HAL
open science

# CNN weight sharing based on a fast accuracy estimation metric

Etienne Dupuis, David Novo, Ian O'Connor, Alberto Bosio

## To cite this version:

Etienne Dupuis, David Novo, Ian O'Connor, Alberto Bosio. CNN weight sharing based on a fast accuracy estimation metric. Microelectronics Reliability, 2021, 122, pp.#114148. 10.1016/j.microrel.2021.114148 . hal-03257748

**HAL Id: hal-03257748**

**https://hal.science/hal-03257748v1**

Submitted on 24 Nov 2021

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# CNN Weight Sharing Based on a Fast Accuracy Estimation Metric

Etienne Dupuis[1], David Novo[2], Ian O'Connor[1], and Alberto Bosio[1]

[1] *Univ Lyon, Ecole Centrale de Lyon, CNRS, INSA Lyon, Université Claude Bernard Lyon 1, CPE Lyon, CNRS, INL, UMR5270, 69130 Ecully, France*

[2] *LIRMM, Université de Montpellier, CNRS, France*

{*etienne.dupuis, ian.oconnor, alberto.bosio*}*@ec-lyon.fr, and david.novo@lirmm.fr*

**Abstract**

The computational workload involved in CNNs is typically out of reach for low-power embedded devices. The Approximate Computing paradigm can be exploited to reduce the CNN complexity since it improves performances and energy-efficiency by relaxing the need for fully accurate operations. In this work, we target weight-sharing as an approximate technique to reduce the memory footprint of a CNN. More in detail, we prove that optimizing the number of shared weights can enable significant network memory compression without noticeable accuracy loss without retraining or fine-tuning steps. However, we observe that the exploration time can easily explode in state-of-the-art CNNs. We thus propose the use of a fast accuracy estimation metric to guide the design space exploration and drastically reduce the exploration time up to 12×. Compared with state-of-the-art CNN approximation methods, we obtained more than 4× compression on GoogleNet on the ImageNet dataset with less than 1% accuracy loss in less than 5 hours and without any retraining step.

*Keywords:* Convolutional Neural Networks, Approximate Computing, Model Compression, Weight Sharing, Design Space Exploration, Embedded System

## 1. Introduction

Deep Neural Networks (DNNs) are currently one of the most widely used predictive tools in the field of machine learning. They are used today in various applications such as object recognition, drug discovery, and natural language processing, as well as safety-critical applications like autonomous driving. Recent DNNs are articulated around the use of multiple chained convolution layers, called Convolutional Neural Networks (CNNs), using a dot product between input and filter [1].

Unfortunately, the computational workload and memory footprint of CNNs are often out of reach for low-power embedded devices [2]. For example, the amazing performance of AlphaGo [3] required a supercomputer composed of 2,000 CPUs and 250 GPUs for a total of about 600 KW of power consumption, while the human brain of a go player requires about 20W.

Novel computing paradigms and emerging technologies are under investigation to make CNNs available for low-power devices. Among them, the Approximate Computing (AxC) paradigm leverages the inherent error resilience of CNNs to improve energy efficiency, by relaxing the need for fully accurate operations. CNNs have a high degree of redundancy in terms of their architecture and parameters, this redundancy is not always necessary for accurate prediction. This observation has paved the way for several highly recognized approximation techniques [2] such as pruning[4], quantization[5], low-rank factorization[6], and weight-sharing[4].

In this work, we intend to investigate the opportunity of Weight-Sharing (WS). How WS can help in reducing the memory footprint of CNNs, what is the best granularity (i.e., network-, layer-, channel- or kernel-wise) to apply it and finally develop a framework to allow design space exploration for approximating CNNs. The main contributions of this paper can be summarized as follows:

1. We present the WS and its implication in CNN compression;
2. We present a granularity analysis to identify the best opportunity for WS application;
3. We propose the use of a heuristic design exploration method based on a fast accuracy estimation metric for the efficient CNN compression;
4. Thanks to the proposed metric, we were able to implement a multi-objective design space exploration to identify trade-offs between compression and accuracy.

The rest of the paper is organized as follows: Section 2 introduces the concepts and state-of-the-art weight-sharing AxC technique. Section 3 describes the proposed weight-sharing exploration. Section 4 introduces the adopted heuristic while Section 4.1 presents the obtained results. Finally, Section 5 concludes the paper.

## 2. Weight-Sharing Approximation Related Works

A wide range of approximation techniques targeting CNN has been proposed so far in the literature. Among them, we distinguish between three different classes. The first class, known as *structure refinement*, includes the techniques that modify the computational structure (*i.e.,* the network layers and their parameters) of a given CNN model. Some notable examples include the knowledge distillation [7] that uses the model as a teacher to train smaller models (i.e., students); compact architectures [8] where layers are transformed into more hardware-friendly ones; and the Neural Architecture Search [9] that identify the most appropriate CNN architecture satisfying a given set of constraints. The second class, known as *data refinement*, focuses on optimizing the finite precision data representation of the given CNN model without modifying its computational structure. Notable examples are the weight pruning [10] aiming at setting less important weights to zero; the quantization [11] aiming at changing the data types of the weights and intermediate results to more efficient representations (i.e., lower bit-width); and the weight-sharing [4] aiming at replacing clusters of similar weights with a common (shared) value. The third class, named *operator refinement* includes all the techniques aiming at approximating the arithmetic operators used by the given CNN model computation structure [12].

In this paper, we focus on the Weight-Sharing (WS) technique, which belongs to the second class and targets a reduction in the number of different weight values to achieve a significant reduction of the CNN memory footprint. WS identifies **clusters of weights** and replaces all the weights belonging to the same cluster by the cluster's centroid. Let us resort to the example of Figure 1 where starting from an initial 5x5 convolutional kernel matrix, clustering is applied. Usually, clusters are identified by using the $K$-means [13] algorithm as described in DeepCompression [4].

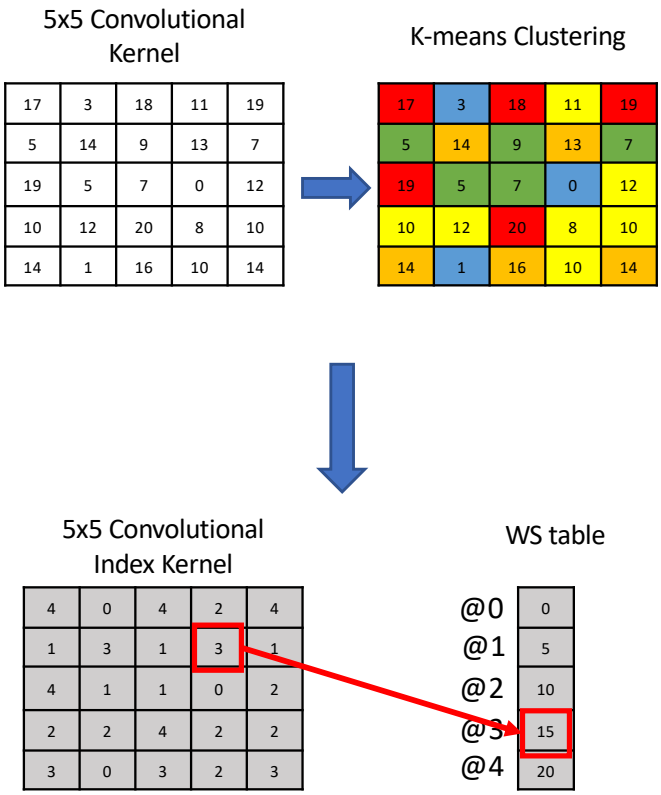



Figure 1: Weight-Sharing example for a 5x5 matrix with weight values grouped in 5 different clusters

From the original matrix in Figure 1, $K$-means is applied to identify 5 different clusters (i.e., depicted through colors) corresponding to cluster centroids (0, 5, 10, 15, and 20). These values are stored in the so-called *WS table*. In our example, we have weight values represented as integer values stored in 8-bit variables. The 5x5 convolutional kernel matrix will thus require 200 bits of memory (5x5x8 bits). On the other hand, the WS table requires 40 bits of memory (each centroid value is still represented as an 8-bit integer). However, in the kernel matrix, we will store only the centroid **index** (from 0 to 4) instead of its value. We thus only need 3 bits per index, leading to 75 bits of storage.

To sum up, the memory footprint will be 115 bits (40 bit + 75 bit) much lower compared to the original 200 bit. Memory footprint reduction is thus the main advantage of WS, however, it is important to mention that reducing memory usage also implies less energy consumption because fewer data have to be transferred from/to memory (i.e., this is the well-known architectural power-wall problem as discussed in [14]).

More formally, we can define the memory footprint reduction as follows. The size of the weight matrix can be reduced from $B$ bits for each value to $\lceil log_2(K) \rceil$ with $K$ being the number of clusters.

The size of the memory footprint then becomes $W \times \lceil log_2(K) \rceil + K \times B$ instead of $W \times B$, where $W$ is the number of weights. We can thus define the Compression Ratio (CR) as

$$CR = \frac{W \times B}{W \times \lceil \log_2(K) \rceil + K \times B}. \tag{1}$$

In the literature, there are many practical applications of WS. Existing solutions differ from the abstraction layer, the use of combined techniques, and the need for retraining steps after the approximation to compensate for the accuracy loss. A simple WS application is the vector quantization [15], where a model-level WS approach based on $K$-means is applied to fully connected layers of a trained network. Hashed net[16] proposes to randomly group weights into buckets sharing the same value before the training step. A more efficient method is presented in DeepCompression [4], it achieves impressive results due to the use of several model-level AxC techniques as pruning, WS, and coding. Deep $K$-means [17] uses regularization terms to encourage weights to concentrate at retraining time. Moreover, it also targets the architecture level by exploiting data re-shaping techniques optimized for higher data reuse with row-stationary dataflow[18] at inference time. Soft weight-sharing[19] uses a simple (re)training step with a regularizer to efficiently compress the network using pruning and weight-sharing. It is also possible to reduce complexity by encoding both weights and features maps. Then, at the architecture-level, it computes multiplications using a lookup table. Both LookNN[20] and Quantized CNNs[21] are based on this concept and achieve good results at inference time, particularly if energy efficiency is taken into account.

Although CNN retraining after approximation allows accuracy loss recovery, there are many reasons to develop design methods that can largely benefit from clustering without requiring any further retraining. For example, retraining may require access to the full training data which may not always be possible due to the size of the entire dataset or to legal and privacy compliance reasons. Furthermore, training is a long and compute-intensive step, and thus, it is costly and can delay the CNN time-to-market.

In our previous works [22, 23], we developed a simple, but efficient, cluster identification method. The method was first based on the identification of the layers' sensitivity to approximation. Then, a greedy algorithm aims at identifying, for each layer, the best number of clusters. Every time a solution is generated, the whole CNN model is evaluated (i.e., scored) to quantify the impact of the WS on the inference accuracy. Due to the high computational cost of CNN scoring, the previous approach could not scale to the size of state-of-the-art CNN models. Instead, in this work, we propose a new exploration method based on a fast accuracy estimation metric to drastically reduce the number of scoring, and thus, the overall computational time. Moreover, we also extend our explo-

ration framework to optimize the WS application for maximizing the compression rate (Eq. 1) and, at the same time, minimizing the impact on the CNN accuracy.

## 3. Weight-Sharing Exploration

This section presents the approach used for exploring the opportunities of approximating CNNs using weight-sharing. We first present the granularity analysis used to identify the best WS application, then the proposed design space exploration.

In the previous section, we show that the number of clusters $K$ plays a crucial role in terms of CR and accuracy. The term "accuracy" is the capability of the CNN to correctly behave. For example, if the CNN is supposed to classify images then, its accuracy is quantified as the ratio between correct classifications over the number of processed inputs. When the correct label corresponds to the first ranked output, we refer to the term "top-1" accuracy. When correct classification corresponds to the first five ranked outputs, we refer to the term "top-5" accuracy.

To show how can vary the accuracy of a CNN w.r.t. $K$ (i.e., in our case $K$ is the degree of approximation), we resort to the chart depicted in Figure 2. The target CNN is the MobileNet V2 [24] trained using the ImageNet database [25]. The chart only reports the first layer. On the X-axis, we plot different $K$ (from 0 to 120) and the Y-axis reports the achieved accuracy. The accuracy strictly depends on $K$. We can easily note that around $K = 38$ the top-1 accuracy is really close to the reference one (the black bar). It is really important to mention that the reference layer has 864 weights. Each weight is stored into a 32-bit floating-point variable leading to a memory footprint of 27,648 bits. Now if we keep 38 clusters, we need 5,184 bits for storing the 864 cluster indexes (6 bits each) plus 1,216 bits for storing the 38 centroids values. The achieved CR is about **4× without any accuracy loss. The CR changes depending on the bit-width of the CNN weights. In the above example, if 16-bit data types are used, the CR will be 2×, while for 8-bit the CR will be 1.25×. In this paper, we only consider CNN models where weights are represented using a 32-bit floating-point variable to fairly compare our approach with the state of the art.**

This example confirms once again the potential of the WS technique. However, to approximate a CNN using WS, one could try to act at different granularity here listed from coarser to finer grain:

- *Layer*: all the weights of a layer are clustered together;
- *Channel*: all the weights of a filter are clustered together;
- *Kernel*: all the weights of a channel corresponding to a specific input channel are clustered together.

Figure 3 shows a graphical example of the above granularity level for a given convolutional layer. Each of the above granularity levels allows to achieve different CRs (eq.1) and accuracy loss w.r.t. the reference (non approximated) CNN. Figure 4 plots the trade-off obtained at different levels of granularity. These experiments were carried out on LeNet. Each solution corresponds to a network with a certain number of clusters. The results clearly show that the "kernel" and "filter" granularity does not lead

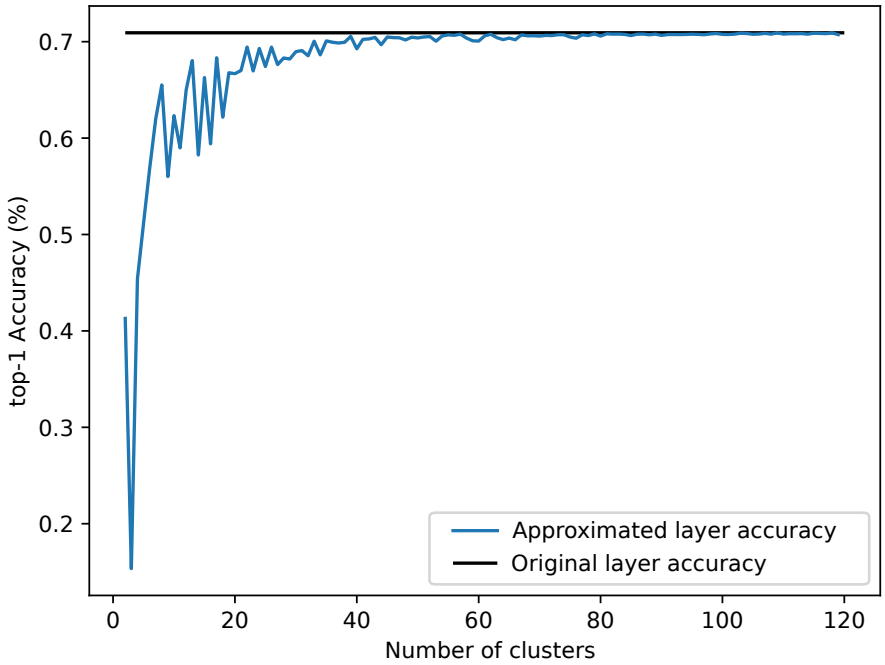



Figure 2: Achieved top-1 accuracy, obtained by varying the number of shared values for the first layer of MobileNet V2[26]

to a good tradeoff between compression and accuracy. On the other hand, the "layer" granularity allows up to $3/times$ higher compression ratios for the same accuracy loss. For the remainder of the paper, we thus exploit the **layer granularity** in our weight sharing technique. The complete description of this analysis is provided in [27].

In the next subsection, we will present our design space exploration for approximating CNNs.

### *3.1. Design Space Exploration for AxCNNs*

Figure 5 sketches the overall flow of the proposed design space exploration for Approximate CNNs (AxCNNs). The starting point is a trained CNN (i.e., the Reference CNN) with the Dataset (DS) used for training. We got trained CNNs and DS from the open-source framework ONNX [28] model zoo. It is important to mention that our AxCNNs framework is fully compatible with any CNNs described in ONNX format. Therefore, any ONNX-compatible framework can be used for training. The reference CNN (Ref CNN in Figure 5) is then "scored" on the Testing Dataset to measure its Accuracy. The CNN scoring is implemented by using MXNET[29].

Algorithm 1 shows the pseudo-code of the approximation framework implementing the WS. It takes as input the Reference CNN, its accuracy, the number of clusters $k$

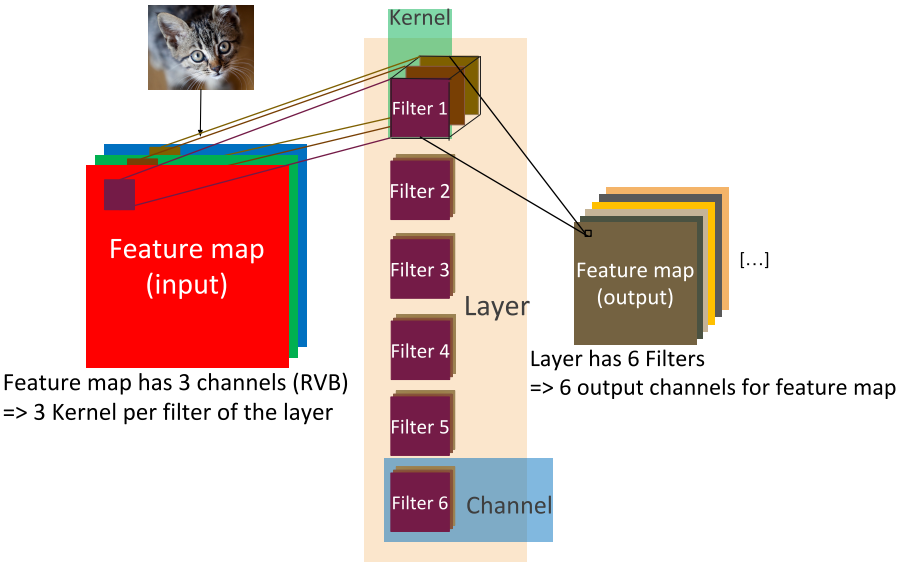



Figure 3: Different granularities for the weight sharing. Weights can be shared at the level of the layer, the channel or the kernel.

expressed as a range (min to max), and the testing Dataset. The algorithm processes one CNN layer at a time. For each layer, the $K$-means algorithm is executed by varying the number of clusters $K$ from $minCluster$ to $maxCluster$. The $K$-means is implemented using the libKMCUDA [30], a CUDA-based $K$-means library. Each $K$-means execution provides an approximation of the CNN. The latter is scored to determine the accuracy loss. At the end of the clustering loop, the best $K$ is determined. Before moving to the next layer, the current CNN layer is approximated through weight sharing based on the best $k$. At the end of the algorithm, we compute the final accuracy of the approximate CNN and the compression rate. The approximation framework is thus based on a greedy algorithm with a complexity equal to $O(N \times |K_{range}|)$, where $N$ is the number of layers composing the CNN and $|K_{range}|$ is $maxCluster - minCluster$.

### *3.2. Experimental Setup*

In this section, we present the experimental setup used for the validation of the AxCNNs flow of Figure 5. First of all, we targeted three state-of-the-art image classifier CNNs trained and tested using the ImageNet[31] dataset working with 224x224 images with 1,000 class output. CNNs trained model came from the ONNX model zoo.

Table 1 presents the details of each CNN in terms of computing complexity measured as the number of Multiply ACcumulate operations (#MAC), the number of layers, memory footprint representing the size of weights in MegaBytes (each weight is stored in a 32bit floating-point format) and the top-1 accuracy. Each selected CNN implements a specific trade-off between accuracy versus complexity. We considered the following three targets:

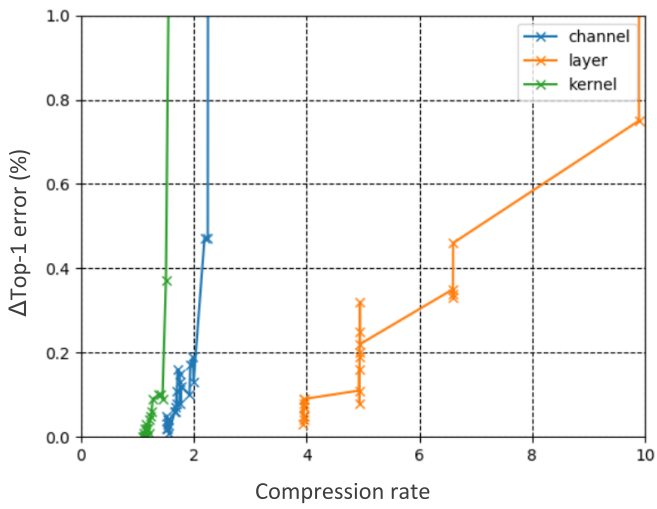



Figure 4: Top-1 accuracy loss and memory compression over varying numbers of clusters at different sharing granularity citeDupuis2020autoWS

- **Accuracy:** This target aims to maximize the accuracy of the CNN thus leading to the highest computational complexity (almost 2 GOPS required for inference) and memory footprint (almost 50 megabytes are required to store the weights of the model) (ResNet18 v2 [26]);
- **Memory:** This target aims to minimize the memory footprint (less than 2 megabytes) of the CNN thus leading to the lowest accuracy and low computational complexity (less than 0.5 GOPS) (SqueezeNet v1.1 [32]);
- **Energy efficiency:** This target aims to minimize the energy required during the inference thus leading to the lowest computational complexity(less than 0.5 GOPS) medium memory footprint (14 megabytes) and good accuracy (MobileNet v2 [24]).

We performed experimental validation of Algorithm 1. All the experiments were executed on a server equipped with a single NVIDIA TESLA V100 GPU and the latest CUDA version of MXNET[29] for the CNN scoring. The results are presented in Table 2.

For each Network, we report the clustering range ($K_{range}$ from min to max), the Accuracy Loss (AL), the Accuracy Ratio (AR), the Compression Ratio (CR), the number of iterations (#It), the time required to execute all the $Kmeans()$ functions, the time required to execute all the $score()$ functions and in the last column the total time (i.e., the sum between the Cl. and Sc. time). Before analyzing the results, let us briefly describe the overall set-up. First of all, the clustering range has been empirically determined for each network in the same way as the example reported in Figure 2. That means, that for each CNN layer, we execute the $k$-means algorithm until we reach the reference accuracy. This explains why different CNNs have different $K_{range}$s. The Accuracy Loss

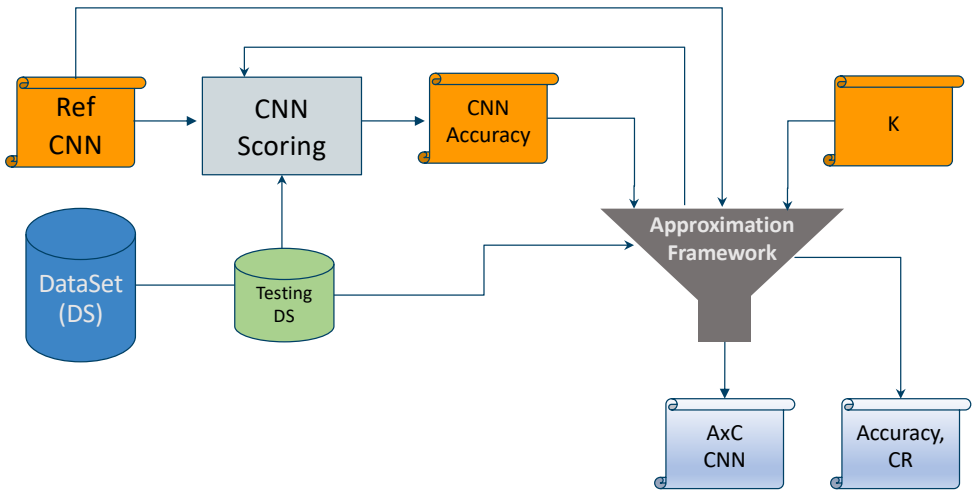



Figure 5: Overall flow of the proposed method

is computed as the difference between Reference accuracy and Approximate accuracy. The Accuracy Ratio is computed as the ratio between the Approximate accuracy over the Reference accuracy. The CR is computed through equation 1. The number of iterations is simply obtained as the number of elements of the $K_{range}$ times the number of CNN layers.

The first comment on the results is about the important CR that has been obtained. We were able to successfully compress each of the reference networks by up to 5× with a very small accuracy loss (less than 1.5%). The interesting point is that the CR depends mostly of the input range for the number of shared values and is rather constant for the different CNNs shown in Table 2. In fact, even for SqueezeNet v1.1, which is already designed to have a very low memory footprint, we achieve a 5× CR. In other words, even if the reference CNN has been designed to be efficient in terms of the memory footprint, we were able to further reduce it. The second comment is about the achieved accuracy ratio. We clearly paid the price of the approximation, however, the AR is always greater than 97%. We can thus safely state that the accuracy loss due to the approximation is negligible (up to 1.43%).

The last comment is about the required execution time. The cost of Algorithm 1 is dominated by more than 90% by the **scoring time** (i.e., the time to execute the inference of the test dataset to measure approximated CNN top-1 accuracy). Hower as the complexity of the algorithm is $\emptyset(N * |K_{range}|)$, with $N$ being the number of layers and $K_{range}$ the number of solutions evaluated for each layer. This limited scalability prohibits the use of Algorithm 1 for larger networks. In the next section, we thus present an indirect way to estimate the accuracy of CNN without running the inference.

**Algorithm 1:** Design Space Exploration

**Input:** $CNN$, $minCluster$, $maxCluster$, $RefAccuracy$, $DS$, $inertiaFiltreRatio$
**Output:** $AxCNN$, $AxCAccuracy$, $CR$

```
for layer in getLayers(CNN) do
    modelSet ← [];
    i ← 0;
                // Compute different version of the layer with different number of cluster
    for K ← minCluster to maxCluster do
        modelSet[i] ← Kmeans(CNN[layer], K);
        i ← i + 1;
    end for
                                                    // Score versions and select the best
    bestAccuracyLoss ← None;
    for j ← 0 to i do
        accLoss ← (RefAccuracy − score(modelSet[j], DS));
        if accLoss < bestAccuracyLoss then
            bestAccuracyLoss ← accLoss;
            bestK ← K;
        end if
    end for
end for
```

Table 1: Characteristics of the selected CNNs

| **Network** | **Target** | **#MAC** | **#layer** | **Memory** | **Top-1 Acc. (%)** |
|---|---|---|---|---|---|
| ResNet18v2 | Accurate | 1,82x10$^9$ | 21 | 48Mb | 69,7 |
| SqueezeNetv1.1 | Compressed | 0,36x10$^9$ | 26 | 1,6Mb | 57.5 |
| MobileNetv2 | Efficient | 0,32x10$^9$ | 54 | 14Mb | 70,8 |

## 4. Sensitivity Metric

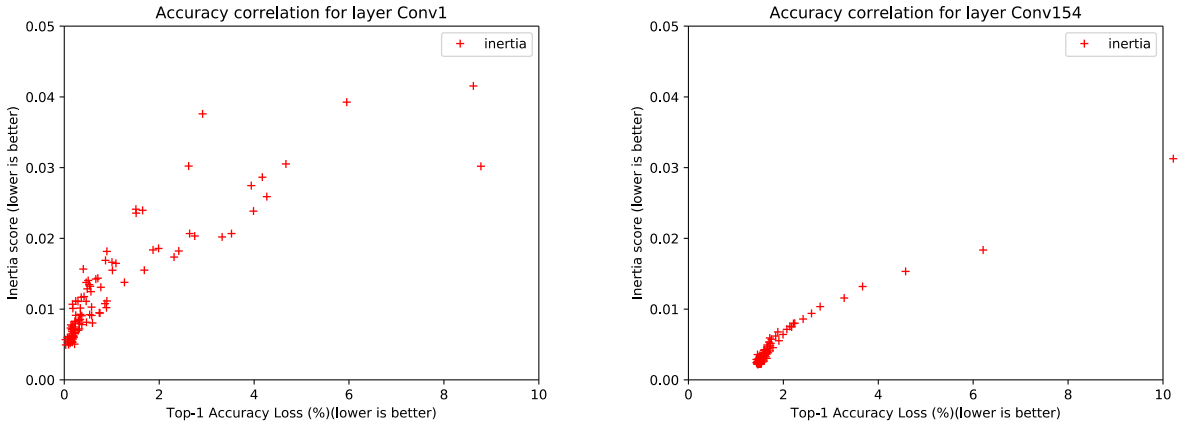
The key to identifying an alternative to the need of scoring each CNN approximation relies on the analysis of the $k$-means algorithm. Indeed, $k$-means clusters data by trying to separate samples in $k$ groups of equal variance, minimizing a criterion known as the **inertia** or within-cluster sum-of-squares [33] defined by the following equation:

$$Inertia = \sum_{i=0}^{N} \min_{\mu_j \in C} (||x_i - \mu_j||^2) \tag{2}$$

with $N$ is the number of samples, $x_i$ is the ith sample, $K$ is the number of clusters, $C$ is the clusters and $\mu_j$ are the centroids. Inertia can be described as a measure of how coherent are the clusters, and the lower the inertia the better the coherence. We thus argued that the inertia could be a good estimation of the accuracy of the approximated CNN, the lower the inertia the higher the accuracy.

Table 2: AxCNNs DSE Results obtained with Algorithm 1

| Network | $K_{range}$ | AL | AR | CR | #It | Cl. time | Sc. time | Tot. Time |
|---|---|---|---|---|---|---|---|---|
| ResNet18v2 | 40-80 | 0.22% | 99.68% | 5.28 | 840 | 02:19:51 | 13:50:40 | 16:10:31 |
| SqueezeNetv1.1 | 40-80 | 0.53% | 99.05% | 5.17 | 1040 | 00:06:51 | 12:58:38 | 13:05:30 |
| MobileNetv2 | 2-120 | 1.43% | 97.98% | 4.85 | 6372 | 01:12:54 | 84:49:10 | 86:02:05 |




(a) Layer 1 of MobileNet V2

(b) Layer 154 of MobileNet V2

Figure 6

Correlation between the inertia and the top-1 accuracy for two distinct layers of MobileNet V2[24]. Having different number of weights (i.e., 864 for layer 1, 1,280,000 for layer 154) results in different correlation profiles.

Figures 6 report two examples of how good is the inertia as accuracy loss estimation by showing the inertia and the accuracy loss obtained after clustering layers with $K_{range} = [2, 120]$. Both the examples are related to MobileNet V2 [24]. In particular, we reported layer 1 of MobileNet (Figure 6a) and layer 154 (Figure 6b). These layers have been chosen because they have a different complexity expressed in terms of weights: 864 for layer 1, 1,280,000 for layer 154. Both graphs plot the CNN accuracy loss (obtained through CNN scoring) w.r.t the inertia. Despite the differences, the correlation in the small layer is not as strong as in the large layer, both cases confirm that lower inertia leads to higher accuracy.

As the inertia proved to be a good estimation of the approximated CNN accuracy, we modified Algorithm 1 to use it. In Algorithm 1, after each $k$-means application, the approximated CNN has been scored. In Algorithm 2, the idea is to use inertia to filter a subset of the *best* approximate candidates to be scored. The number of selected candidates depends on the $inertiaFiltreRatio$ parameter. The complexity of Algorithm 2 is thus $O(N \times inertiaFiltreRatio)$.

### *4.1. Experiments*

The experiments have been carried out under the conditions described in Section 3.2. Table 3 presents the obtained results. The table format is the same as Table 2 except

**Algorithm 2:** Design Space Exploration with Inertia Filtering

**Input:** $CNN$, $minCluster$, $maxCluster$, $RefAccuracy$, $DS$, $inertiaFiltreRatio$

**Output:** $AxCNN$, $AxCAccuracy$, $CR$

**for** layer in $getLayers(CNN)$ **do**
  $modelSet \leftarrow []$;
  $i \leftarrow 0$;
  // Compute different version of the layer with different number of cluster
  **for** $K \leftarrow minCluster$ **to** $maxCluster$ **do**
    $modelSet[i] \leftarrow Kmeans(CNN[layer], K)$;
    $i \leftarrow i + 1$;
  **end for**
  // Filter the versions of the layers based on their inertia
  $sortByInertia(modelSet)$;
  $fiteredModelSet \leftarrow takeNFirst(modelSet, inertiaFiltreRatio * i)$;
  // Score remaining versions and select the best
  $bestAccuracyLoss \leftarrow None$;
  **for** $j \leftarrow 0$ **to** $i * inertiaFiltreRatio$ **do**
    $accLoss \leftarrow (RefAccuracy - score(fiteredModelSet[j], DS))$;
    **if** $accLoss < bestAccuracyLoss$ **then**
      $bestAccuracyLoss \leftarrow accLoss$;
      $bestK \leftarrow K$;
    **end if**
  **end for**
**end for**

Table 3: AxCNNs DSE-inertia Results obtained with Algorithm 2

| Network | Filt. (%) | AL | AR | CR | Cl. time | Sc. time | Tot. Time |
|---|---|---|---|---|---|---|---|
| ResNet18v2 | 5 | 0.47% | 99.33% | 4.93 | 02:20:33 | 00:59:34 | 03:20:07 |
| | 15 | 0.23% | 99.67% | 4.95 | 02:26:38 | 02:27:16 | 04:53:54 |
| | 33 | 0.23% | 99.67% | 4.94 | 02:30:03 | 04:48:00 | 07:18:04 |
| SqueezeNetv1.1 | 5 | 0.57% | 98.97% | 4.77 | 00:06:51 | 01:01:39 | 04:53:03 |
| | 15 | 0.55% | 98.01% | 4.8 | 00:06:51 | 02:15:48 | 02:22:39 |
| | 33 | 0.49% | 99.12% | 4.73 | 00:06:52 | 04:46:11 | 01:08:31 |
| MobileNetv2 | 5 | 2.78% | 96.08% | 4.54 | 00:59:12 | 06:16:21 | 07:15:33 |
| | 15 | 1.58% | 97.77% | 4.55 | 01:00:19 | 18:49:56 | 19:50:15 |
| | 33 | 1.13% | 98.41% | 4.55 | 01:07:02 | 40:57:55 | 42:04:57 |

for the column "Filtering". Filtering value corresponds to $inertiaFiltreRatio$ parameter of Algorithm 2. In our set of experiments, we tested the following values: 5%, 15%, and 33%. Once again, filtering is the percentage of approximate CNN versions that will be scored. When Filtering is equal to 100, Algorithm 2 has the same complexity and provides the same results as Algorithm 1.

From the results in Table 3, we can see that at highest values of Filtering generally results in higher values of AR. This improvement comes at the cost of a higher optimization time due to a larger number of candidates passing through the inertia filter to the costly scoring step. Like in Algorithm 1, the scoring time dominates the clustering time and the CR is between 4.5 and 5, due to the same input range. However, using a small filtering value (i.e., 5%, 15%) significantly reduces the overall exploration time with a limited impact on the compression rate and accuracy loss.

It is particularly interesting to compare the case of MobileNet. Using inertia (Algorithm 2) with the Filtering equal to 33% the achieved AR is 98%, better than the result obtained with Algorithm 1 (see Table 2). However, the most relevant impact of the proposed heuristic is on the overall exploration time. In this case, we reduce it by 50% (i.e., 40 hours instead of 80). When reducing the Filtering value to 5% the AR is just 1% less than Algorithm 1 but here the execution time reduction by $12/times$ (i.e., 7 hours instead of 80). Similar results were obtained for the other CNNs. Looking at CR, we did not observe a significant difference with the variation of the filtering value. Achieved CRs are slightly less than those of Algorithm 1 but still significant. These results prove the efficiency of using inertia as a quick and effective metric for determining the impact of weight-sharing on CNN.

### *4.2. Comparison with prior work*

We compared our results with the two state-of-the-art methods. The first one, named Deep K-means [17], adds the K-means objective metric (i.e., inertia as explained at the beginning of Section 4) to the optimization objective used during training. Applying a weight-sharing aware training brings weights closer to the cluster centroids, which reduces the error introduced in the subsequent weight sharing step. They used a fixed *cluster rate* defined as the number of cluster $K$ divided by the number of weights of the layer $N$. On the other hand, the second method, named DP-Net [34], uses dynamic programming over the K-means algorithm to optimize the weight sharing. This brings a performance improvement as well as a guarantee of the convergence of the clustering compared to the highly-variable K-means complexity. They use a fixed and homogeneous number of clusters for the network. Both papers [17, 34], reported results on GoogleNet [35] with the Imagenet dataset. We thus apply our exploration algorithm on the same CNN model and dataset. More in detail, we use the available GoogleNet Pytorch implementation achieving 69.78% top-1 accuracy on the Imagenet dataset with standard preprocessing including normalization,re-scaling, and center-crop. We used an inertia filter value of 15%, which shows the best trade-off between computation time and result. Finally, it is important to stress the fact that a fair comparison is only possible with Deep K-means since it is the only approach that can be used without retraining. Table 4 shows the results of the comparison. The first column depicts the method, the second specifies if the retraining phase is applied or not. Finally, the third and fourth columns report the CR and AL respectively. As already pointed out, our approach does not use retraining, Deep K-means can use it or not, while DP-Net always uses retraining.Even though that

Table 4: Comparing our result with other weight sharing techniques on GoogleNet[35] using Algorithm 2

| **Method** | **Retraining** | **CR** | **AL** |
|---|---|---|---|
| Deep K-means(2018)[17] | No | 1.5 | 1.22 |
| | No | 2 | 3.7 |
| | No | 3 | 13.72 |
| | No | 4 | 48.95 |
| | Yes | 1.5 | 0.26 |
| | Yes | 2 | 0.17 |
| | Yes | 3 | 0.36 |
| | Yes | 4 | 1.95 |
| DP-Net (2020)[34] | Yes | 7 | -0.3 |
| | Yes | 10 | 1.56 |
| Our method | No | 4.55 | 0.83 |

our method is the only one that does not use retraining, our results fall in between the two reference methods. Interestingly, we achieve a higher CR than Deep K-means at a lower accuracy loss. It is important to mention that when Deep K-means does not use retraining, our approach simply works better. For similar CR (about 4x) we achieved 0.83 of Accuracy Loss while Deep K-means got 49% of AL. The DP-Net, however, finds a more efficient solution by better leveraging the retraining phase. Results of 4 showed that our approach is comparable with state-of-the-art, even if no retraining is applied. Unfortunately, no data are reporting the execution time of Deep K-means and DP-Net for further comparison.However, the execution time of our algorithm took less than 5h on our NVIDIA Tesla V100 GPU. On the same machine with the same CNN model and dataset, 5h corresponds to the time required to run about 5 epochs of retraining. We can thus conclude that our exploration time is far less than the required retraining step of Deep K-means and DP-Net.

### *4.3. From single objective to multi-objective*

Results shown in Table 3 reflect the behavior of Algorithm 2. The objective of Algorithm 2 is to minimize the accuracy loss without considering other constraints. However, It would be interesting to investigate different trade-offs between Accuracy Loss and Compression Rate. In other words, the user can be interested to accept higher AL for higher CR. This means that we have to consider more objectives to optimize our algorithm. We thus modified Algorithm 2 to have two objectives: (i) minimize accuracy loss and (ii) maximize compression rate.

To achieve this modification, we start by enlarging the number of candidates selected during each layer exploration. Instead of selecting only the candidate showing the lowest AL, we explore the AL vs. CR trade-off and select Pareto optimal candidates. This implies enlarging the complexity of the exploration for the next layers. Indeed, we need to run the exploration for each selected candidates of the previous layer, raising the complexity of the layer-wise exploration of layer $i$ to $O(|S_{i-1}| \times |K_{range}|)$, where $S_{i-1}$ is the number of selected candidates. The resulting complexity of the multi-objective exploration for the whole network is therefore $O(\prod_{i=1}^{N} |S_{i-1|} \times |K_{range}|)$. In the worst

case (i.e., every candidate is Pareto optimal from the first layer to the last, meaning $|S_{i-1}| = |K_{range}|$), the complexity becomes $O(\prod_{i=1}^{N} |K_{range}|^2) \longrightarrow O(|K_{range}|^{2N})$. For example, for a $N = 2$ layer CNN, given a $K_{range} = [2, 128]$, and considering that the clustering step takes 1 second, it will take more than 56 years. To avoid this explosion of the number of candidates, we need to restrain their count between each iteration, we can see in Equation 1 that in the memory footprint of the approximated model given by the formula $W * \lceil log_2(K) \rceil + K * B$, the most important term is the first one $W * \lceil log_2(K) \rceil + K$, and this is because $W$ tends to be up to several million for the largest network. This is why instead of selecting candidates that are Pareto optimal in terms of their AC and CR, we use their AC and number of bits required to store the indexes of the shared values ($n_{bitindex}$), which is given by the formula $n_{bitindex} = \lceil log_2(K) \rceil$. Considering that $max(S_{i-1}) = \lceil log_2(|K_{range}|) \rceil$, the complexity of the exploration is reduced to

$$O(N * \lceil log_2(|K_{range}|) \rceil * |K_{range}|), \tag{3}$$

which our experiments show affordable even for a large network. The resulting multi-objective hierarchical exploration algorithm is detailed in Algorithm 3

**Algorithm 3:** Design Space Exploration with Multi-Objective (Accuracy loss & compression rate)

**Input:** $CNN$, $minCluster$, $maxCluster$, $RefAccuracy$, $DS$, $inertiaFiltreRatio$; $K_range$
**Output:** $optimalAxCNNPopulation$,

```
population ← [CNN]
for layer in getLayers(CNN) do
    newPopulation ← []
    i ← 0   // Compute different version of the layer with different number of cluster
    for candidateinpopulation do
        for K ← minCluster to maxCluster do
            newPopulation[i] ← Kmeans(CNN[layer], K);
            i ← i + 1;
        end for
    end for                    // Filter the versions of the layers based on their inertia
    sortByInertia(newPopulation);
    newPopulation ← takeNFirst(newPopulation, inertiaFiltreRatio * i);
                                                          // Score filtered candidates
    score = []
    for j ← 0 to i * inertiaFiltreRatio do
        score[j] ← (RefAccuracy − score(newPopulation[j], DS));
    end for
        // filter Pareto optimal candidates based on their accuracy loss and n_bitindex
    newPopulation ← paretoFilter(newPopulation)
    population ← newPopulation
end forOptimalAxCNNPopulation ← population
```

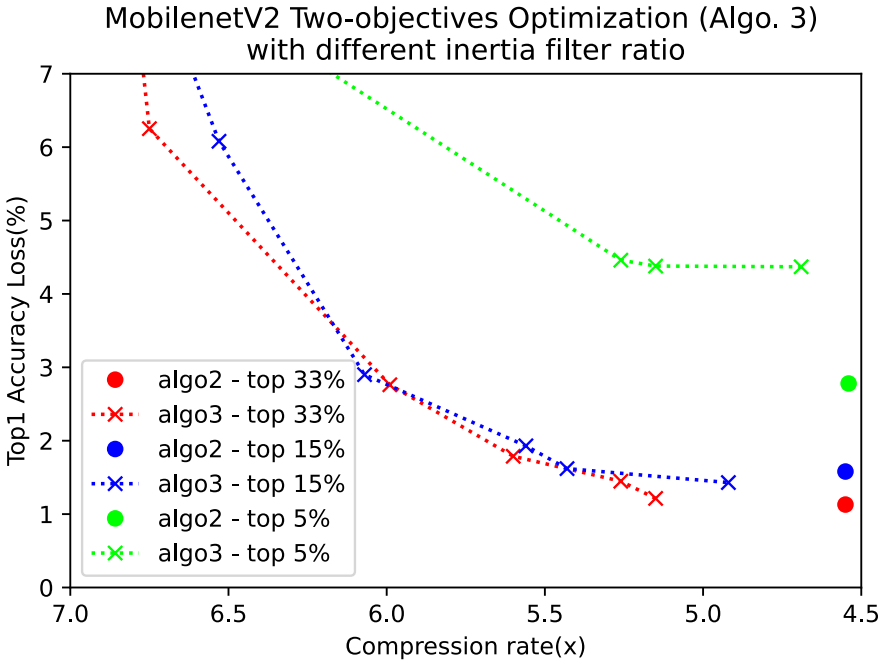



Figure 7: Results of multi-objective Explorations for MobileNet v2 with 3 different inertia filtering value. Comparison with results obtained with the second algorithm

Figure 7 presents the results obtained by applying the multi-objective version of Algorithm 3 to MobileNet v2. The chart reports on the X-axis the Compression Rate (eq 1) and the Y-axis shows the Accuracy Loss. Three Pareto frontiers corresponding to different inertia filter values are shown. As expected, a higher CR is achieved at the cost of higher accuracy loss and *vice-versa*. From these results, it is now possible to select among different solutions presenting a trade-off between AL and CR.

The second point of the results shown in Figure 7 is that the filtering level impacts the results, again this was expected because depending on the filtering level we have a more or less precise evaluation of the applied approximation. On the other hand, we can note that the Pareto frontier obtained using 15% of filtering is close to the one related to 33% of filtering. This means that inertia is really an efficient way to estimate the accuracy loss when WS is applied. Also choosing 15% as a value for the inertia filter provides the best trade-off between results and execution time, as the obtain Pareto front is nearly the same as the one with 33% but the execution time is 2x faster (see Table5).

Figure 7 also compares the results of Algorithm 3 with the results of Algorithm 2. We can see that except for 5% inertia filtering value, the solutions obtained with Algorithm 3 dominate those of Algorithm 2.

Table 5 reports the execution time required to obtain the three Pareto frontiers. The time is reported in hours and we divide it as clustering (cl.) and scoring (sc.), while the total time is simply the sum. As already pointed out in table 3, the scoring time is the dominating the overall execution time. Looking at the absolute values, we can note that

Table 5: MobileNet v2 - multi-objective DSE Elapsed time, obtained with Algorithm 3 on MobilenetV2

| **Filtering** | **Cl. time (hours)** | **Sc. time (hours)** | **Tot. Time (hours)** |
|---|---|---|---|
| 5% | 4 | 32 | 36 |
| 15% | 6 | 120 | 126 |
| 33% | 6 | 280 | 286 |

executing the multi-objective exploration with 5% of filtering requires 36 hours, which is $2/times$ less than running Algorithm 1 on the same CNN. Finally, we can safely use 15% of filtering to achieve results as precise as the 33% filtering but with $2/times$ less of execution time. Once again, this confirms that the use of inertia as a sensitivity metric allows a significant speedup and opens the door to more complex exploration for Weight Sharing based approximation.

## 5. Conclusion and Future Work

This paper presented our work towards the application of the weight-sharing technique to approximate a given CNN. A Design Space Exploration has been first introduced through a greedy algorithm. The presented DSE algorithm was able to successfully compress a CNN incurring in a very small accuracy loss. Reaching around 5× compression with less than 1% top-1 accuracy loss. The most important advantage is the fact that our approach is a one-shot conversion, and thus, we can avoid the prohibitive cost and constraints tied to network retraining.

Despite the good results, the computational cost of this first DSE algorithm was prohibitive (up to 80 hours of optimization), especially for bigger CNNs. We thus identified an alternative way to estimate the accuracy without running a full inference. We used the inertia computed during $k$-means execution as a local proxy for accuracy loss. We showed that inertia correlates well with accuracy and we were able to reduce the overall execution time up to 12× with a negligible cost in terms of accuracy and compression loss.

The reduction in the number of scoring iterations provided by the filtering allowed for more complex two-objective optimization, showing the trade-off between compression rate and accuracy loss. The results we obtained on the Mobilenet CNN show that setting the inertia filtering to 15% already gives a close to optimal Pareto front with greatly reduced optimization time.

We have compared our results with two state-of-the-art CNN weight sharing methods. And despite both of them use training as a way to recover the accuracy loss due to the approximation, we obtained similar if not better results. We obtained more than 4× compression on GoogleNet on the ImageNet dataset with less than 1% accuracy loss in less than 5 hours and without any retraining step. We intend to continue our work on approximating CNNs by first exploring other design space objectives such as performance and hardware metrics. Finally, we will extend the framework to support different approximation strategies such as pruning and quantization.

## 6. Acknowledgments

This work has been funded by the French National Research Agency (ANR) through the AdequatedDL research project (ANR-18-CE23-0012).

## References

[1] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, V. Asari, A state-of-the-art survey on deep learning theory and architectures, Electronics 8 (2019) 292. doi:10.3390/electronics8030292.

[2] V. Sze, Y. Chen, T. Yang, J. S. Emer, Efficient processing of deep neural networks: A tutorial and survey, Proceedings of the IEEE 105 (12) (2017) 2295–2329. doi:10.1109/JPROC.2017.2761740.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489. doi:10.1038/nature16961.
URL `http://dx.doi.org/10.1038/nature16961`

[4] W. J. D. Song Han, Huizi Mao, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, arXiv (2016).
URL `https://arxiv.org/abs/1510.00149`

[5] C. Baskin, E. Schwartz, E. Zheltonozhskii, N. Liss, R. Giryes, A. M. Bronstein, A. Mendelson, UNIQ: Uniform Noise Injection for Non-Uniform Quantization of Neural Networks, arXiv (Apr. 2018).
URL `http://arxiv.org/abs/1804.10969`

[6] A. Acharya, R. Goel, A. Metallinou, I. Dhillon, Online Embedding Compression for Text Classification using Low Rank Matrix Factorization, arXiv (Nov. 2018).
URL `http://arxiv.org/abs/1811.00641`

[7] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531 (2015).

[8] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, ArXiv abs/1704.04861 (2017).

[9] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, Z. Chen, L. Wang, A. Xiao, J. Chang, X. Zhang, Q. Tian, Weight-sharing neural architecture search: A battle to shrink the optimization gap, ArXiv abs/2008.01475 (2020).

[10] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, Y. Wang, A systematic DNN weight pruning framework using alternating direction method of multipliers, CoRR abs/1804.03294 (2018). arXiv:1804.03294.
URL `http://arxiv.org/abs/1804.03294`

[11] A. Zhou, A. Yao, Y. Guo, L. Xu, Y. Chen, Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights, CoRR abs/1702.03044 (2017).
URL `http://arxiv.org/abs/1702.03044`

[12] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, K. Roy, Energy-efficient neural computing with approximate multipliers, ACM Journal on Emerging Technologies in Computing Systems (JETC) 14 (2018) 1 – 23.

[13] J. A. Hartigan, M. A. Wong, A k-means clustering algorithm, JSTOR: Applied Statistics 28 (1) (1979) 100–108.

[14] A. Bosio, I. O'Connor, G. S. Rodrigues, F. K. Lima, E. I. Vatajelu, G. Di Natale, L. Anghel, S. Nagarajan, M. C. R. Fieback, S. Hamdioui, Rebooting computing: The challenges for test and reliability, in: 2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2019, pp. 8138–8143.

[15] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing Deep Convolutional Networks using Vector Quantization, arXiv (Dec. 2014).
URL `http://arxiv.org/abs/1412.6115`

[16] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, CoRR abs/1504.04788 (2015). arXiv:1504.04788.
URL `http://arxiv.org/abs/1504.04788`

[17] J. Wu, Y. Wang, Z. Wu, Z. Wang, A. Veeraraghavan, Y. Lin, Deep k-Means: Re-Training and Parameter Sharing with Harder Cluster Assignments for Compressing Deep Convolutions, arXiv (Jun. 2018).
URL http://arxiv.org/abs/1806.09228

[18] Y. Chen, J. Emer, V. Sze, Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 367–379. doi:10.1109/ISCA.2016.40.

[19] K. Ullrich, E. Meeds, M. Welling, Soft weight-sharing for neural network compression, ArXiv abs/1702.04008 (2017).

[20] M. S. Razlighi, M. Imani, F. Koushanfar, T. Rosing, LookNN: Neural Network with No Multiplication, in: Proceedings of the Conference on Design, Automation & Test in Europe, 2017, pp. 1779–1784.
URL http://dl.acm.org/citation.cfm?id=3130379.3130793

[21] J. Wu, C. Leng, Y. Wang, Q. Hu, J. Cheng, Quantized convolutional neural networks for mobile devices, IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016).

[22] E. Dupuis, D. Novo, I. O'Connor, A. Bosio, On the automatic exploration of weight sharing for deep neural network compression, in: Proceedings of the 23rd Conference on Design, Automation and Test in Europe, DATE '20, EDA Consortium, San Jose, CA, USA, 2020, p. 1319–1322.

[23] E. Dupuis, D. Novo, I. O'Connor, A. Bosio, Sensitivity analysis and compression opportunities in dnns using weight sharing, 2020 23rd International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS) (2020) 1–6.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks (2018). arXiv:1801.04381.

[25] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[26] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRR abs/1512.03385 (2015). arXiv:1512.03385.
URL http://arxiv.org/abs/1512.03385

[27] E. Dupuis, D. Novo, I. O'Connor, A. Bosio, On the automatic exploration of weight sharing for deep neural network compression, to appear in Proceedings of DATE2020 (2020).

[28] J. Bai, F. Lu, K. Zhang, et al., Onnx: Open neural network exchange, https://github.com/onnx/onnx (2019).

[29] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, CoRR abs/1512.01274 (2015). arXiv:1512.01274.
URL http://arxiv.org/abs/1512.01274

[30] V. Markovtsev, M. Cuadros, src-d/kmcuda: 6.0.0-1 (Feb. 2017). doi:10.5281/zenodo.286944.
URL https://doi.org/10.5281/zenodo.286944

[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09, 2009, pp. 248–255.

[32] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, K. Keutzer, Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size, CoRR abs/1602.07360 (2016). arXiv:1602.07360.
URL http://arxiv.org/abs/1602.07360

[33] S. P. Lloyd, Least squares quantization in pcm, IEEE Transactions on Information Theory 28 (1982) 129–137.

[34] D. Yang, W. Yu, A. Zhou, H. Mu, G. Yao, X. Wang, Dp-net: Dynamic programming guided deep neural network compression, ArXiv abs/2003.09615 (2020).

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, CoRR abs/1409.4842 (2014). arXiv:1409.4842.
URL http://arxiv.org/abs/1409.4842