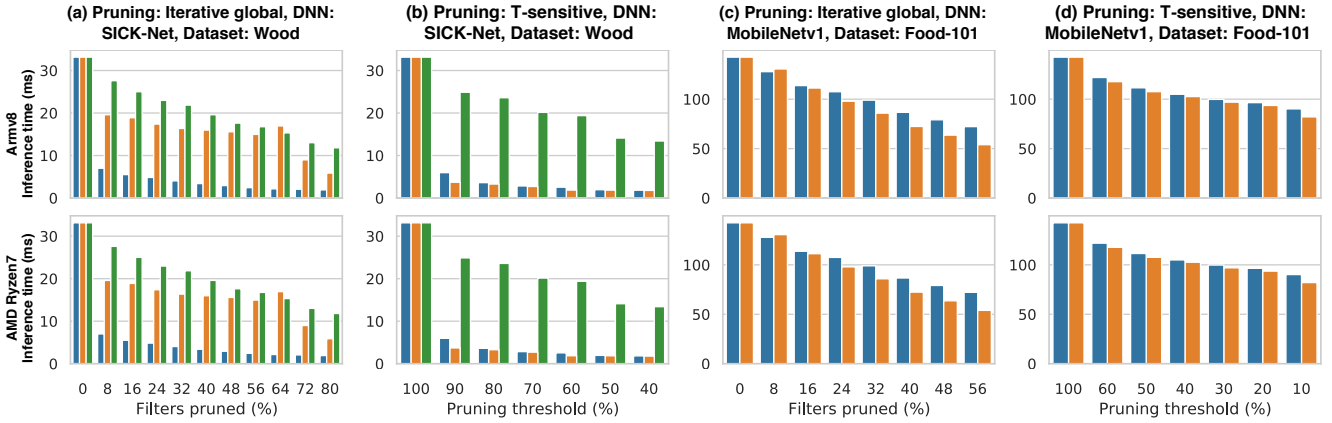

Fig. 7. Evaluating the different strategies and ranking criteria with regard to inference time

TABLE III
ACCURACY LOSS AND SPEEDUP SUMMARY ON ARMV8

Model & dataset	Ranking criterion	Iterative global	T-sensitive
SICK-Net (Wood)	**Mean Weight**	- 0.97% , 4.73x	**- 0.82% , 12.99x**
SICK-Net (Wood)	**L1-norm**	+ 1.19% , 3.69x	- 0.65%, 5.55x - 1.18% , 9.18x
SICK-Net (Wood)	**Taylor**	- 0.20% , 2.15x - 1.05% , 2.55x	- 0.28% , 2.65x
MobileNetv1 (Food-101)	**Mean Weight**	- 0.53% , 1.33x	- 0.49% , 1.45x
MobileNetv1 (Food-101)	**L1-norm**	- 0.57% , 1.25x	**- 0.64%, 1.47x**
MobileNetv1 (ImageNet)	**Mean Weight**	-	**- 4.9%, 1.3x**

TABLE IV
WOOD FLIPPER ACCURACY LOSS AND SPEEDUP SUMMARY ON THE EMBEDDED CAMERA

Ranking criterion	Iterative global	T-Sensitive
Mean Weight	- 0.97% , 7.97x	**- 0.82% , 10.70x**
L1-norm	+ 1.19% , 1.85x	- 0.65% , 5.45x - 1.18% , 6.42x

inference to a real-world setup with only -0.82% accuracy loss. Results are reported in Fig. 8 and Table IV.

VI. CONCLUSION

In this paper, we first motivated the use of filter pruning instead of weight pruning as it relaxes the need for specialized hardware/software needed for handling matrix sparsity. Then, we reviewed different filter ranking criteria and proposed Mean Weight filter ranking criterion which offers both simplicity and efficiency in pruning the less important filters. Further, we combined the ranking criterion with two different filter pruning strategies so-called *iterative global* and *T-sensitive*

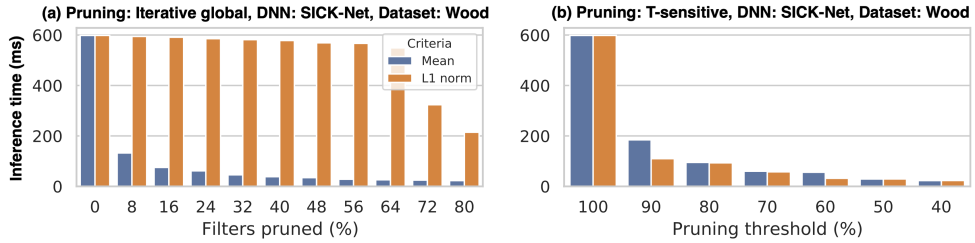

Fig. 8. Evaluating the SICK-Net inference time reduction on the embedded camera

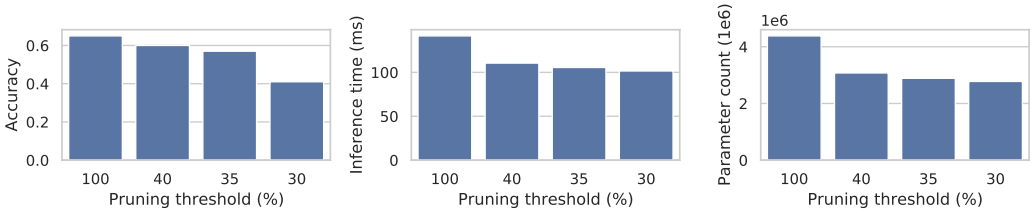

Fig. 9. Evaluating T-Sensitive Mean Weight pruning (TS-MW) on MobileNetv1 trained with the ImageNet dataset, tested on ARMv8

pruning. *T-sensitive* pruning is a proposed variation of layer-sensitive pruning which assigned a pruning threshold to all layers. By introducing a structured filter pruning algorithm, we removed all selected filters and their dependencies from a DNN model, thus speeding up inference and facilitating the deployment of neural networks on low-energy embedded devices equipped with lower-end CPUs. With speedup results of up to 13x for the SICK-net, we demonstrated that our combined ranking and pruning strategy together with the filter removal algorithm outperforms more complex approaches such as the Taylor ranking and is more stable than the simple L1-norm saliency ranking.

An interesting area for further research is the generalisation of the proposed solution to NLP tasks. Transformer architectures are on the rise and replacing filters with attentions could extend their portability to low-energy embedded devices.

Acknowledgment

The research was supported in part by VINNOVA (Sweden's Innovation Agency) through the Trust-E project of the Eureka PENTA and EURIPIDES programmes, and in part by SICK Sensor Intelligence. It is also supported by STINT through the MuRP project.

References

[1] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. USA: Cambridge Univ. Press, 2014.

[2] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, B. C. Van Esesn, A. A. S. Awwal, and V. K. Asari, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[3] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[4] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Coordinated batching and dvfs for dnn inference on gpu accelerators," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 10, pp. 2496–2508, 2022.

[5] H. Bitalebi, V. Geraeinejad, and M. Ebrahimi, "Near llc versus near main memory processing," in *Proceedings of the 14th Workshop on General Purpose Processing Using GPU*, ser. GPGPU '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3530390.3532726

[6] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Batchsizer: Power-performance trade-off for dnn inference," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 819–824. [Online]. Available: https://doi.org/10.1145/3394885.3431535

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014, pp. 609–622.

[8] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[9] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[11] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[12] L. Lu, M. Guo, and S. Renals, "Knowledge distillation for small-footprint highway networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 4820–4824.

[13] X. Wang, Z. Lin, C. Yang, and J. D. Owens, "Accelerating dnn inference with graphblas and the gpu," in *IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–6.

[14] K. Hegde, J. Yu, R. Agrawal, M. Yan, M. Pellauer, and C. Fletcher, "Ucnn: Exploiting computational reuse in deep neural networks via weight repetition," in *ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 674–687.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[16] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[17] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *Proceedings of the European conference on computer vision (ECCV)*, 2014.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[19] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *ASPLOS*, 2020, pp. 907–922.

[20] H. Li, N. Liu, X. Ma, S. Lin, S. Ye, T. Zhang, X. Lin, W. Xu, and Y. Wang, "Admm-based weight pruning for real-time deep learning acceleration on mobile devices," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2019, pp. 501–506.

[21] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, K. Gopalakrishnan, and L. Chang, "Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[22] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-dnn: energy efficient low-precision deep neural networks by compensating quantization errors," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[23] K. Paupamah, S. James, and R. Klein, "Quantisation and pruning for neural network compression and regularisation," in *International SAUPEC/RobMech/PRASA Conference*. IEEE, 2020, pp. 1–6.

[24] S. M. Nabavinejad, H. Hafez-Kolahi, and S. Reda, "Coordinated dvfs and precision control for deep neural networks," *IEEE Computer Architecture Letters*, vol. 18, no. 2, pp. 136–140, 2019.

[25] U. Kulkarni, S. Meena, S. V. Gurlahosur, and G. Bhogar, "Quantization friendly mobilenet (qf-mobilenet) architecture for vision based applications on embedded platforms," *Neural Networks*, vol. 136, pp. 28–39, 2021.

[26] M. Wang, W. Zhou, Q. Tian, and H. Li, "Deep graph convolutional quantization networks for image retrieval," *IEEE Transactions on Multimedia*, 2022.

[27] L. Mauch and B. Yang, "A novel layerwise pruning method for model reduction of fully connected deep neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 2382–2386.

[28] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, and Y. Wang, "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 184–199.

[29] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing dnn pruning to the underlying hardware parallelism," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 548–560, 2017.

[30] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4340–4349.

[31] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[32] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[33] A. See, M.-T. Luong, and C. D. Manning, "Compression of neural machine translation models via pruning," *arXiv preprint arXiv:1606.09274*, 2016.

[34] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[35] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[36] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 815–832.

[37] Y. Lecun, J. Denker, and S. Solla, "Optimal brain damage," vol. 2, 01 1989, pp. 598–605.

[38] B. Hassibi, D. G.Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," 02 1993, pp. 293 – 299 vol.1.