Sorbonne Université
Institut des Systèmes Intelligents et Robotiques
Datakalab

Spécialité:
Informatique et Mathématiques appliquées

Efficient Neural Networks: Post Training Pruning and Quantization

Thèse de doctorat

Yvinec Edouard

Composition du jury

Pr. Catherine ACHARD	- **Présidente du jury**
Pr. Vincent LEPETIT	- **Rapporteur**
Pr. Eric MOULINES	- **Rapporteur**
Dr. Remi GRIBONVAL	- **Examinateur**
Pr. Matthieu CORD	- **Invité**
Dr. Kevin BAILLY	- **Directeur de Thèse**
Dr. Arnaud DAPOGNY	- **Encadrant Industriel**

Soutenue le 15 novembre 2023

Preamble

As I conclude a significant chapter in my journey to becoming a researcher, I would like to trace back to the origins. As I was studying in the double bachelors program at Sorbonne Université in mathematics and computer science, I met Arnaud Dapogny who taught, to my class, computer vision basics. Back then, I was a beginner in gymnastics and gym workouts in general, and it was quite obvious that Arnaud was very skilled in the matter. Consequently, I decided to ask him where and when did he use to train which happened to be at the same place where I used to perform my own training routine. He offered to train together and from then began our friendship.

The year after, I studied in the master of applied mathematics at Sorbonne and I decided to prepare myself for a prestigious master of mathematics applied to finance called the master El Karoui. Although at the time I had my doubts about this decision and was also envisioning to study artificial intelligence. After a long conversation with Arnaud, he explained to me that in finance, one has to work really hard but in exchange can make a lot of money while in artificial intelligence one can work less hard and still make a lot of money. However, as a person of great moral values, that abides by strong principles and based on this information, I went on to study artificial intelligence in the master MVA.

In the meantime, Arnaud took me in as an intern at Datakalab to study gaze tracking. This was my first hands-on experience in deep learning research. The interactions were great, and I was given room to explore and test ideas but still under a close supervision from my supervisors Arnaud and also Kevin Bailly. This experience ended with a publication at the face and gesture conference (FG 2020). After this internship and a second one for my end of study, Arnaud and Kevin enrolled me at Datakalab and crafted a PhD thesis subject for me: deep neural networks acceleration.

This PhD in an industry was my first proper work experience which was greatly facilitated by great mentors in the persons of Arnaud and Kevin, but also thanks to the kind (and patient) developer Gabriel Kasser as well as the mindful direction, Lucas and Xavier Fischer. As the months passed, I had the opportunity to contribute to the academic knowledge regarding deep neural network compression and was given room for investigating my own intuitions and innovations. From the wise advice of Arnaud and Kevin, my initial works focused on data-free compression in order to avoid power hungry challenges. Because of this strategic choice and thanks to some well-timed ideas, I contributed to post-training quantization and pruning through the publications of several articles in prestigious conferences such as NeurIPS and ICLR. Furthermore, as Kevin was supervising other PhD candidates, Gauthier Tallec and Jules Bonnard, working on other deep learning domains, I was exposed to other fields which led to the application of contributions from the compression domain to affective computing for instance. This work was significantly facilitated thanks to the work of Gauthier Tallec. Overall, I believe that the numerous discussions and interactions with my two PhD colleagues had a direct impact on the quality of my research. On top of which, I received guidance from Matthieu Cord on a regular basis, that enabled me to better target and track the moves and trends within the deep learning community.

As I grew more confident in my work and skills, Kevin, Arnaud and Matthieu gave me opportunities to evolve on the other traits of a researcher outside of research in itself. For example, Matthieu brought me to the French-German meet-up on deep learning that all the large public labs from both countries attended. This was my first experience in a scientific conference because of Covid. On the other hand, Kevin allowed me to present the compression stack from Datakalab to our privileged partners. All in all, from my perspective, the PhD had the right balance between industrial interactions, academic research and public representation for me to learn and develop the necessary skills to become as accomplished as possible. However, one element is missing in this picture: teaching. While, I had all the opportunity to learn research, as a PhD candidate in a private company, I would have missed on teaching. Fortunately, a former employee, Boris Dorado, who taught mathematics at Central Supelec in the MSTM, offered me to take his place as he wanted to follow a new academic formation. I gladly took this opportunity and started teaching on my vacations. Further down the road two other opportunities to teach appeared, one from Kevin at Sorbonne and one from Arthur Douillard at Epita. Thanks to this experience, I learned two lessons: first, they were all great teachers that were barely possible to replace and second, teaching to others is the best way to learn to communicate. This is an opportunity for me to thank all of my students who for the most part were better students than I was a teacher. As I learned my lesson, it reminded me of a quote from my favorite teacher in second year, Camille Pouchol, who said something along the lines "understanding complex concepts or explaining to the many simpler ones, it is not always trivial to figure which is best". Well, for myself, I would say that, in applied deep learning, one of these options do not exist. Consequently, I wanted to turn this manuscript in a pedagogical journey through deep neural network compression for all the AI researchers and deep learning enthusiasts.

For the remainder of the manuscript, I will present a general overview of the deep compression domain with high-level explanations of each set of methods such as neural architecture search, pruning and quantization. I will also embellish this landscape with some technical details and mistakes to avoid based on my own knowledge of the field. Each of the sections will end with a framed summary of the key takeaways for a better readability. For each specific subdomain and based on what was introduced, I will propose a list of key challenges that remain to be addressed: some were solved in this thesis or by other researchers and some remain open to this day. The second and third chapters will focus on my personal contributions to the field of pruning and quantization with the technical details, theoretical proofs and experimental validations. The last chapter will open-up on what appears to me as the key challenges that have yet to be addressed in deep compression as a whole. I wish you a good time reading my prose.

Abstract

Deep neural networks have grown to be the most widely adopted models to solve most computer vision and natural language processing tasks. Since the renewed interest, sparked in 2012, for these architectures, in machine learning, their size in terms of memory footprint and computational costs have increased tremendously, which has hindered their deployment. In particular, with the rising interest for generative AI such as large language models and diffusion models, this phenomenon has recently reached new heights, as these models can weight several billions of parameters and require multiple high-end GPUs in order to infer in real-time. In response, the deep learning community has researched for methods to compress and accelerate these models. These methods are: efficient architecture design, tensor decomposition, pruning and quantization. In this manuscript, I paint a landscape of the current state-of-the art in deep neural networks compression and acceleration as well as my contributions to the field. First, I propose a general introduction to the aforementioned techniques and highlight their shortcomings and current challenges. Second, I provide a detailed discussion regarding my contributions to the field of deep neural networks pruning. These contributions led to the publication of three articles: RED, RED++ and SInGE. In RED and RED++, I introduced a novel way to perform data-free pruning and tensor decomposition based on redundancy reduction. On the flip side, in SInGE, I proposed a new importance-based criterion for data-driven pruning. This criterion was inspired by attribution techniques, which consist in ranking inputs by their relative importance with respect to the final prediction. In SInGE, I adapted one of the most effective attribution technique to weight importance ranking for pruning. In the third chapter, I lay out my contributions to the field of deep quantization: SPIQ, PowerQuant, REx, NUPES, and a best practice paper. Each of these methods address one of the previous limitations of post-training quantization. In SPIQ, PowerQuant and REx, I provide a solution to the granularity limitations of quantization, a novel non-uniform format which is particularly effective on transformer architectures and a technique for quantization decomposition which eliminates the need for unsupported bit-widths, respectively. In the two remaining articles, I provide significant improvements over existing gradient-based post-training quantization techniques, bridging the gap between such techniques and non-uniform quantization. In the last chapter, I propose a set of leads for future work which I believe to be the current, most important unanswered questions in the field.

Abstract (fr)

Les réseaux de neurones profonds sont devenus les modèles les plus utilisés, que ce soit en vision par ordinateur ou en traitement du langage. Depuis le sursaut provoqué par l'utilisation des ordinateurs modernes, en 2012, la taille de ces modèles n'a fait qu'augmenter, aussi bien en matière de taille mémoire qu'en matière de coût de calcul. Ce phénomène a grandement limité le déploiement industriel de ces modèles. Spécifiquement, le cas de l'IA générative, et plus particulièrement des modèles de langue tels que GPT, a fait atteindre une toute nouvelle dimension à ce problème. En effet, ces réseaux sont définis par des milliards de paramètres et nécessitent plusieurs GPU en parallèle pour effectuer des inférences en temps réel. En réponse, la communauté scientifique et les spécialistes de l'apprentissage profond ont développé des solutions afin de compresser et d'accélérer ces modèles. Ces solutions sont : l'utilisation d'architecture efficiente par design, la décomposition tensorielle, l'élagage (ou pruning) et la quantification. Dans ce manuscrit de thèse, je propose de dépeindre une vue d'ensemble du domaine de la compression des réseaux de neurones artificiels ainsi que de mes contributions. Dans le premier chapitre, je présente une introduction générale au fonctionnement de chaque méthode de compression précédemment citée. De plus, j'y ajoute les intuitions relatives à leurs limitations ainsi que des exemples pratiques, issus des cours que j'ai donnés. Dans le second chapitre, je présente mes contributions au sujet du pruning. Ces dernières ont mené à la publication de trois articles: RED, RED++ et SInGE. Dans RED et RED++, j'ai proposé une nouvelle approche pour le pruning et la décomposition tensorielle, sans données. L'idée centrale était de réduire la redondance au sein des opérations effectuées par le modèle. Á l'opposé, dans SInGE, j'ai défini un nouveau critère de pruning par importance. Pour ce faire, j'ai puisé de l'inspiration dans le domaine de l'attribution. En effet, afin d'expliquer les règles de décisions des réseaux de neurones profonds, les chercheurs et les chercheuses ont introduit des techniques visant à estimer l'importance relative des entrées du modèle par rapport aux sorties. Dans SInGE, j'ai adapté l'une de ces méthodes les plus efficaces, au pruning afin d'estimer l'importance des poids et donc des calculs du modèle. Dans le troisième chapitre, J'aborde mes contributions relatives à la quantification de réseaux de neurones. Celles-ci ont donné lieu à plusieurs publications dont les principales: SPIQ, PowerQuant, REx, NUPES et une publication sur les meilleures pratiques à adopter. Dans SPIQ, PowerQuant et REx, j'adresse des limites spécifiques à la quantification sans données. En particulier, la granularité, dans SPIQ, la quantification non-uniforme par automorphismes dans PowerQuant et l'utilisation d'une bit-width spécifique dans REx. Par ailleurs, dans les deux autres articles, je me suis attelé à la quantification post-training avec optimisation par descente de gradient. N'ayant pas eu le temps de toucher à tous les aspects de la compression de réseau de neurones, je conclus ce manuscrit par un chapitre sur ce qui me semble être les enjeux de demain ainsi que des pistes de solutions.

Contents

Chapter 1

Landscape of Deep Neural Networks Compression

While some predictive tasks know analytical solutions [58], some others do not [220]. Prime examples of such problems are related, but not limited, to computer vision [48] and natural language processing [192]. Consequently, the search for approximate solutions to these task has grown rapidly, with deep learning models overtaking the lead since 2012 [120]. Their strong performance sparked a new interest from the machine learning community, which led to the discovery of several mathematical properties [83, 132, 108]. To many enthusiasts and experts, deep neural networks are a class of over-parameterized functions which are empirically effective at solving non-convex problems [273]. As a result, most of the scientific literature focused, and still focuses, on the training phase of deep neural networks and improving the final accuracy of these models. In such research, deep neural networks are often described as sets of parameterized functions which are learned from the landscape of a loss function [39] sampled over the distribution of the available data [54]. However, in the context of industrial applications, the generalization capacities of these functions (e.g. their accuracy) is no longer the main obstacle to their massive deployment but rather their computational cost. Consequently, the need for compression techniques has grown to the point where it is now considered as a proper domain of study in deep learning, called deep compression. In this introductory chapter, we define neural networks from the perspective of deep compression and paint a general landscape of the methods for efficient deployment.

1.1 Introduction to deep learning

In most deep learning methods, deep neural networks are viewed as a composition of functions. From the perspective of deep neural network efficient inference, such model is first defined as a set of computational blocks. The resulting object will be a directed graph where each node represents a mathematical operation and each vertex corresponds to an actual composition. In the following sections, we highlight the cost of the most commonly found computational blocks in modern deep learning models and their impact on hardware.

Let's consider a neural network F defined by its layers (or nodes) $(f_l)_{l\in[\![1;L]\!]}$, some of which comprise a weight tensor W_l and bias tensor b_l. Among the most common neural network architectures, we propose the following classification of nodes:

- Mat-Vect multiplications: these layers encompass fully-connected layers and convolutional layers. Such layers take as inputs a tensor X which may have any shape and perform an operation which is equivalent to a matrix (weights) and vector (inputs) multiplication. For the sake of simplicity, we will always assume that these operations are equivalent to a naive fully-connected layer [143].
- Normalization layers: batch-normalization [101] and layer normalization [14] layers were introduced in order to improve the stability of training. The defined operation is a sequence of affine transformations of the features X which can be generalized as $f_l : X \mapsto \gamma\frac{X-\mu}{\sigma}+\beta$ where γ and β are learned through stochastic gradient descent [181] and μ and σ are approximations of the expectation and standard deviation of X respectively.
- Activation layers: the previously mentioned layers only perform affine transformations of the data and as such can learn very few predictive functions [230]. To circumvent this limitation and enable

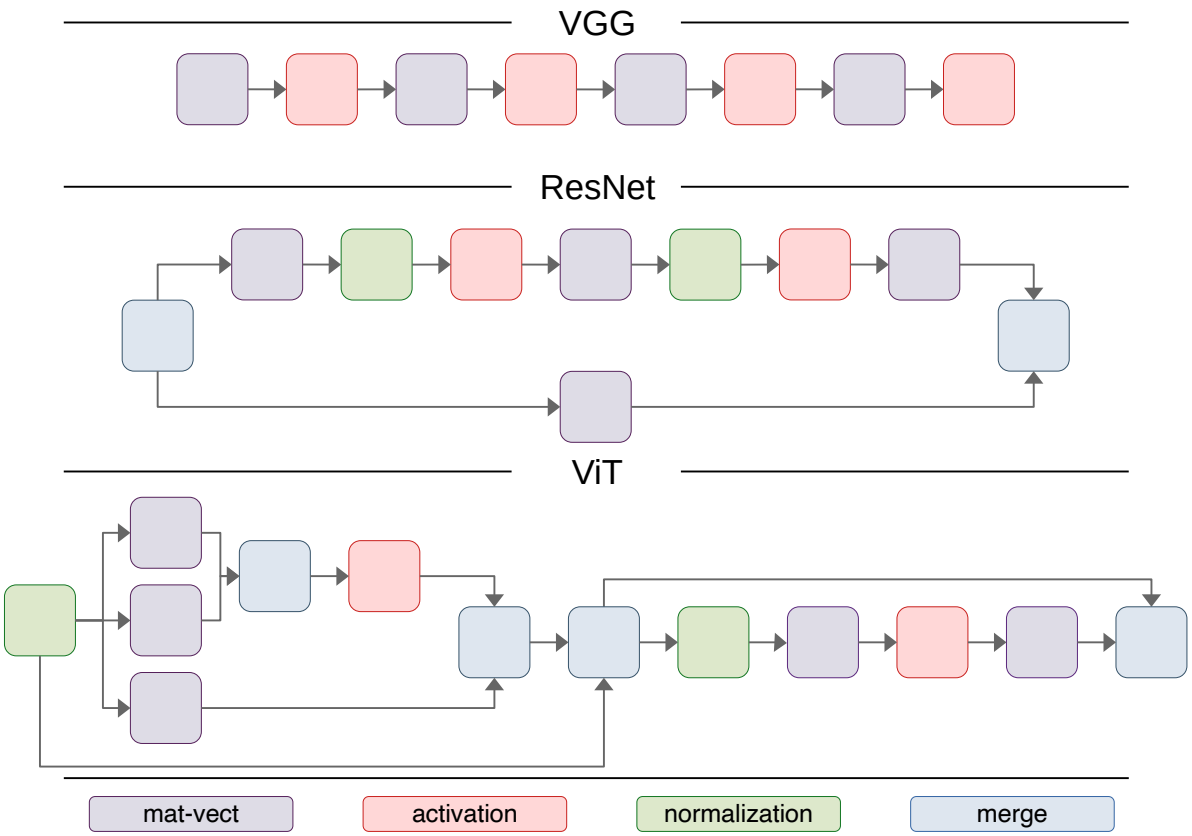


Figure 1.1: Illustration of characteristic computational sub-graphs from the most commonly used architectures in computer vision: VGG [191], ResNet [86] and ViT [55]. These architectures illustrate the most commonly observed computational blocks in the deep learning literature (sequential operations, residual connections and multi-head self-attention).

deep neural networks to approximate any continuous function on a compact set to any given precision, we add non-linear transformations to the graph called activation functions. Prime examples of such functions are the sigmoid [148], softmax [17], ReLU [3] and GELU [159].

- Merge layers: the final set of nodes that we consider are a consequence of the well known skip connections [86]. When several intermediate outputs are considered, they should be merged using either concatenation [180], addition [147] or multiplication [222].

For the sake of clarity, we illustrate some well known deep neural architectures in Figure 1.1. Although the mat-vect operations (fully connected and convolutional layers) are often the first layers that come to mind when we think about efficient inference, we will highlight the specific costs induced by all node types in the following sections.

1.1.1 Neural Networks and Matrix-Vector Multiplication

Contrary to matrix-matrix multiplication, there exists only one matrix-vector multiplication algorithm [202]. In practice, the main algorithmic degree of freedom is the number of operations performed in parallel, as matrix-vector operations can be heavily parallelized [171]. The most striking example of this practice is the case of convolutional layers for which Intel (with their inference engine, OpenVino [75]) provides a distinct set of instruction calls for each use-case (different kernel sizes, strides, paddings, ...). In this thesis, we will not extend too much on inference engines [146] and will remain, as much as possible, agnostic.

While the deep learning community has not focused on improving matrix-vector algorithms, the associated layers have been the center of attention when it comes to compression. This is explained by the fact that these layers contain most of the parameters (normalization and activation nodes may also use few parameters), and thus most of the memory footprint, of deep neural networks. They are also

responsible for most of the computations (GFlops[1]) during the forward pass. As we will detail in section 1.2, the memory footprint of deep neural networks, which comes from the weight values storage, has become a critical issue that hinders their deployment on edge devices and, sometimes, even on large scale GPUs. This is often addressed with pruning and quantization which we will introduce in sections 1.5, 1.4 and detail in Chapters 2, 3 respectively.

Although other formats have grown in popularity[2], we will systematically assume a baseline 32 bits encoding of the tensors that are manipulated in memory. This representation usually offers a sufficient approximation of real numbers. However, many practitioners may forget that this representation can also lead to numerical errors or approximations with respect to the expected exact result (using values in $\mathbb{R}$)). Throughout our discussion of compression techniques, we will highlight where such errors may hinder performance[3].

While mat-vect nodes are the most commonly found and are at the core of the expressivity of deep neural networks, normalization nodes are now ubiquitous in modern architecture.

1.1.2 Normalization Layers

Batch-Normalization Layers: First introduced by Ioffe *et al.* [101], batch normalization layers (BN) perform a standardization of the features before scaling and shifting them,

$$\text{BN} : X \mapsto \gamma \frac{X}{\sigma + \epsilon} + \beta \tag{1.1}$$

At inference, the key aspect of these layers (especially as compared to layer normalization layers [14]) is the absence of any update of the statistics μ and σ. In the case of batch normalization layers, μ is updated in order to satisfy $\mathbb{E}[X - \mu] \approx 0$ on the training set and remains static at inference. Similarly, σ is computed such that $\mathbb{V}[X/\sigma] \approx 1$, and stored. Consequently, during inference, the BN layers use the stored statistics and do not compute them on the fly.

Batch-Normalization Folding: Consequently, a BN layer is, at inference, a sequence of affine transformations and as such can be merged with the subsequent layers that also perform affine transformations. This merging operation consists in editing the graph in order to use only one affine transformation in place of two when possible. This process is referred to as layer folding. For the sake of simplicity, let's assume that we have a fully-connected layer followed by a BN layer. Then the sequence of operations S computes

$$S : X \mapsto \gamma \frac{WX + b - \mu}{\sigma} + \beta \tag{1.2}$$

This set of operations is very costly as we have a lot of back and forth between different instructions: first compute "WX" and add the bias "$\cdot + b$" then subtract the mean "$\cdot - \mu$"... The folded operation reads

$$S = S_{\text{folded}} : X \mapsto (\gamma \frac{W}{\sigma})X + (\frac{b - \mu}{\sigma} + \beta) = W_{\text{folded}} X + b_{\text{folded}} \tag{1.3}$$

The folded operation only computes one affine transformation with the new weight values $\gamma \frac{W}{\sigma}$ and bias $\frac{b-\mu}{\sigma} + \beta$. Although the original sequence and the folded one are mathematically identical, as previously mentioned, the numerical approximation can induce slight modifications of the predictions. In practice, the removal of BN layers leads to 20-50% latency reduction, depending on the network and hardware[4]. In our work *To fold or not to fold* [247], we propose an optimal algorithm to determine whether a BN layer can be folded or not. In general, batch-normalization folding can also be performed on more complex graphs where a BN layer is folded in several layers, as detailed in our work [247].

[1] GFlops: the number of floating point operations (measured in 10^9 operations). GFlops and the number of parameters removed are the two main metrics for pruning. Similarly, BOPs (number of bit-wise operations) are gaining traction in the quantization community. However, as we will discuss, these metrics offer interesting but limited insight on the actual performance of a model at inference time.

[2] While 32 bits remain the default representation, the rise of large language models [262] has pushed the community toward the adoption of 16 bits (fp16) representation for the most recent research.

[3] Numerical errors are important to account for, in critical systems such as airplanes landing assistance. In section 1.1.2, we will discuss a compression method that does not modify the predictive function from the mathematical perspective but may lead to marginal changes in practice due to numerical approximations.

[4] If you want to give it a try, please follow this small practical session on your device. batch-normalization folding is one of the few compression steps that can be fairly simply implemented for immediate acceleration. https://gitlab.com/ey_datakalab/advancedmachinelearning_compression

This step is crucial to any effective neural network compression technique, but it suffers from several limitations. First, some architectures [55] do no use BN layers, but rather layer normalization (LN) layers.

Layer-Normalization Layers: These layers perform essentially the same operation but with two distinctions. On the one hand, LN layers are usually performed on a different axis of the considered tensor. We do not detail this aspect of LN layers as this does not impact the inference. On the other hand, LN layers, contrary to BN layers, always compute the mean and standard deviation on the fly, for each example separately rather than storing it during training, as illustrated in Fig 1.2. Consequently, the operations performed by these nodes cannot be fully folded.

To fold a LN layer, we will explicit how to compute the $WX+b-\mathbb{E}[WX-b]$ at once. To do so, we recall that $\mathbb{E}[WX+b] = W\mathbb{E}[X]+b$. In other words, we search for a transformation T of X such that $W\mathbb{E}[X]+b = T(X)$. However, the expectation of an affine transformation is the affine transformation of the expectation. Consequently, we get

$$T(X) = \left(W \times \begin{pmatrix} 1-\frac{1}{\#X} & \dots & \frac{1}{\#X} \\ \vdots & \ddots & \vdots \\ \frac{1}{\#X} & \dots & 1-\frac{1}{\#X} \end{pmatrix} \right) X + b \tag{1.4}$$

where $\#X$ denotes the dimension of X. As a result, we have folded the centering operation. Unfortunately, the reduction step cannot be folded in an affine transformation, as it is quadratic with respect to the inputs. Thus, LN layers cannot be further folded.

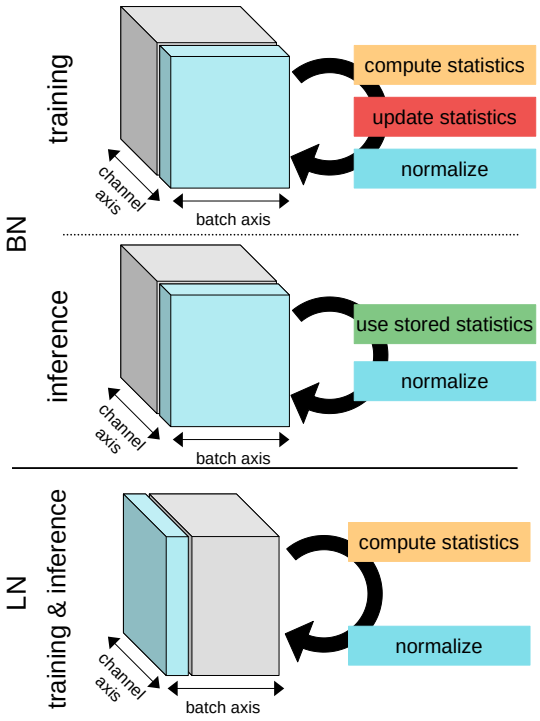


Figure 1.2: We illustrate the call to the normalization layers parameters.

A second limitation to normalization layers folding is the fact that we lose the information about the statistics learned during training. These statistics can be very helpful for many other compression techniques[5] [157, 240, 243]. In short, some compression techniques require knowing the support of intermediate features in order to map these values to a compressed space [99]. Their data-free counterparts leverage the training statistics in order to keep a control over the ranges of the intermediate feature values [157]. Some other methods propose to generate synthetic training examples from the weight values of a pre-trained model by optimizing a white noise input [240]. To do so, they regularize the generation process by constraining the intermediate statistics to match the ones extracted during training [243].

A third limitation of BN folding is the low impact of the memory footprint. As previously mentioned, most of the parameters of a model correspond to the mat-vect nodes (fully-connected and convolutional layers) and are responsible for most of the model footprint. Consequently, the removal of BN layers only lead to a marginal reduction of the total number of parameters.

In summary, while BN folding is a very effective acceleration technique with strong mathematical properties (preservation of the predictive function), it is limited both in its application (architecture) and practical results (low number of parameters removed).

Furthermore, there are instances where BN folding cannot be performed: mainly in the presence of a non-affine operation, *i.e.* an activation layer.

1.1.3 Activation Functions

Similarly to mat-vect nodes, activation functions have been a key component to the good performance of deep neural networks since their introduction [123]. From the perspective of compression, the baseline and

[5] The usage of the train statistics is very important in quantization, but we keep this conversation for the section 1.4.

best suited activation function is the rectified linear unit (ReLU) [3], defined as $\text{ReLU}(X) = \max(X, 0)$. Its strengths are two-fold: First, ReLU is almost cost-free in terms of computations. In order to compute it we only have to check the value of a single bit. Following the convention[6] on floating point values, if the sign bit is 0 then we shall return X and 0 otherwise. Furthermore, the sign bit is always located on the most significant bit (easier access). In other words, ReLU is as costly as a binary operation[7]. Second, ReLU is piece-wise linear, which means that for all $\lambda \in \mathbb{R}_+$ we have $\text{ReLU}(\lambda X) = \lambda \text{ReLU}(X)$. This property has been leveraged in many compression techniques [157]. In short, if we need to scale down or up the activations or weights of a node, we can fold this operation in any subsequent affine node as long as we have a ReLU activations. This is a weak form of compliance with operation folding.

Unfortunately for us, the deep learning community is progressively abandoning the ReLU in favor of other activation functions[8] such as ReLU6, Hardswish [94], GELU [159] and SiLU [56]. While it appears that these new transformations offer better performance when used in the context of stochastic gradient descent based optimization [181], they significantly hinder most compression techniques.

Fortunately, based on their analytical definitions, we can assert their similarity with the ReLU activation function.

$$\text{GELU} : X \mapsto \frac{X}{2}\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}(X + 0.044715X^3)\right]\right) \qquad \text{SiLU} : X \mapsto \frac{X}{1+e^{-X}} = X\sigma(X) \tag{1.5}$$

where σ is the sigmoid function[9]. In section 1.3, we will detail a compression technique which enables the replacement of non-ReLU activations with the inference-friendly ReLU.

While the activation functions introduce some computational overhead, they do not affect the memory footprint of the neural network at inference, up to the instructions related to their executions, which is negligible. On the contrary, the remaining nodes: feature merging nodes are often overlooked when it comes to deep acceleration.

1.1.4 Merging Nodes

Even the simplest feature merging nodes, such as the concatenation and addition nodes, introduce a significant overhead. This overhead does not result from their specific computations, but rather from the need to store more intermediate features simultaneously. In order to explain this phenomenon, we need to highlight a key difference between inference and training: the absence of the backward propagation. Because we do not perform backward propagation at test time, we do not need to store all the intermediate features exploited in the chain rule. As a result, for a sequence of nodes without intermediate connections (also called skip connections [86]), we only need to store the current features that we are manipulating. These scenarii are illustrated in Fig 1.3. However, when we observe merging nodes (and thus skip connections), we need to keep the two (or more) intermediate feature tensors in order to perform the forward pass. This phenomenon can be memory costly for some computer vision models, which use high resolution images [31]. Unfortunately, there is only one solution to tackle this problem: using a different model.

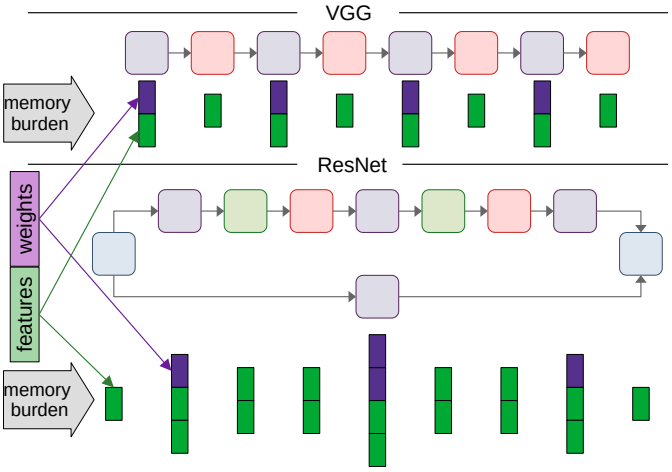


Figure 1.3: Illustration of the memory footprint at each inference stage, with purple boxes for the weights and green boxes for the features.

A more complex and costly merging mechanism is the matrix multiplication of intermediate feature tensors. A prime example

[6] This is not systematically the case in computer science. However, it is the case for all deep learning libraries (TensorFlow, Keras, PyTorch, ONNX, JAX,...)

[7] As we will see, to this day, binarization of the computations is the highest compression rate that can be achieved for deep neural networks inference.

[8] In section 1.1.5, we provide a detailed description of the most commonly used models and benchmarks for deep neural network compression and specify the activation functions in each architecture

[9] Throughout the thesis, we will try our best to simplify the notations. Here the σ refers to the sigmoid while, previously, σ referred to the standard deviation. In the other sections and unless stated otherwise, σ will, by default, refer to the standard deviation.

of such merging is the attention module from the trending transformer architecture [55]. While the mat-vect multiplication algorithm is unique (up to the parallelization), the matrix multiplication algorithms are numerous and well studied [202]. The choice of such algorithm is closely related to the hardware support. As such, it is common practice to leave such optimization to the inference engine[10]. On the other hand, there exists some techniques specific to the acceleration of this mechanism: tensor decomposition and clustering, which we discuss in section 1.5. In short, these techniques either reduce the size of the matrices or approximate the operation using a sub-sample of the intermediate computations.

In order to introduce the different methods for deep neural networks acceleration and compression, we introduce once and for all the models and benchmarks that we will use in our experiments. These models are the starting point from which we try to achieve faster and less costly inference.

1.1.5 Neural Networks and Datasets

In order to clarify the discussion regarding models and datasets, we classify them in three categories: the debugging examples, the realistic industrial use-cases and the large generative models.

The first category, debugging examples are also referred to as toy examples. We choose to call them debugging examples as they should be considered as good illustrations to showcase whether a compression method implementation runs or not, and nothing more. We observed this issue repeatedly during the thesis, and this has been reported on many occasions in different surveys of the field [131, 70, 36]. For example, a compression method that work very well on a small network trained on MNIST [49] can very well be ineffective when applied to a large model on ImageNet [47]. As a result, these benchmarks should not be considered for future evaluations of deep neural network compression. Nonetheless, we will use them in our study, as they are often the only available point of comparison to some older methods. For our debugging examples, we consider all networks trained on MNIST, a set of 60,000 28×28 grayscale images for 10-classes classification with a test set comprising 10,000 images. We also consider models trained on Cifar10[11] [114], a set of 50,000 32×32 RGB images for 10-classes classification, with a test set comprising 10,000 images. The considered models are almost systematically from the ResNet [86] and Wide-ResNet [257] families, which are known for their introduction of the skip connection and narrower architectures respectively. During this thesis, we also worked on MobileNet v2 [183] trained for Cifar10. However, contrary to the ResNet models, the MobileNets do not have a specific architecture adapted to Cifar10 which makes them significantly over-parameterized. For all of these models, we list their characteristics with respect to inference in Table 1.1.

The second category, industrial use-cases, could be summarized as ImageNet [47] models and BERT [52]. Indeed, when it comes to computer vision applications, ImageNet has become the reference benchmark for real-world scenarii. Although many compression techniques also evaluate their performance on other tasks such as object detection on VOC [60] or COCO [134] as well as image segmentation on VOC [60], CityScapes [44] or ADE20K [269], we observe that, in practice, the performance of a compression method, on these benchmarks, can be deduced from the performance on ImageNet. This empirical phenomenon can be attributed to several elements. First, the ImageNet classification task has a complexity (prediction over a 1000 classes) similar to the dense (or semi-dense) predictions from the aforementioned datasets. Second, it is known to improve performance for these tasks as it is often leveraged as a pre-training task.

> In summary, if a compression method works on ImageNet, it will most likely work on almost every other similarly sized applications such as VOC, COCO, CityScapes, ADE20K...

However, evaluating on ImageNet should no longer be considered sufficient. While a similar architecture (or backbone) will behave similarly with respect to compression, this property does not hold across architectures. In practice, It appears that the community struggles to depart itself from ResNet[12] and move on to the EfficientNet [210] and transformer families such as ViT [55], DeiT [216] and CaiT [217]. Consequently, we will evaluate the compression techniques on both older architectures (e.g. ResNet) and

[10] Although DeepMind designed a neural search for even better matrix multiplication algorithm [62] and did improve the state-of-the-art. Such work is rather a testament to the strength of deep learning methods than an actual compression technique.

[11] We did not work on Cifar100, but this dataset also falls in the category of debugging datasets. While not so many researchers have the resources to train on larger scaled dataset, we would recommend these researchers to work on less training intensive fields (post-training compression) rather than relying on empirical results that may not translate in actual deployment use-cases.

[12] Some methods still evaluate on VGG. However, this benchmark is progressively disappearing and ResNet should follow.

Table 1.1: Summary of the inference ralted attributes of the most commonly used deep architectures in compression benchmarks. The characteristics were obtained using the torchsummary [219] and deepspeed [149] libraries.

	Architecture	#params	Flops (10^6)	MACs (10^6)	RAM (Mb)	Activations	Year
debugging examples	ResNet 8	78,186	26.25	12.97	0.30	ReLU	2016
	ResNet 20	269,722	81.67	40.55	1.03	ReLU	2016
	ResNet 56	853,018	252.57	125.49	3.25	ReLU	2016
	ResNet 110	1,727,926	508.93	252.89	6.59	ReLU	2016
	ResNet 164	2,602,906	765.28	380.29	9.93	ReLU	2016
	Wide ResNet 28-10	36,489,290	11,913.28	5,951.12	139.20	ReLU	2017
	Wide ResNet 40-4	8,955,050	2,829.95	1,411.83	34.16	ReLU	2017
	MobileNet v2	3,504,872	15.06	7.39	13.37	ReLU6	2018
industrial applications	ResNet 18	11,689,512	3,636.25	1,814.07	44.59	ReLU	2016
	ResNet 34	21,797,672	7,339.39	3,663.76	83.15	ReLU	2016
	ResNet 50	25,557,032	8,211.11	4,089.18	97.49	ReLU	2016
	VGG16	138,357,544	30,973.78	15,470.26	527.79	ReLU	2014
	MobileNet v2	3,504,872	614.97	300.77	13.37	ReLU6	2018
	MobileNet v3	5,483,032	444.94	216.59	20.92	Hardswish	2020
	EfficientNet B0	5,288,548	793.64	385.81	20.17	SiLU	2019
	EfficientNet B7	66,347,960	10,472.72	5,169.87	253.10	SiLU	2019
	ViT b16	86,567,656	33,723.33	16,848.50	330.23	GELU	2021
	ViT h14	632,045,800	323,918.10	161,884.38	2,411.06	GELU	2021
	DeiT S	86,390,784	35,336.53	17,655.87	329.55	GELU	2021
	CaiT XXS24	11,956,264	4,351.81	2,169.28	45.61	GELU	2021
	BERT	109,482,240	96,724.84	48,318.97	417.64	GELU	2019
large auto-regressive models	OPT 7B	6,658,473,984	28,392,045.13	14,192,983.28	25,400.06	ReLU	2022
	OPT 13B	12,853,473,280	13,417,330.83	6,707,711.43	49,032.11	ReLU	2022
	OPT 30B	29,974,540,288	31,159,501.14	15,578,149.72	114,343.80	ReLU	2022
	Dolly v2 3B	2,775,086,080	2,806,570.07	1,402,801.73	10,586.11	GELU	2023
	Dolly v2 7B	6,856,056,832	6,973,753.57	3,486,265.18	26,153.78	GELU	2023
	LLaMA 7B	6,738,415,616	6,930,503,49	3,465,026,47	25,705.01	SiLU	2023
	LLaMA 13B	13,015,864,320	13,427,749.65	6,713,522.18	49,651.58	SiLU	2023
	LLaMA 33B	32,528,943,616	33,641,743.96	16,820,185.11	124,088.07	SiLU	2023
	LLaMA 65B	65,285,660,672	67,534,959.02	33,766,353.14	249,045.03	SiLU	2023
	Stable Diffusion	865,910,724	3,044,273.45	1,520,971.94	26,153.78	SiLU	2023

newer ones (e.g. EfficientNet). Similarly to the previous category, we summarize the properties of these models in Table 1.1.

The third category, the large generative models, comes from the recently introduced large language models (LLMs) [262] and diffusion models [179]. Their difference from the previously mentioned models comes from their sheer size. While the causal aspect of these models does affect some compression methods (discussed in detail in section 4.3), the main issue comes from the fact that training such models is not an option for almost every researcher and even loading such models can become a challenge past 30 billion parameters. As a result, these models have renewed the interest for data-free compression. A second, significant, aspect of these models is the presence of outlying values among weight and intermediate features [50]. This is an important challenge and, as such, it will be a recurrent point of discussion in the quantization chapter (see sections 3.1.3 and 3.1.4). As examples of such models, we will consider the OPT [262], Dolly v2 [46] and LLaMA [218] families of LLMs as well as the stable diffusion models [179]. We report their attributes in Table 1.1. In order to measure the performance of these models,

we considered the common sense reasoning[13] benchmarks from EleutherAI [68] for LLMs and the CLIP score [89] for the diffusion model.

From this landscape of the current state of deep learning through the lens of deep neural network acceleration and compression, we propose to motivate the subject from both an economical and environmental perspective.

1.2 Deep Neural Networks Deployment

In deep neural networks deployment, we usually consider two main setups: edge devices and cloud computing. From the inference perspective, using a set of small devices or a centralized server for inference leads to distinct constraints and goals, which we discuss in the following sections.

1.2.1 Edge Devices

Edge devices are usually the first use-case that comes to mind for deep neural network compression. In other contexts, an edge device is defined as any device that provides an entry point to a network provider (e.g. routers). However, in our case, an edge device will be any low power computing device. Examples range from most microcontroller units to smartphones. Working on edge devices introduces a specific set of constraints.

First, depending on the application, if a user interacts directly with the device, then the most likely criterion for acceleration would be latency, which corresponds to the response time for a single prediction. In order to measure the *latency* properly, it is important to account for several aspects of inference that lie outside the scope of deep acceleration:

1. The warm-up: most inference engines and libraries have a warm-up for the first few inferences, which corresponds to the selection of the algorithms for each computation and the loading of the neural network. In practice, we usually measure the performance over multiple iterations (about 1000) after a warm-up of a few iterations (about 10).

2. Satellite applications: on complex systems, it is likely that the inference of the deep neural network is not the unique application running. Thus, the resources are shared which may hinder the performance. In practice, we isolate the neural network inference and perform it alone. However, for industrial applications, it may be relevant to do *in-situ* measurements.

In other instances, where a device is not interacting with a user but rather runs constantly, the most relevant metric for acceleration is *throughput*. This corresponds to the number of inferences that can be completed in a given amount of time (usually one second). The main difference with latency is our ability to adapt the batch-size. To put it in simple words: the latency corresponds to the runtime in batch-size 1 regime, while the throughput is the runtime with the optimal batch-size. While these metrics are important to keep in mind, they are not commonly measured in academic research for one main reason: they are too complex to reproduce, thus disable fair comparisons. To put it in a nutshell, unless we use the exact same device, with exact same software updates and exact same satellite applications, two identical networks will not share the same latency nor throughput. This is the reason why we report the number of parameters, memory usage, number of Flops and MACs which all are independent of the hardware. The Flops are the number of floating point operations performance in a single forward pass, while the MACs is the number of multiply–accumulate operations which correspond to the addition of the result of a multiplication of two scalars to an accumulator.

Second, one of the main reasons to opt for the usage of edge devices is the energy consumption [27, 7, 72, 241]. As illustrated in Figure 1.4, we can see the power consumption of several edge devices in different execution mode: sleep (red) or peak activity (blue). While, we observe a significant difference from larger hardware devices, we also note that the peak consumption can be more than 200 times larger than the sleep mode performance (e.g. on an STM32 device). As a result, there is a significant interest in limiting the peak consumption of such devices. This is even more important on mobile devices (e.g. smartphones) where the battery life is of paramount importance in the user experience [160]. According to Ionascu *et al.* [102], the battery life is given by $\frac{\text{battery capacity}}{averagecurrentconsumption}$ (in Amperes). Consequently,

[13] There is no consensus on the appropriate evaluation metric for LLMs. A recent work [77] highlighted the limitations of the current evaluations. These issues are particularly pronounced when fine-tuning is involved, which will not be the case in our study of LLMs.

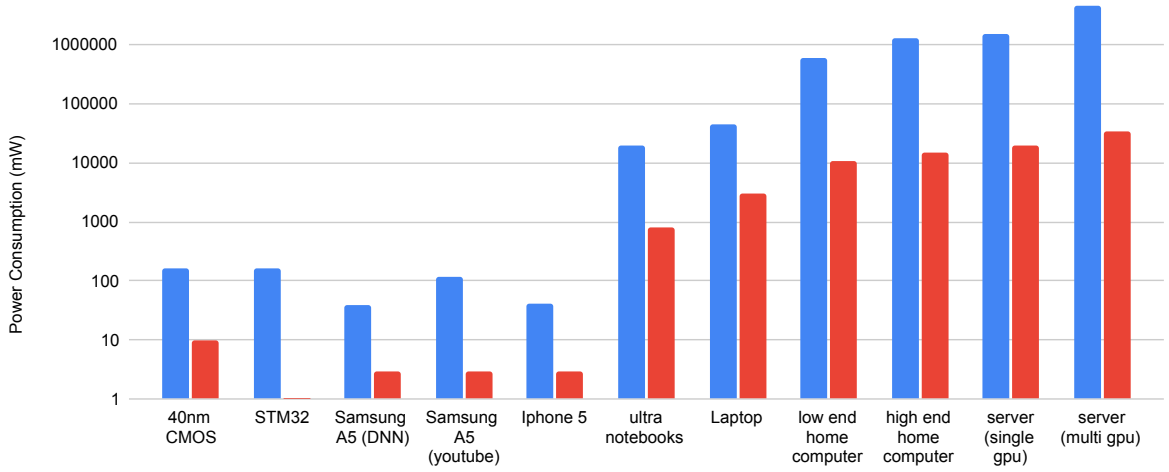


Figure 1.4: We put in perspective the power consumption of several hardware devices: 5 references for edge devices (five first) and 6 large devices (six last). In red, we report the sleeping mode power consumption and in blue the peak consumption. We considered two microcontroller units: a 40nm CMOS [72] and an STM32 [102]. The remaining edge devices are mobile device (smartphones): a Samsung A5 [258] and an Iphone 5 [151]. With respect to large devices, we report the power consumption of laptops: an ultra notebook [270] and a laptop [5] with standard capacity. We also report the performance of home desktops: a low-end [6] and a high-end [6] device. For the largest considered computers, we add the energy consumption of two server-grade computers: single GPU [105] and multi-GPU [57].

deep neural network compression is of paramount importance in order to preserve the user experience and limit the impact of related applications on the battery capacity[14].

Third, edge devices are often constrained by the necessity to run other computationally intensive operations. This is mostly the case with mobile devices such as smartphones. As reported in Figure 1.4, the YouTube application on its own draws more power than inferring a small convolutional neural network. Consequently, we cannot simply bound the size of a deep neural network to exactly fit the available memory and compute, but rather to fit the available resources when the system operates. An example of this kind of situation is the image processing occurring with recent smartphones recording applications [224]. In short, the application performs video encoding and image processing simultaneously.

Finally, edge deployment may add some more constraints, such as ultra low-power consumption. Some devices for the internet of things (IoT) [84] can only draw a few Watts per month as they are not wired and changing their battery is very costly. While the previously mentioned devices can have a few hundred Kb to a few Gb cache, such a tiny device will only have up to a dozen of kilobytes of memory for compute [35]. With Table 1.1 in mind, such a scenario requires extreme compression levels for deployment. Another challenging constraint arises from the backward compatibility: for instance with mobile device, a user will not change their phone as we release a new deep neural network architecture or compression technique. Consequently, methods should be designed for backward compatibility as much as possible.

On the other hand, edge devices often allow running applications which are privacy compliant, by design. As the computations are performed locally, the data is not transferred, which enables to protect the privacy of the users (e.g. the user identity) and the data which is crucial when dealing with health data [170].

In summary, edge devices (micro controller units and mobile devices) offer multiple strong benefits: low power consumption and privacy. However, these advantages come with a series of challenges: latency (for the user experience), shared resources (for complex software) and backward compatibility (the devices are not easily upgradeable).

[14] From a human health perspective, battery waste have been noted to significantly impact the health of Californian residents [172].

While edge devices represent an interesting objective for deep neural network compression and acceleration, they only represent a fraction of the inference workloads. To this day, cloud computing still encompasses 90% of the inference market, according to Gartner, Inc[15]. However, this number is predicted to drop to 25% in the future.

1.2.2 Cloud and Large Devices

Large devices and cloud computing constitute a straightforward environment for deep neural network deployment after training. The training process is always more computationally intensive than inference for an equivalent number of predictions. Consequently, hardware designed to support training can also serve the purpose of inference. In practice, multiple GPUs are required for training, while a single GPU is often enough for inference.

However, inferring on the could comes with a significant drawback: energy consumption. Masanet *et al.* [145] estimate that the current energy consumption from all data centers in the world is equivalent to half of the total energy produced by nuclear power plants in France or half of that of the US. Furthermore, projections [11] anticipate that, at the current rate, data-centers (around the world) energy consumption will be multiplied by 8 times over, before 2030. This problem has reached such a scale that the cost of running a data center is more correlated to the energy than it is to the hardware price [194]. Consequently, any sustainable development of deep neural network deployment will rely on either a drop in availability or heavy compression.

Contrary to edge devices, cloud computing is usually more permissive with respect to latency and focuses on throughput, as multiple users and tasks can be hosted on a shared device and solved with the same instance of the deep neural network. Another important distinction is the usage of the available resources. On edge devices the software may perform other computationally intensive operations, this is not the case on cloud computing as other operations are negligible as compared to the available compute and can be off loaded to dedicated and better suited devices. Furthermore, cloud computing is not suited for privacy rights compliance[16], as by definition the data is uploaded to the cloud. On top of this, the energy cost of streaming data [241] also increases to the total cost of using cloud computing. For example, uploading data to the cloud requires about 60mW for 2.5Mb of data, while processing this much data costs about 95mW on some edge devices as shown in Figure 1.4. In other words, in terms of power consumption alone, sending the data to the cloud and then receiving it back already costs more than processing it locally with an edge device.

In summary, cloud computing and inference on large devices comes with major drawbacks as compared to edge computing: lack of privacy compliance and major energy consumption (from the computation and upload of the data). However, these weaknesses are often shadowed by the advantages of cloud computing: straightforward migration from the training phase and easily upgradeable hardware and models.

To conclude this discussion on deep neural networks deployment in practice, we would like to mention hybrid solutions. In many instances it is necessary to run some of the compute (too heavy for an edge device) on the cloud but still perform most operations on the edge [233]. This is the kind of collaborations that have been proposed by Datakalab for people counting : where the images are processed on the edge and only aggregated statistics are uploaded to the cloud[17].

Stemming from this landscape on deep neural networks deployment, it appears that deep compression and acceleration is a field of paramount importance which will keep on growing with the challenges to come. Consequently, in the next sections, we will describe the most commonly used techniques to achieve more efficient inference. The order in which we will go through corresponds to the order that we would personally advise any enthusiast and expert to follow in their own journey towards efficient deployment.

[15]It is quite challenging to obtain open data regarding this subject as companies do not share such information. Nonetheless, these predictions from Gartner have been trusted by the deployment community. https://gitlab.com/ey_datakalab/advancedmachinelearning_compression

[16]This assertion can be mitigated by the work done at ZAMA, as researchers use holomorphic cryptography to protect the data over which computations are performed. However, their solution adds a massive overhead.

[17]This was documented by the CNIL (french watchdog for data privacy). Future work were considered to push further the collaboration between cloud and edge with hybrid inference (first layers on the edge and last layers on the cloud).

1.3 Using the Appropriate Architecture

In order to achieve efficient inference when tackling a task, it is of paramount importance to start from an appropriate deep neural network architecture. For instance, based on research in some fields [26] it appears that models such as VGG16 [190] and ResNet 50 [86] are still commonly leveraged for various computer vision tasks. However, these models comprise 138 and 25 million parameters, respectively. In comparison, EfficientNet B0 [210] has 5 million parameters for a similar or even higher accuracy (from 76.15% to 77.40% on ImageNet). In other words, working from EfficientNet B0 over VGG16 or ResNet 50 will lead to 27.6× and 5 × compression head start without any accuracy drop. Another example would be object detection models, which can either be single stage such as Yolo [213] or two stages such as Faster RCNN [213]. However, in practice, using a single stage detector usually translates in a significant latency improvement at virtually no cost in precision. This is mostly due to the focus of the detection community on the latter. As a result, it is of paramount importance for any resource-limited deep learning enthusiast to keep in touch with the most recently realized models and their performance[18]. To facilitate this preliminary work, the open AI community has done a lot of the heavy lifting, most recently with HuggingFace [231]. The large number of model zoos (from TensorFlow, Torchvision and HuggingFace) guarantees free access to most of the state-of-the-art deep neural architectures.

1.3.1 Neural Architecture Search

In order to further improve the efficient inference starting point, we have still several aspects over which we can work before and during training in order to facilitate the remaining steps specific to inference: neural architecture search (NAS), using the adequate layers and leveraging specificities from the target test distribution.

Neural architecture search has gained significant traction over the past few years, leading to the release of some of the best performing computer vision backbones available to this day, such as MobileNet v3 [94] and EfficientNet v2 [209]. Although we will detail many classes of approaches to NAS, we observe a common general definition of this challenge: define a set of possible architectures called the *search space*, then define a strategy to explore the search space called the *search policy* based on a performance metric (usually the precision of the models). The first class of NAS methods are evolutionary methods. The principal, called neuroevolution [21] loops over the following three steps: generation, fitness, mutation. Assume a model sampling strategy (generation strategy at time t), the generated models are evaluated (fitness) using metrics that can be non-differentiable. This is practical for multi objective processes (runtime duration can be an objective). Then, at the next generation time $(t+1)$, the new models are generated from the previous generation weighted by their fitness (increase the likelihood of best performing models) with mutations (to create new examples). This process is summarized in Figure 1.5. Most research in the field focus on improving the sampling strategies, fitness evaluation and mutation process [176, 199, 174, 200].

The second group of NAS methods is based on reinforcement learning (RL). Similarly to evolutionary methods, RL is suitable for searches in a discrete model space. However, RL-based NAS aims at reducing the computational cost of evolutionary methods, which tend to explore the search almost in its entirety [21]. From a RL perspective, we define an agent called the controller that predicts new architectures.

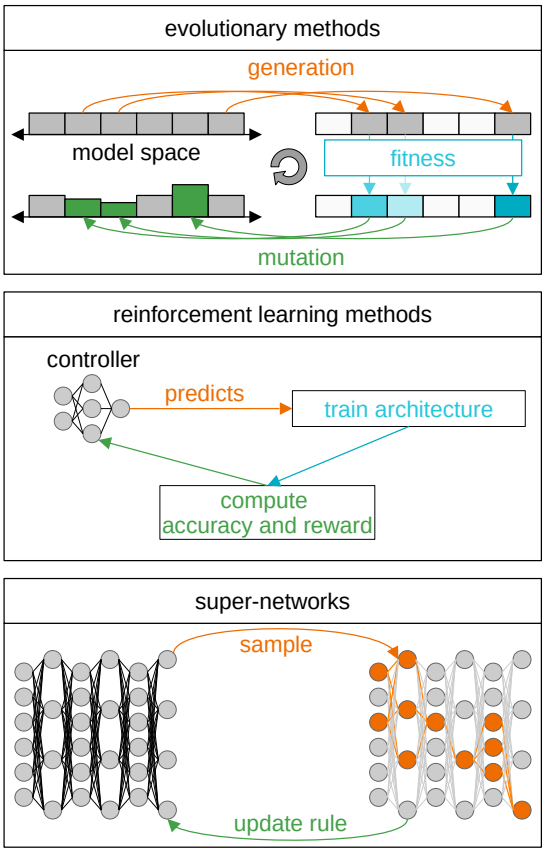


Figure 1.5: The different NAS categories.

[18] In particular, it is important to keep in mind that academic benchmark do not systemically make use of the most recent architectures for the sake of comparison which limits their relevance as deployment instructions or recommendations.

The controller is rewarded based on the validation accuracy of the sampled network, as illustrated in Figure 1.5. Because the accuracy is deterministic from the neural network architecture, as long as we set a seed, this corresponds to a one-armed bandit setting[19] [232]. Many methods have been proposed in order to perform sub-architectural selections and reward them [16, 271]. However, despite these efforts, the remaining cost of RL and evolutionary methods is the necessity to train multiple architectures in order to evaluate them, which leads to unreasonable costs and hinders the scalability of these methods to larger models.

The last group of NAS techniques is referred to as super networks or weight sharing. Super networks were introduced in order to significantly decrease the computational burden from NAS. Formally, a super network represents the search space, while each subnetwork represents a candidate element from the search space (see Figure 1.5). Consequently, two distinct subnetworks from the same super network share a fraction of their pre-trained weights. In return, the fine-tuning of these subnetworks is significantly less costly. However, a significant limitation to this approach is the cost of training a very large super network. As a result, some methods, proposed to learn a small scale network and then scale it up in order to offer a family of architectures from a single NAS [272, 169]. This method was exploited in DARTS [138] which learns the cell structure before stacking it to obtain the full deep architecture. In order to further optimize the NAS method, compound scaling [210] was introduced in order to efficiently find the scaling coefficients for the architecture family. The method simply consists in using a grid search in order to obtain the scaling factor α from the baseline architecture to its bigger counterpart. This factor is applied to the depth and width of the architecture in order to scale it up (e.g. if $\alpha = 2$ then the model will be twice as deep with layers twice as large). Then, in order to obtain larger members of the family, the coefficient is simply powered up (a^n). This method led to the implementation of the EfficientNet family [210].

In its current form, NAS could strongly benefit from actual inference performance insights. For instance, let's consider an EfficientNet architecture. From Table 1.1, it reads that its smallest family member only requires 5 million parameters and about 8 GFlops to perform one prediction on ImageNet. While this performance is significantly better than that of ResNet 50, it should be noted that we are faced here with a false friend: the depthwise convolution. The depthwise convolution [38] consists in a standard 2d convolution up to the fact that each output channel only depends on one input channel, which drastically reduces the number of parameters[20]. However, in practice, while the depthwise convolution does reduce the memory footprint of a model, it does not usually improve the latency of this model due to the lack of hardware optimization dedicated to this type of layer. Consequently, such architectural choices should be avoided. For example, the depthwise convolution or odd number of output neurons[21] are false friends in the context of efficient inference. As a result, this PhD work, among others, can provide insight on how to further improve the performance of current NAS methods.

However, the current costs of performing NAS remains a staggering obstacle for small research structures. So much so, during this thesis, we did not directly try to improve existing NAS methods. Consequently, we will assume that a baseline architecture is provided for the compression task: e.g. we have to work with an EfficientNet. Still, before jumping into the most common compression techniques (quantization and pruning), we will discuss how to refine our initial baseline architecture in order to simplify our future work.

1.3.2 Few Shot Architecture Improvement

The main architectural improvement that we propose consists in replacing ReLU-like activation functions with a proper ReLU. Prime examples of such activations are the SiLU and GeLU which are growing more and more common in most recently introduced deep architectures, namely EfficientNets and Transformers.

Some works have been conducted in this direction, which lead to the release of an updated version of the EfficientNet architecture: EfficientNet-lite [177]. This new family of models is based off the original EfficientNet architecture, with the SiLU being switched for ReLU activations. The authors retrained the models from scratch and, although their process is significantly more computationally costly, similarly to our results, they observed a slight accuracy drop from the activation function change. Consequently, as long as a small accuracy drop remains within the bound of a satisfactory accuracy for the end usage, switching to ReLU should be recommended prior to any further deep compression. During this PhD

[19] The one-armed bandit problem corresponds to a single player (agent) facing N playing machines (environment) and trying to determine which machine to play in order to maximize his or her gains.

[20] Let's consider n_i input channels, n_o output channels and a convolution of kernel size 3×3. The standard 2d convolutional layer will encompass $9n_in_o$ weight values, while the depthwise convolution will only require $9n_o$.

[21] The second part will be extensively discussed when we will tackle pruning in the dedicated chapter 2.

thesis, we conducted a similar study where we leveraged distillation to remove such activation functions (see Appendix A).

In the next section, we will introduce the first compression-only method of this manuscript. All the aforementioned methods are either multipurpose (e.g. NAS which can be performed with only the final accuracy in mind) or a pre-processing (e.g. switching the activation function). This first compression technique is based on knowledge distillation [91] and has been a core element of Datakalab's product line-up.

1.3.3 Knowledge Distillation for Deep Compression

Knowledge distillation [91] consists in leveraging the features learned by a given model, called the *teacher*, and using it to guide a second model, called the *student*. For example, the previously introduced method for activation function switch is a form of knowledge distillation where the teacher is the pre-trained model and student is a copy of the teacher where we made a small architectural change.

With regard to deep compression, knowledge distillation is usually seen as a way to improve the accuracy of a smaller model. For example, the EfficientNet and MobileNet v3 families were trained using a noisy student procedure [238]. The procedure is fairly simple. We use a teacher model trained on labeled images, in order to provide pseudo labels on unlabeled images. Consequently, we can train our student model (the target model that we will use) on the combination of labeled and pseudo labeled images. Similarly, other knowledge distillation methods have been proposed in order to offer higher accuracy for a student model with a lower inference cost, e.g. learning to infer on smaller images [64]. Recently, adapters [93] and LoRa [95] have gained a lot of traction as low-cost distillation methods for large language models. However, all these methods required full training processes and even the addition of unlabeled data to datasets that are already large. In order to circumvent this limitation, JITNet [155] introduced a new paradigm: online model distillation[22]. Let's consider a video stream, a teacher model labels a few images and the student model is fine-tuned on these frames. From a compression perspective, the resulting task is significantly simpler (less need for generalization capacities) and thus can be tackled by a smaller neural network. However, it comes with some drawbacks: the teacher inference cost, the need for training steps and the frame selection.

At Datakalab, Gabriel Kasser, Eden Belouadah and Arnaud Dapogny worked on adapting JITNet to object detection. They focused on the specific challenges, intrinsic to the method, such as the necessity to infer the teacher model in order to get the pseudo labels, the necessity to support back-propagation in order to optimize the student and the frame selection. Their solution involved the efficient design of a low-cost student model and continual learning tricks in order to avoid unintended overfitting. As a result, the team was able to achieve on par precision with a Yolo v8 X [213] (68.2M parameters) for car counting on the highway, using a student comprising only 100K parameters.

> The key takeaway from these achievements is the importance to define a precise task to address and leverage all the available[23]resources in order to craft a strong compressed baseline from which we will start our journey in the landscape of deep compression techniques.

In this PhD thesis, we decided to focus on post-training compression and acceleration. This decision was mostly motivated by the cost of large scale training. Although these costs have gone down and can usually lead to better compression rates, we managed to narrow the gap between post-training and general deep compression. Also, with the availability of the IDRIS [24] resources from the French government, we decided to scale up in model size rather than in terms of optimization computational requirements (batch size and training duration). This decision is inline with the rise of large language models [262] and diffusion models [29].

We focused on the two remaining compression techniques: quantization and pruning. Each of which corresponds to an intuitive way to compute an approximated prediction naively: simpler operations

[22] or train on the test set :)

[23] From an industrial perspective, the notion of available resources also includes the licensing. For instance, up until mid july 2023 the Llama [218] language models from Meta were not licensed for commercial purposes and as a result the best available cost-free LLM was Dolly v2 [46] from Databricks. However, the Llama licence changed for commercial purposes which in itself does not affect the model performance but made them available, thus changing the leader-board for private companies.

[24] The IDRIS is part of the Genci and provides resources for research in many fields, including AI. We were granted access to three projects including 50k hours of V100 compute, 20k hours of A100 compute and lastly 200k of A100 compute.

or fewer operations. In order to sort these methods, we propose to focus on the most effective first: Quantization.

1.4 Efficient Arithmetic: Quantization

In order to reduce the memory footprint and also accelerate any deep neural network, quantization replaces complex individual scalar multiplications by simpler ones. From a hardware and software perspective, a simpler multiplication is first defined by a lower bit-width [113]. The resulting weight values and intermediate features, represented on fewer bits, have a smaller footprint and require less individual bit operations. However, representation format change requires hardware support through proper data types and instruction sets [221]. For instance, in the most commonly used development language in AI (python), the only supported bit-widths are 1, 8, 16 and 32 bits. While some languages can handle any bit precision, hardware-level operations must also support these precision levels in order to leverage them at inference. For instance, a GPU A100 supports 1, 4, 8, 16 and 32 bits which excludes ternary quantization (2 bits) [164]. The second compression induced by quantization arises from replacing the native floating point representation by a fixed point representation [70]. Given a specific device, floating point operations may be better supported than fixed point operations (e.g. CPUs). However, it is important to note that while some hardware (e.g. GPUs) will better leverage fixed point operations, it is also the most appropriate format to support for efficiency [90].

In the following sections, we go through the details of floating point and fixed point explicit quantization processes.

1.4.1 Floating Point Quantization

In order to detail the explicit implementation of floating point quantization, we need to recall the definition of this representation format. Let's consider the default scalar value representation in the deep learning community: floating point on 32 bits. Generally speaking, a floating point value is defined by three sets of bits: one first bit to encode the sign, n bits for the significand also known as the mantissa or coefficient, and m bits for the exponent. Formally, given a real number a, its floating point encoding is given by its scientific notation, e.g. if $a = 398.2$ then, we write it $a = 3.982 \times 10^2$ where the mantissa encodes 3.982 and the exponent will encode 2. As a result, there is a trade-off between the size n of the mantissa and the size m of the exponent. Intuitively, more bits in the mantissa provides a finer grained grid in between exponent values, while more bits to the exponent enable to encode smaller and larger values in magnitude. In the international format (IEEE 754), the mantissa uses $n = 8$ bits and the exponent $m = 23$ bits.

Other format have been introduced and supported in recent years for deep learning libraries (e.g. TensorFlow and PyTorch). The most notable ones are the half representation (float 16) [185], the bfloat16 [212] and TensorFloat [166]. As a general rule of thumb, these formats can be used for inference from a model trained in float32 without further work. the most commonly adopted of these new formats is the half format. In this situation, we use $n = 5$ mantissa bits and $m = 10$ bits for the exponent, for a total of 16 bits. From our experience, its usage does lead to immediate and straightforward benefits: double the speed and half the memory footprint. It is worth noting that some operations remain on 32 bits, such as the loss computation and the gradients. However, this format offers limited support for functions such as the logarithm and exponential due to the low number of bits assigned to the exponent and thus limited range of values. On the other hand, the bfloat format addresses this limitation with its $n = 7$ bits for the mantissa and $m = 8$ bits for the exponent. However, this format offers a low precision given a fixed exponent, which can be troublesome during the final steps of the training when the learning rate and weight updates are very small. Consequently, TensorFloat was introduced to address the limitations from both the float16 and bfloat formats by using $n = 10$ bits mantissa and $m = 8$ bits exponent, leading to a 19 bits total. Because of this odd number of bits, its implementation has been mostly limited to TPU cores.

Before we move on to further floating point compression, we would like to take the time to highlight how to implement a training and inference using these simple formats in PyTorch and TensorFlow. The code blocks 1.1 and 1.2, below, contain all the necessary lines for half precision inference in PyTorch and TensorFlow, respectively. While both implementations are fairly simple, only the current[25] PyTroch implementation systematically works on Nvidia GPUs.

[25] We worked with torch 2.0.0 and tensorflow/keras up to 2.12.0.

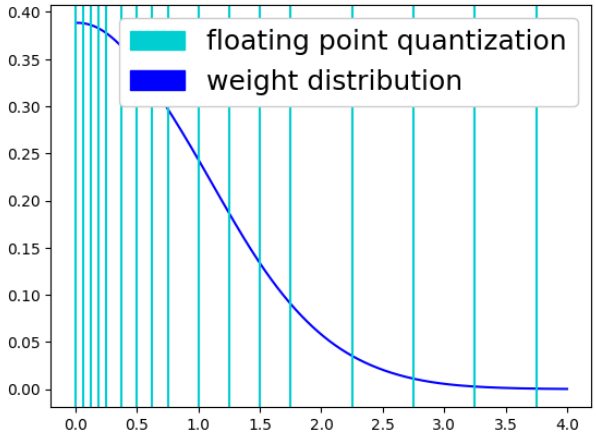


Figure 1.6: Low bit floating point quantization of a Gaussian distribution.

```python
import torch

model = ...
model = model.to(torch.half)
model.eval()
with torch.no_grad():
    with torch.cuda.amp.autocast():
        model(inputs)
```

Code 1.1: PyTorch-style half-precision implementation.

```python
import tensorflow as tf
import tensorflow.keras.mixed_precision as mp

mp.set_global_policy("mixed_float16")
model = ...
for weight in model.weights:
    weight.dtype = tf.half
model.predict(inputs)
```

Code 1.2: TensorFlow-style half-precision implementation.

In order to achieve further compression using floating point operations, other formats have been introduced using a total of 4 or 8 bits and diverse configurations of mantissa and exponent sizes. Most work, in the field [118], focus on the assignment of the appropriate split mantissa/exponent to each tensor. In Figure 1.6, we illustrate the encoding of a Gaussian distribution using floating point values. We observe that these formats offer piece-wise uniform encoding (contrary to fixed point, which offer a global uniform encoding). While, these formats offer greater compression rates as compared to the previous high resolution floating point ones, they also introduce a new challenge: the accumulation. To highlight this challenge, let's consider a matrix (our weights) $\tilde{W}$ and a vector $\tilde{X}$ (intermediate input feature), both quantized to 8 bits floating points. For any output neuron o, we have

$$\begin{aligned}(\tilde{W}\tilde{X})_o &= \sum_i \tilde{W}_{o,i} \times \tilde{X}_i \\ &= \sum_i \text{mantissa}(\tilde{W}_{o,i} \times \tilde{X}_i) \times \text{exponent}(\tilde{W}_{o,i} \times \tilde{X}_i).\end{aligned} \tag{1.6}$$

We can immediately guess that $\text{exponent}(\tilde{W}_{o,i} \times \tilde{X}_i)$ is most likely to overflow as the number of bits assigned to the exponent decrease. This is even worse for the mantissa, as the overflow is likely to arise from individual scalar multiplication as well as from the summation. Consequently, the default approach to this aspect of quantization consists in accumulating the results on a register using a higher precision (usually 16 or 32 bits). Consequently, the accumulated output is stored on an accumulator that has to be converted back to the appropriate low-bit format after the computation has been performed[26]. As a result, the usage of smaller quantization formats leads to extra burden: the accumulation and conversion steps, which in practice translates in a less straightforward estimation of the actual performance speed-up.

[26] While this thesis is not focused on hardware support and implementation, it is worth noting that, in practice, most inference engines will fuse the mat-vec nodes with the activation functions in order to perform these with higher precision. This is important even when considering the ReLU activation. While the mathematical output of the ReLU is bit-precision invariant, going down from 32 bits to 8 bits for instance and then performing a ReLU results in exactly 7 bits being used to encode the output. On the other hand, performing the ReLU before the bit reduction leads to an 8 bits output encoding.

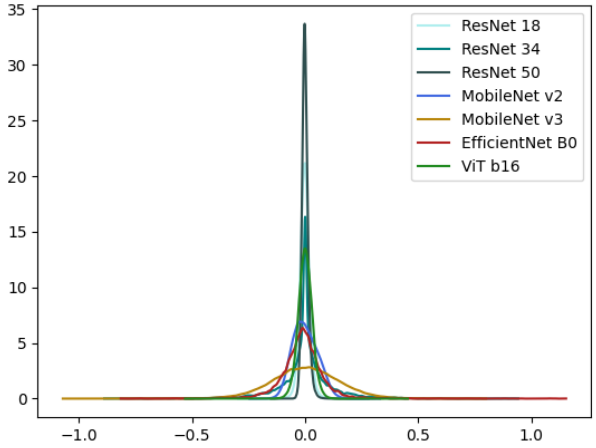


Figure 1.7: Weight density of layers randomly selected from the ResNet family, MobileNet v2, MobileNet v3, vision transformer (ViT) and EfficientNet B0.

While 32 bits floating point values (fp32 or just fp for full-precision) remains the default format for scalar values in deep learning, smaller formats such as float16 (half precision) and bfloat16 are growing in popularity based on their lossless mapping, which provides an immediate memory footprint reduction and speed-up. Stemming on these results, the compression community has been working on even more compressed formats by assigning the appropriate mantissa/exponent precision trade-offs for each tensors. However, these extreme low bit formats introduce extra challenges: overflow are increasingly likely as the bit-width goes down. This is addressed through high precision accumulation and conversions from high-precision to low-precision.

Floating point quantization has achieved strong results on a wide range of deep learning architectures and tasks. However, it suffers from two major drawbacks: first, it is computationally less effective than fixed point quantization [90] and, second, it leads to more accuracy degradation than fixed point quantization [118]. In the next section, we propose to discuss in detail how to perform fixed point quantization.

1.4.2 Fixed Point Quantization

Before detailing the quantization process, we need to sort out a confusing term that we have been using: fixed point quantization. In computer science, a fixed point representation [23] defines a real value a as a first integer n_a to be multiplied by an implicit scaling factor that is assumed to be shared across all numbers encoded in this format. However, in the quantization community, we assume that a fixed point representation is simply an integer representation. This quantization is often referred to as uniform quantization[27].

In order to obtain integer inputs, we usually do not have to spend too much time nor efforts as images are natively encoded on 8 bits as integers and tokenized text [107] is represented by 32 or 64 bits integers. However, these values are usually not encoded with the desired quantization bit-width. In practice, it is usually a bad idea to decrease the input precision if and only if this precision is already information optimal. Consequently, most quantization techniques do not further quantize the inputs (often called activations) of the first layer [113]. Still, if we were to uniformly quantize these activations, the optimal way to do so consists in performing a bit-shift of $B_1 - B_2$ bits where B_1 and B_2 are the number of bits of the initial and target representations respectively. Now, let's consider a mat-vec node and assume that the inputs of this layer are quantized. This assumption is straightforward for the first layer at least. For example, the first convolutional layer of a computer vision network that we want to quantize using 4 bits for the weight values and 8 bits for the activations. Such quantization is noted W4/A8[28]. In practice, most weight values follow a monomodal, almost symmetric distribution that can be modeled by a Gaussian. In Figure 1.7, we plot the density distribution of randomly selected weight values from

[27] It is worth noting that non-uniform quantization is not limited to floating point quantization.

[28] This is the standard notation and may hide further information: even if the first and last layers are not quantized, we will still note the quantization W4/A8.

different deep architectures. The crucial attributes of these distributions with regard to the quantization process are two-fold: first, their support is usually comprised in $[-0.5; 0.5]$ and second, their support varies a lot from layer to layer and architecture to architecture. As a result, in order to cover the entire span of values encoded by a 4 bit representation $\{-8, -7, \dots, 7\}$, we have to scale the weight values. Furthermore, we have to use a specific scale suited for each tensor. This second element highlights why we do not use the implicit scaling from the more general fixed point format. In practice, the baseline quantization method scales the weight values as a pre-process before inference such that the quantized counterparts exactly cover the quantization space, up to the smallest negative value. In other words, to quantize a tensor W on b bits, we apply the transformation

$$Q : W \mapsto \left\lceil \frac{W}{\text{scale}(W)} \right\rfloor \quad \text{with scale} : W \mapsto \frac{\max |W|}{2^{b-1} - 1}. \tag{1.7}$$

Consequently, the quantized weight values $Q(W)$ are encoded on b bits and the resulting quantized mat-vec node computes $Q(W) \times X$ where X is assumed to be quantized. However, similarly to floating point quantization, the product of two low bit representations will systematically overflow [161]. A problem that is solved using a 32 bits accumulator. On the other hand, contrary to floating point representation, we want to avoid the usage of 32 bits floating point during inference, as integer-to-integer and float-to-float conversion can be performed efficiently contrary to cross point conversions. In the context of fixed-point quantization, the standard for high precision to low precision conversion was introduced in 2018 [103]. Formally, let's consider the integer accumulator $Y = Q(W)X$ which is stored on 16 or 32 bits. The goal is to scale it down to $b = 8$ bits (or any other bit precision) using few operations. First, we scale up Y in order to ensure that it's highest value is encoded by a sequence of zeros followed by at least b ones. Second, we perform a bit-shift, such that the highest output value is a series of zeros with exactly b ones. The resulting 32 bits integer is then clamped to a b bits integer.

In summary, fixed point quantization offers better accuracy *v.s.* speed-up trade-offs than floating point quantization at the expense of a more complex process. Specifically, the accumulator requires a more sophisticated solution based on scaling and bit-shifts.

In its current form, quantization is difficult to work with, especially in current deep learning frameworks. In practice, the compression community mostly simulates the quantization process, in order to leverage existing optimization libraries such as PyTorch and TensorFlow.

1.4.3 Quantization Simulation

In order to simulate the quantization process using only the best supported formats (e.g. float32 and float16), we assume that we are given a neural architecture with weights and inputs that are not quantized yet. Let's consider the input tensor X and weight tensor W such that the initial mat-vec node computes $f : X \mapsto WX$. In order to simulate the quantization, we will need a quantization operator Q which maps any input tensor to a quantized space represented on b bits, such that $Q(X)$ and $Q(W)$ are the quantized inputs and weights respectively. However, contrary to the previously introduced quantization process (equation 1.7), in order to allow for further transformation of the weight and input values, we introduce the transformation t. Consequently, when working on quantization simulation, the general definition of the quantization of a tensor A is given by

$$Q : A \mapsto \left\lceil \frac{t(A)}{s(A)} \right\rfloor. \tag{1.8}$$

As a result, the quantized computation would be $f^Q : X \mapsto Q_X(X) \times Q_W(W)$ where the transformation t and scaling function s may change for each quantization operator Q_X and Q_W. However, this quantized computation is scaled such that quantized space is fully covered, which does not correspond to the full-precision range. In simple words, a trained neural network learns relative features in that two neurons that have a different impact based on their relative magnitude. In the current formulation of f^Q, this characteristic is not captured as two distinct neurons are both mapped to the same quantized space. In the previous sections, this element was handled by the accumulator and its conversion back to the appropriate bit-width, *i.e.* in the scaling of the accumulator. When the quantized inference is simulated, this is simply obtained with

$$f \approx f^Q : X \mapsto Q_W(W) \times Q_X(X) \times (s_X(X) \times s_W(W)). \tag{1.9}$$

This extra operation is called the de-quantization and often noted Q^{-1}, although this operation is not lossless due to the rounding step. In practice, this is often implemented with a very specific sequence as illustrated in code 1.3 and 1.4 for PyTorch and TensorFlow respectively.

```
def forward(
        self,
        input: torch.Tensor
    ) -> torch.Tensor:
    quantized_input = Q_X(input)
    qdq_input = deQ_X(quantized_input)
    quantized_weights = Q_W(self.weights)
    qdq_weights = deQ_W(quantized_weights)
    return torch.nn.functional.linear(
        qdq_input,
        qdq_weights,
        self.bias)
```

Code 1.3: PyTorch-style quantized simulation implementation.

```
def call(
        self,
        input: tf.Tensor
    ) -> tf.Tensor:
    quantized_input = Q_X(input)
    qdq_input = deQ_X(quantized_input)
    quantized_weights = Q_W(self.weights)
    qdq_weights = deQ_W(quantized_weights)
    output = tf.linalg.matmul(
        qdq_input,
        qdq_weights)
    return = tf.nn.bias_add(
        output,
        self.bias)
```

Code 1.4: TensorFlow-style quantized simulation implementation.

From these two code blocks, we can derive two observations that are of paramount importance. First, the biases are not quantized. This is due to the fact that they will be added to the accumulator during the actual inference. Consequently, during the simulation, we do not need to handle the bias bit precision as it shares the same precision as the accumulator. Second, the simulation of the quantization and de-quantization $Q^{-1}(Q(A))$ of a tensor A is performed once before the mat-vec operation. While this process, in itself, has no impact on the relevance of the simulation when we use the baseline quantization operator from equation 1.7, it may lead to inconsistencies between real inference and the simulated quantization. The most important one is the granularity of the quantization.

As we quantize and de-quantize the tensors, we use a scaling operation. However, we did not discuss the details of this scaling operation with respect to the dimensions of the tensor to scale. In order to make the appropriate design choice, we need to explicit the constraints: at inference, only the activations (*i.e.* the intermediate outputs of each layer) will be scaled during the accumulation type conversion. The intuition behind this constraint being that we do not want to perform a conversion after each individual scalar operation. Formally, if we were to use a scaling for each input neuron and a scaling for each output neuron, then the computations of a fully connected layer would read

$$W \times X = \begin{pmatrix} \sum_i Q_X(X_i) \times Q_W(W_{1,i}) \times (s_X(X_i) \times s_W(W_{1,i})) \\ \vdots \\ \sum_i Q_X(X_i) \times Q_W(W_{n_o,i}) \times (s_X(X_i) \times s_W(W_{n_o,i})) \end{pmatrix} \tag{1.10}$$

at inference. This implementation would lead to a lot of type conversions (as many as individual scalar multiplications) which would defeat the purpose of quantization, simply because the scaling factors are no longer factorized along the summation operation. As a result, for a given mat-vec node of output shape comprising n_o neurons (or feature maps), the scaling factors can be of any shape that divides the output shape. However, due to the limited hardware support for such operations, the scaling factor is usually limited to either 1 or n_o (number of output neurons) for the weights and limited to 1 on the activations[29]. These two granularities are often referred to as *per-tensor* (scalar scaling) and *per-channel* (scaling vector of size n_o). In practice, some researchers use a batch-normalization layer in order to implement the scaling operation in the per-channel setup. Such implementation are particularly effective for quantization-aware training.

A common mistake in the regular implementation of quantization simulation, is the use of per-channel quantization for both the weights and the activations. Intuitively, this offers a finer granularity, leading to higher accuracy than other quantization schemes. However, as we will thoroughly discuss in Section 3.1.2, such implementation does not translate in practice and lead to no speed-ups[30].

[29] During this thesis, we published an article on how to circumvent this limitation while remaining compliant with efficient inference [256]. We will go into these details in the quantization chapter.

[30] This mistake has been made, even in highly cited works [18]. Although, the cited paper has many other major contributions which overshadow this mistake.

A second design choice that may have a significant impact on the final accuracy of the quantized model is the order of the operations. Some implementations [127], propose to perform the quantization of the activations only after the mat-vec operation rather than before. This choice implies that a control over the simulated precision of the next node rather than the current one. As a result, for deep architectures using skip connections, when the first layer is not quantized (or quantized to the same precision as the model input encoding, e.g. 8 bits in computer vision), any results marked as W4/A4 are actually a mix of W4/A4 and W4/A4/A32 (or W4/A8 in computer vision) which makes them unfair to compare with simulations quantizing both the inputs and outputs. On the other hand, the proposed quantization simulation implementation does not suffer from this limitation. However, it does not quantize the model outputs, which only impacts regression models at a negligible cost.

Similarly, a third design choice is to focus only on mat-vec nodes without paying attention to the activation and merge nodes. With respect to activation functions, these operations are often fused, by the inference engine, at inference in order to be performed on the accumulator without loss in precision. This is quite straightforward for the ReLU activation, which behaves in the same fashion regardless of the quantization format (floating or fixed point). However, this is not so trivial for other activation functions such as the SiLU and GELU when they are not replaced. To circumvent this limitation, there are essentially two approaches. The first one consists in implementing support for such approaches like I-Bert [109] for a 32 bit integer-based GELU and softmax. The second approach, consists in adapting the weight values in order to compensate the information loss from the rounding of the activation function outputs [61, 156]. The optimization process of the weights in itself can either be performed using a small calibration set (see Section 3.2 in Chapter 3), post training or during training (see Section 3.3 in Chapter 3). On the other hand, merge nodes (such as add or concatenation), at inference, perform a quantization and de-quantization in order to set the inputs at the same scaling factor. During simulation, this is handled by the input quantization of the next layer. The only significant change occurs when we observe many skip-connections interconnected, as in DenseNets [97] or NasNets [272]. However, empirically, we observe that these nodes can remain not quantized. This does not lead to a significant accuracy drop as compared to the original accuracy. In summary, it is often sufficient to focus on the quantization of the mat-vec nodes only.

The fourth (and last) design choice that has been made in the quantization simulation is the rounding operation, which provides exact integer values at a cost: the resulting network has a zero derivative almost everywhere (when the derivative is not zero, it is undefined). Consequently, any stochastic gradient descent optimization on the quantized network cannot be performed in a straightforward fashion. Many works have been proposed in order to circumvent this limitation: gradient-based post training quantization (GPTQ) [156, 127, 228, 139] and quantization-aware training (QAT) [45, 22, 73]. Each of these subdomains of quantization have been thoroughly studied during this PhD thesis and have a dedicated section (sec 3.2 and sec 3.3 respectively). In short, GPTQ assumes a pre-trained neural network and optimizes the weight values such that the intermediate features and predictions of the full-precision model are preserved. This is achieved by using a soft version of the quantization (usually a sigmoid-like function) and the optimization is performed over a small calibration set (usually 1024 unlabeled data points). On the other hand, QAT aims at training a quantized neural network directly on the full training set. To do so, there are two main solutions to the zero derivatives challenge: simply omit the rounding step in the backward pass (straight-through estimation [22] or STE) or use a soft function to approximate the rounding step (similar to GPTQ).

In a nutshell, the quantization process is not well suited for the current deep learning libraries, thus we rather work with a simulated quantization process. While the use of a simulation enables a simpler compatibility with other deep learning work, it introduces its own set of assumptions and approximations: we only quantize the activations of each node. If one want to achieve finer simulation, they should also simulate the output quantization and quantize the merging nodes. However, in practice this is not necessary. The other significant challenge with simulated quantization: the zero derivatives almost everywhere which sparked research for specific solutions such STE and soft rounding.

In Appendix B, we propose several implementations of the simulated quantization of floating points, uniform quantization as well as other quantization schemes. Furthermore, if the reader is interested in TensorFlow/Keras implementations of the quantization of a convolutional neural network, you can read

this blog post on the matter on medium[31]. The reader can also follow the practical session[32] that we use for the advanced machine learning course at Sorbonne.

As we introduced the proper way to simulate quantization, it appeared that there are many remaining challenges in the domain of quantization that we ought to discuss in order to complete our landscape of compression techniques. In the following section, we highlight each of these challenges.

1.4.4 Quantization Challenges

As we quantize a deep neural network, several configuration choices can be made. Each of these choices lead to a different compression *v.s.* accuracy trade-off. The core challenges of quantization then consist in combining the strength of the different options. Explicitly, these challenges are:

- The **bit-width choice**: while hardware devices usually support a restricted number of bit-widths (1, 4 and 8 for non-FPGAs devices), intermediate representations (e.g. 6 bits) often lead to better trade-offs[33]. This challenge can be generalized as the search for the adequate bit-width for each individual computation. This is often referred to as *mixed-precision*.

- The **quantization format**: we saw that quantized space can be uniform (fixed point) or non-uniform (floating point). However, in the literature, uniformity refers to the mapping Q rather than the target grid space. In other words, some non-uniform mappings can map to a uniform grid (e.g. using power functions [255] or log functions [153]). These mappings often offer a tighter fit to the full-precision distribution at the expense of a more sophisticated inference. We worked a lot on the limitations of these methods, which we address in the sections 3.1.3 and 3.2.2.

- The **granularity**: as previously mentioned, quantization can be done per-tensor or per-channel on the weights and is always performed per-tensor for the activations. During this PhD thesis, we worked on a solution to achieve per-channel quantization on both the weights and activations without requiring a re-scaling at each scalar operation (see Section 3.1.2). Other granularities have also been proposed, such as group-wise quantization for large language models. In section 3.1.3, we will see the limitations of such granularity and how to avoid its usage.

- The **scaling factors**: while we proposed an explicit way to compute the scaling factors of the weights (using the maximum magnitude and the bit-width), we did not detail the procedure for the activations, which may vary for different inputs. There are many solutions to this problem, which we will discuss across the quantization chapter.

- The **data availability**: quantization can be performed either post-training or during training. These two setups lead to different performance trade-offs: QAT often achieves stronger compression rates at the expense of the training cost and access to data. In order to decrease the computational cost of the quantization pre-process and have a higher compliance with data privacy, data-free quantization was proposed. These methods are highly scalable with respect to the model sizes, but often struggle to achieve high accuracies at low bit-widths. Gradient-based post-training quantization was introduced as a middle ground and is performed on a subsampled set. During this thesis, we focused on both data-free and gradient-based post-training techniques, which we present in detail in the dedicated chapter along-side a smaller discussion in QAT in section 3.3.

This concludes our overview of deep neural network quantization. While these methods can offer significant improvements in terms of accuracy *v.s.* compression trade-offs, they require hardware support. Even when this support is available, further compression can be achieved using pruning and tensor decomposition, which constitute the last set of compression methods.

1.5 Trimming the Model: Pruning and Tensor Decomposition

While quantization leverages the intuition on faster runtime through simpler operations, pruning and tensor decomposition leverage the flip side of the coin: restricting the number of operations. In the following sections, we go through the landscape of pruning, which encompasses sparsity, pruning and

[31] https://medium.com/@ey_46515/neural-networks-and-edge-devices-34f6958cd54c

[32] https://gitlab.com/ey_datakalab/advancedmachinelearning_compression

[33] In REx [246], we figured a way to use hardware-friendly bit-precisions while enabling intermediate representations (e.g. the same expressivity as 6 bits).

tensor decomposition. The first distinction that we emphasize on is the distinction between pruning and sparsity.

1.5.1 Pruning and Sparsity

Pruning is generally defined as any technique that aims at removing operations from the computational graph of a deep neural network [124]. As such, pruning, similarly to quantization, can have different levels of granularity. Namely, the most commonly considered granularities are structured [12], semi-structured [92] and unstructured pruning [124].

Structured pruning has been extensively studied and is defined as any pruning that removes whole computational blocks, e.g. neurons and feature maps. In terms of tensors, for a given mat-vec node with weight tensor W this is equivalent to removing rows or columns in the case of fully-connected layers (W is a matrix) or removing output or input channels in the case of convolutional layers. In the current state of pruning, this is the most constrained setup and also the simplest to leverage as no work is required post-pruning. As compared to quantization and other pruning setups, the most important benefit from structure pruning is the absence of any requirement for a specific hardware support.

Unstructured pruning, or sparsity[34], has been the most studied pruning method and is also the most intuitive to work on: simply set some weight values to 0. The intuition is based on sparse matrix operations efficiency. There are three main sparse matrix multiplication configurations: sparse-dense algorithms [188], sparse-sparse [198] and dense-sparse ones [81]. Because, we only sparsify the weight values W in the product WX, we focus on sparse-dense algorithms. The intuition behind this is the difficulty to statically sparsify them. In other words, we do not want our sparsity mask to change for each input (dynamic) but rather we would like a fixed mask (static) for inference speed-ups which would be too detrimental for the accuracy. The core challenge of sparse matrix multiplication is the irregular data access. These accesses induce a lot of overhead on modern hardware which are designed for heavy parallelized computations rather than efficient memory access. Consequently, sparse inference leverages shared memory access reuse and efficient encoding. In the remainder of this work, we will leave this work to the inference engine. However, the reader should keep in mind that unstructured pruning is not straightforward to leverage and, generally, does not lead to linear speed-ups. As a general rule of thumb, 90% sparsity (*i.e.* 90% parameters set to 0) often leads to similar speed-ups as 50% structured pruning.

Lastly, semi-structured pruning has been introduced in [63] as a middle ground between structured and unstructured pruning. Let's consider a weight matrix W, a semi-structured pruning with format N:M aims at removing N scalar weight values in a subset of a row of W that comprises M values. the most commonly studied format being the $2:4$ pruning rate, which corresponds to the removal of 50% of the weight values. This format, like sparsity, requires storing the indices of the values to compute, which introduces memory overhead as compared to structured pruning. On the other hand, unlike sparsity, it offers better inference speed-ups on modern devices.

Generally speaking, the core challenge of pruning consists in the identification of the weight values that shall be removed. This is often addressed by either removing the least important weight values according to some criterion (*magnitude-based* pruning) or by removing redundant computations according to some similarity measure (*redundancy-based* pruning). Similarly to quantization, these criteria can be implemented either in a data-free manner, using a small calibration set or involving a full training. However, contrary to quantization, pruning is significantly less effective when performed outside of training. In other words, removing operations costs more than simplifying the operations, hence the need for re-training or fine-tuning post-pruning[35]. We distinguish essentially three approaches to pruning with fine-tuning:

- **Iterative magnitude pruning**: popularized in the lottery ticket article [65] in 2018, the core idea of iterative magnitude pruning (IMP) consists in training a neural network, select the least important neurons, deduce a trimmed down architecture, reset the weights using the same initialization sampling rule and repeat the process until convergence. In practice, the convergence is a pre-defined number of iterations, but could as well be defined a total number of parameters removed from the initial architecture. This pruning paradigm is computationally intensive as it requires multiple

[34] The terms of sparsity and pruning are often inter-changed, although they do not refer to the same concepts: pruning is more general and sparsity refers specifically to the unstructured pruning. In other contexts, sparsity can refer to the number of weights pruned regardless of the structure.

[35] It is quite a debate in the pruning community: how to perform fine-tuning for a fair comparison as it commonly accepted that most of the heavy lifting comes from this step [119], *i.e.* if the reader was to choose between a good pruning criterion and a good fine-tuning method, the reader should pick the fine-tuning method.

training phases. In the literature, we observed two major trends towards improving IMP: first, improve the weight selection [126]. Second, reducing the training cost through a more suited sparse optimization [59].

- **Pruning at initialization**: going further in the direction of the training cost reduction, pruning at initialization consists in considering an initialized model and derive the least important weight values immediately before sparse training. The core challenge being the selection among untrained values which are not encoding for any semantic representations yet. Still, empirical results show that the architecture of neural network itself is enough to sort which neuron to prune thanks to the symmetry *i.e.* because before training pruning the neuron 1 or neuron 17 of a layer is equivalent to pruning neuron 2 and 23 of the same layer, the only goal is to identify how to prune the layer level rather than the neuron identity. There is no consensus yet in the community with regard to pruning at initialization: *can we really obtain the same accuracy by training from scratch the same model as we get from post-training pruning*[36] *?* (yes [226] or no [67]). In the current state, the answer would be: *available PAI methods still underperform as compared to other pruning techniques.*

- **Post-training pruning**: based on the massive amount of available pre-trained neural networks, the standard approach to pruning consists in two steps: first, given an already trained DNN, identify the neurons to remove. Second, apply a fine-tuning step to the trimmed down model. Consequently, the community has followed two paths for post-training pruning improvement: either improve the fine-tuning step or the neuron selection. However, this trend has introduced a challenge: how to compare pruning methods? This question is non-trivial due to the overwhelming number of fine-tuning approaches [119]. This limits the conclusions that can be drawn from the pruning literature to *this combination of pruning criterion and fine-tuning outperforms this other combination,* which hinders the understanding of deep architecture critical operations as pruning criterion are not so often studied by themselves.

Nonetheless, pruning techniques have offered a wide range of importance measurement techniques each applied to different pruning protocol leading to significant inference speed-ups and industrial projects with [116] paving the way. However, similarly to quantization, in order to comply with deep learning libraries such as PyTorch and TensorFlow, researchers and enthusiasts often simulate pruning. In the following section, we present the best practices for pruning simulation.

1.5.2 Simulated Pruning

Contrary to quantization, it may appear counterintuitive to simulate pruning rather than simply using it. While this intuition is correct at inference for structured pruning, this is false for any other situation. First, during any training, it is important to note that current deep learning frameworks are based on a computational graph. In a nutshell, the main programming language (often python) interprets the code and translates it into a computational graph. There exists essentially two approaches: static and dynamic computational graph. The first one defines the graph and then runs data through it, e.g. in TensorFlow. The second one,, dynamic computational graph defines the graph has it is run, e.g. in PyTorch. In either case, a change in the computational graph induces significant overhead, which explains why we would rather define a graph that can simulate any pruning over a frequent graph edition (this is also important for the aforementioned super networks in NAS). On the other hand, outside of training, sparsity, or unstructured pruning, is challenging to leverage and as such is only simulated in order to estimate the resulting accuracy.

We distinguish two main ways of simulating pruning: masking and zero hard-coding. Masking consists in apply a binary mask M to a given weight tensor W. The core advantage of this formulation is the simplicity to perform a soft masking that can be learned. During training, the masking is implemented as, $\sigma(M) \times W$ where σ is the sigmoid activation function. During the simulated inference, we compute $\lceil \sigma(M) \rfloor \times W$. On the flip side, zero hard-coding consists in directly replacing weight values with zeros. The limitation with this approach arises from the optimization process, which is additive *i.e.* weight values are updated with $W \leftarrow W + \delta$. Consequently, we need to enforce the zero values to the gradients update or reset these values after every optimization step. While this introduces a computational overhead, it is more memory efficient, as we do not need to store the masking tensor. Masking and zero hard-coding are both valid simulation choices, and there is no specific conceptual pitfall to either one.

[36] It is worth noting here, that post-training literally means after the training and contrary to its usage in quantization, does not suggest a less costly process over a small set.

However, in some cases, they do share a common issue regarding the relevance of the simulation with respect to actual inference performance. In order to highlight this error, let's consider a popular architecture design: the skip connection. Let W_1 and W_2 be two weight tensors such that there exists an intermediate feature Y computed from intermediate inputs X_1 and X_2 as $Y = W_1X_1 + W_2X_2$. Without loss of generality and for the sake of clarity, we will use the masking notation. Then, their pruned counterpart would read as $\tilde{Y} = \sigma(M_1)W_1X_1 + \sigma(M_2)W_2X_2$. While this implementation will run in any deep learning library and does not represent any problem in sparse setups, it shows its flaws with the structured pruning. In structured pruning, we want to actually remove output neurons from the layers and in our example, we want to remove neurons in W_1X_1 and W_2X_2. However, let's assume that we removed the first neuron in W_1X_1 but not in W_2X_2. As a result, the first neuron of $\tilde{Y}$ will be equal to the first output neuron of W_2X_2 and thus cannot be actually pruned. To support such situations, one has to pad the outputs, which can be quite costly as the padding is not performed on contiguous values in memory as we only pad the pruned neurons. In order to avoid this pitfall, it is best practice to prune the two mat-vec nodes as if it was one node, as follows $\tilde{Y} = \sigma(M_{\text{shared}})W_1X_1 + \sigma(M_{\text{shared}})W_2X_2$.

In summary, we have two main simulation options: masking and zero hard-coding. Each offer advantages: masking for latency and zero hard-coding for memory footprint. Overall, pruning is quite straightforward to simulate as long as we are meticulous with skip connections and ensure that a pruned neuron will actually be pruned at inference.

Due to its simplicity, pruning has been the favored but not only child in the DNN compression community. The other set of compression techniques that can effectively remove parameters and computations is called tensor decomposition. In the following section, we propose to present an overview of this last set of compression techniques.

1.5.3 Tensor Decomposition

Contrary to pruning, Tensor decomposition does not aim at removing operations directly, but rather propose to re-formulate these operations in a more parameter efficient fashion (which may result in indirect speed-ups). Historically, the first tensor decomposition method was the generalization of singular value decomposition [165, 1] (SVD) to high dimensional tensors. In practice, all the intuitions from the study of matrices hold for deep learning tensors. Consequently, for the sake of simplicity, we will assume that the considered tensors are matrices. Let's consider a weight tensor W, its corresponding singular value decomposition is a (U, D, V) triplet such that D is a diagonal tensor and for any X, we have $UDVX = WX$. In this formulation, if W is a matrix of size $n \times m$, then (U, V, D) defines $n \times n + m \times m + \min(m, n) > n \times m$. The parameter reduction occurs when the rank r of the diagonal matrix D is small enough, then the tensor decomposition becomes $WX = (U\sqrt{D}) \times (\sqrt{(D)}V)X$ and requires $r \times (n+m)$ parameters. Consequently, the criterion for the SVD usage is: $r < \frac{nm}{n+m}$. In practice, this condition is not often verified. However, forcing the removal of some eigenvalues, *i.e.* artificially lowering r, offers a strong baseline for fine-tuning and often leads to good compression rates [15, 237, 266]. Intuitively, this bears similarities with DNN pruning as we remove expressivity from the network: in pruning we remove neurons, in tensor decomposition we remove singular values which correspond to a linear combination of neurons.

Other, more sophisticated, tensor decomposition have been proposed as extensions of the low rank SVD approximation. The first example of such decomposition is the Canonical polyadic decomposition CANDECOMP/PARAFAC, also known as the CP-decomposition [214]. The key idea is to approximate a k-dimensional tensor W as an outer-product (noted $\otimes$) of one dimensional factors $w_1, ..., w_k$, such that $W \approx w_1 \otimes \cdots \otimes w_k$. The resulting inference is computed as $W(x_1, ..., x_k) \approx w_1x_1 \cdots w_kx_k$. The resulting layer comprises $n_1 + \cdots + n_k$ parameters instead of $n_1 \times \cdots \times n_k$. However, in practice, such compression is too brutal and leads to significant accuracy drops. Several methods have been proposed in order to find the appropriate initialization of W in a sum of r CP-decomposition [122, 13], *i.e.* $W \approx \sum_{i=1}^{r} w_{i,1} \otimes \cdots \otimes w_{i,k}$ and $W(x_1, ..., x_k) \approx \sum_{i=1}^{r} w_{i,1}x_1 \times \cdots \times w_{i,k}x_k$. The second example is the tucker decomposition, which formulates the 3-dimensional tensor W of size $n_1 \times n_2 \times n_3$ using one 3-dimensional tensor w_0 with fewer parameters $d_1 \times d_2 \times d_3$ with three 2-dimensional tensors w_i of size $n_i \times d_i$. The re-composition uses the mode product, which is slightly different from the outer product that is used in the CP-decomposition. Mode product computes all the combinations possible from the CP-decomposition, in other words, instead of doing $\sum_{i=1}^{r} w_{i,1}x_1 \times \cdots \times w_{3,k}x_3$, we do $\sum_{i_1=1}^{r_1} \sum_{i_2=1}^{r_2} \sum_{i_1=3}^{r_3} w_{1i_1}x_1 \times \cdots \times w_{3i_3}x_3$. This tensor decomposition is also very effective in terms of memory (as effective as the CP

decomposition) but computationally more costly. Nonetheless, it appears that tucker decomposition offer the best performance trade-offs in practice as compared to CP-decomposition. Unfortunately, these tensor decomposition techniques have not been thoroughly tested on modern architectures and struggle to spark a new research interest.

On the other hand, simpler and more benchmarked decomposition have emerged in the past five years: separable layers. Separable layers include mixer layers (from the MLP-mixer architecture [215]) and depthwise separable convolutions [38]. For the sake of simplicity we will focus on the latter as the former is derived from it. Let's consider a convolutional layer f with a 3×3 kernel. The weight values W perform two operations simultaneously: mix the spatial information (non-scalar kernel size) and mix semantic information (summation over the input channels). Then, depthwise separable layers split these operations in a sequence: a first layer (depthwise convolution) takes each input map individual and applies a single 3×3 kernel which mix the spatial information. Then a second layer performs the semantic mixing using a standard 1×1 convolution, often called a pointwise convolution. The resulting two layers perform the two operations with $n_i(n_o + 9)$ parameters instead of $9n_in_o$. In practice, such layers have been leveraged in efficient architectures such as the EfficientNets [210] and MobileNets [183, 94]. On the other hand, from a pre-trained convolutional model, the conversion from standard convolutions to such separable layers requires an approximation. The core idea [252] consists in extracting a basis from the 3×3 kernels and use this basis as weights for the depthwise layer. Then, the representation of the original weights in said basis defines the new pointwise values. While this can work well on some deep neural networks, its impact is limited to older architectures such as ResNet 50 that do not already use depthwise convolutions or self-attention modules.

In a nutshell, tensor decomposition can be summarized in four main techniques: the old-fashioned SVD which offers the best performance on modern architectures, the CP-decomposition and tucker decomposition which leverage outer products, and separable layers that are best suited for architecture design rather than post-training compression.

As we just saw, while tensor decomposition has shown little benefits on modern architectures to the exception to the old-fashioned SVD [237] which leads us to say that this field has been abandoned by the community either due to the low maximum reward or the difficulty to propose new methods. Consequently, in the following section, we propose to wrap up our study of trimming methods on a discussion on the most promising challenges that are introduced by pruning.

1.5.4 Pruning Challenges

Based on the definition of the current pruning techniques, several challenges remain open. Some of these challenges have been addressed by the compression community, some have been moved up toward resolution during this thesis, and some remain open. In our opinion, these challenges are:

- The choice between **redundancy *v.s.* importance** pruning: These two paradigms rely on two orthogonal assumptions. Importance-based pruning assumes that deep neural networks learn essential operations and less relevant ones. On the flip side, redundancy-based pruning assumes that despite the efforts to break symmetry at initialization, deep neural networks learn redundant operations. During this thesis, we worked quite extensively on this dichotomy. In short, importance-based pruning usually vastly outperforms redundancy-based pruning. We will discuss this result throughout the next chapter.

- The limitations of **the evaluation metrics**: we mentioned two major metrics for the inference practical speed: *latency* and *throughput*. On the one hand, these metrics are not suited for simple and fair comparison, as previously discussed. On the other hand, the metrics used in pruning, namely the *number of parameters* and *number of flops* removed, do not correlate well with the actual speed-up (although they do correlate well with the memory footprint reduction and power savings in the case of structured pruning). To mitigate this limitation, some works propose alternatives to the standard pruning metrics that offer better indicators [187]. In the last year of this PhD, we supervised a brilliant intern that worked on improving such methods (we will discuss his findings in section 2.5).

- The relevance of **pruning criteria**: Pruning criteria aim at sorting out the weights and neurons by their importance with respect to the predictive function. In practice, it is fairly simple to find the

very least and very most important neurons of a layer, e.g. on ImageNet, it would be the 20-25% least and most important neurons. In other words, the search for the extreme neurons is a simple one. The challenge arises from ranking the remaining weights which have a medium importance. This is one of the aspects over which we worked the most and achieved significant improvements [254] (more details in section 2.3).

- The **pruning budget**: assuming a given pruning paradigm and pruning criterion, the remaining practical question is: *how many neurons should we try to remove?* We rather ask this way rather than what accuracy should we aim for, as it is simpler to anticipate the size of a model than its accuracy. Let's assume that we need to remove 40% of the parameters of the model in order to fit in memory on a device. How many neurons should we remove in each layer? To the best of our knowledge, there is no consensus on that matter. Two major trends are observed: first, using a heuristic in order to split the pruning goal, second, using the pruning criterion applied to a layer as a relative estimation of the layer overall contribution. In this thesis, we showed that both approaches are inadequate, and we will share a better solution in section 2.4.
- Alternative **granularity levels**: While unstructured and structured pruning remain the two most studied format, semi-structured pruning has recently gained a lot of traction. In its current form, it is best suited for GPUs. Other formats adapted to other hardware and/or inference algorithms could offer better trade-offs in terms of inference speed-ups *v.s.* to accuracy. In section 2.2, we present a semi-structured format that we introduced during this thesis, which suits memory efficient devices such as FPGAs.

This concludes our landscape of deep compression techniques. We went through the first and foremost step: select the appropriate architecture which offers the best initial trade-off in terms of accuracy *v.s.* speed. Second, the most promising compression specific technique: quantization. Third and last, we presented all the commonly studied trimming techniques: tensor decomposition and pruning. During this thesis, we worked on all three aspects of quantization (data-free, gptq and qat) as well as several aspects of pruning and tensor decomposition. For the remainder of the manuscript, we propose to start the presentation of our contributions by pruning. This schedule also matches our research schedule for the last three years.

Chapter 2

Deep Neural Network Pruning

At the beginning of this PhD thesis, we worked on data-free pruning and tensor decomposition. The motivation for this approach was to avoid expensive training procedures and compliance with data privacy rights (GDPR). To achieve such constrained compression we designed a method combining weight hashing, redundancy based pruning and tensor decomposition. This led to the publication of two articles RED [252] and an extension RED++ [253]. In order to further push the achievable compression rate, we, later, worked on data-driven pruning and in particular on importance criterion which led to the publication of SInGE [254]. Finally, we adapted the proposed pruning techniques to solve other problems than compression, such as robust inference [251] and fighting overfitting [208].

Generally speaking, there are two major categories of pruning techniques: redundancy-based and importance-based. The latter assumes that some weights of the trained neural network do not contribute much to the decision-making process and as such can be removed. On the flip side, the former methods assume that neural networks learn redundant features, which can be merged with little loss of information. During this thesis, we contributed to both fields. In the first section of this chapter, we propose to start as this PhD started, on redundancy based pruning.

2.1 Redundancy-based Approaches

Let's consider a mat-vec node f with weight tensor W and bias tensor b such that f computes $f : X \mapsto WX + b$. Each output neuron n_i, out of n neurons, is defined by the scalar product $\langle W_i, X \rangle + b_i$. The goal in redundancy-based pruning is to identify and remove redundant neurons.

Neuron Similarity Consequently, we define a distance between two neurons n_i and n_j, e.g. the distance between (W_i, b_i) and (W_j, b_j). However, the contribution of the biases b_i and b_j is complex to estimate with respect to the contribution of the weight tensors W_i and W_j. We will show why this is a constraint that we cannot relax.

Let's consider an activation function σ which could be any popular activation function such as the ReLU, SiLU, GELU, sigmoid or softmax. The property at hand in this situation is the non-linearity: $\sigma(x+y) \neq \sigma(x)+\sigma(y)$. Consequently, we get that $\sigma(\langle W_i, X\rangle+b_i) \neq \sigma(\langle W_i, X\rangle)+\sigma(b_i)$ which induces that a similarity between the weight values $W_i \sim W_j$ does not necessarily lead to a similarity of the output neurons n_i and n_j post activation function. In other words, if the biases are too distinct, we cannot approximate the neurons n_i and n_j using only one neuron. This is why, theoretically, our criterion ought to account for the bias values. Fortunately, in practice the bias values of similar weight values are also similar in value, which simplifies the definition of the distance between two neurons. The choice of such distance is at the core of redundancy-based pruning methods [195, 110, 252, 253]. Before dissecting the state-of-the-art redundancy based pruning techniques, including the ones that we proposed during this thesis, let's detail how the architecture is actually edited with such methods.

Redundancy Elimination Assume that neurons n_i and n_j are considered similar according to a given similarity criterion. The pruning process is called *neuron merging* and consists in defining a new neuron n'_i of weight values W'_i and bias b'_i. The new neuron is supposed to provide a good approximation of the

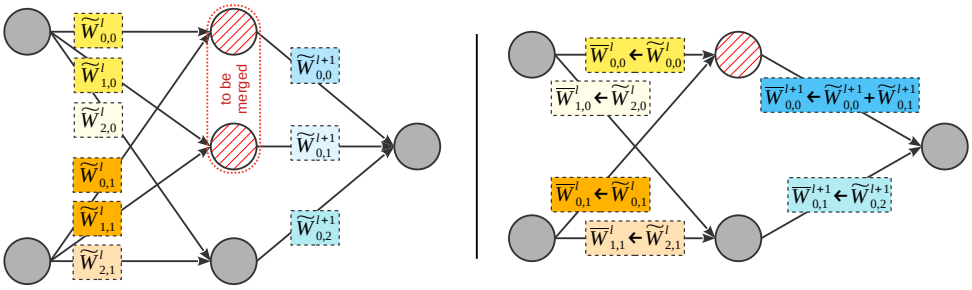


Figure 2.1: Neuron merging in the case of a fully-connected layer with weights $\tilde{W}^1$. Similar colors indicate equal weight values, e.g. $W_{0,0} = W_{1,0}$. The pruned network weights $\bar{W}$ are obtained by merging the first 2 neurons of layer l and simply summing the corresponding weights in layer $l+1$.

two neurons n_i and n_j. The new layer f^* computes

$$f^* : X \mapsto \begin{pmatrix} W_1 \\ \vdots \\ W'_i \\ \vdots \\ W'_i \\ \vdots \\ W_n \end{pmatrix} X + \begin{pmatrix} b_1 \\ \vdots \\ b'_i \\ \vdots \\ b'_i \\ \vdots \\ b_n \end{pmatrix} \approx \begin{pmatrix} W_1 \\ \vdots \\ W_i \\ \vdots \\ W_j \\ \vdots \\ W_n \end{pmatrix} X + \begin{pmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_j \\ \vdots \\ b_n \end{pmatrix}. \tag{2.1}$$

If we now remove the j^{th} row in the new weight tensor W' and bias vector b', the pruned function $\bar{f}$ will output the exact same total information as f^* using $n-1$ neurons instead of n. Then the core idea consists in update the subsequent mat-vec nodes directly connected to f such that $g \circ \bar{f} = g \circ f^*$:

$$\begin{cases} g \circ f^* &= \begin{pmatrix} W_{g1}{}^T \\ \vdots \\ W_{gn}{}^T \end{pmatrix}^T \sigma \left(\begin{pmatrix} W_1 \\ \vdots \\ W'_i \\ \vdots \\ W'_i \\ \vdots \\ W_n \end{pmatrix} X + \begin{pmatrix} b_1 \\ \vdots \\ b'_i \\ \vdots \\ b'_i \\ \vdots \\ b_n \end{pmatrix} \right) + b_g \\ g \circ \bar{f} &= \begin{pmatrix} W_{g1}{}^T \\ \vdots \\ W_{gi}{}^T + W_{gj}{}^T \\ \vdots \\ W_{gn}{}^T \end{pmatrix}^T \sigma \left(\begin{pmatrix} W_1 \\ \vdots \\ W'_i \\ \vdots \\ W_n \end{pmatrix} X + \begin{pmatrix} b_1 \\ \vdots \\ b'_i \\ \vdots \\ b_n \end{pmatrix} \right) + b_g \end{cases} \tag{2.2}$$

Put in simpler words, as we compute twice the input neuron of the subsequent layer, this is equivalent to the summation of the subsequent weight values along the input dimensions[2]. For the sake of clarity, the process is illustrated in Figure 2.1. The only question that remains now is: how to properly measure neuron similarity?

Similarity Criterion In the literature [195, 110], the baseline approach consists in simply performing an l_2 distance measurement between neurons and merging together the pairs that fall under a certain

[1]This notation comes from RED where the weights are slightly changed before pruning, hence the new notation. This change comes from weight hashing, which we discuss in the next paragraphs.

[2]In the equation, we represented the input dimension using the transpose notation in order to make it fit on a single line.

threshold. The merged neuron value is then computed as the barycenter of the clustered neurons. Intuitively, we could directly use clustering techniques such as k-means if and only if we knew in advance the number k of neurons to keep. More generally speaking, other clustering techniques do not have an automated hyperparameter selection dedicated to deep neural network compression. In practice, in the data-free setup, we cannot empirically set these parameters. Consequently, we designed a method which bounds the accuracy degradation from the neural network pruning automatically.

Weight hashing In RED [252], we proposed a novel way to cluster weight values. The core idea is to restrict the number of distinct weight values within each layer independently. Intuitively, the semantic output of each neuron of a given layer are inter-dependent. In other words, the magnitude of a neuron n_i is relevant only with respect to the magnitude of the other neurons of the same layer. On top of this, we work in a data-free setup. Based on these constraints, the problem consists in the design of an effective hashing function. Such functions map a continuous set of values to a discrete one [4]. The core challenge consists in creating as many identical neurons as possible without degrading the predictive function. To do so, we designed the hashing function from the weight density distribution. Consequently, the first step consists in extracting the density distribution $\mathfrak{f}$ of the weight tensor W of a layer f. This can be achieved using a kernel density estimation method to a sampled distribution. In our case, the sampled data are the weight scalar values and the kernel K is Gaussian. Consequently, we get an explicit formulation for the estimated density $\tilde{\mathfrak{f}}$:

$$\tilde{\mathfrak{f}} : \omega \mapsto \frac{1}{\#W\Delta} \sum_{w \in W} K\left(\frac{\omega - w}{\Delta}\right) \tag{2.3}$$

where $\#W$ refers to the number of scalar weight values in the tensor W and Δ is the bandwidth of the Gaussian kernel, which remains to be defined. Thus, the only remaining element to determine in order to have a data-free weight hashing function is the bandwidth Δ. In RED [252], we proposed to use the median of the differences between consecutive weight values. Formally, let's consider the scalar weight values $w_1, \ldots, w_{\#W}$ ordered such that $\forall i < j \; w_i < w_j$. The differences d_i of consecutive weight values are defined as $d_i = w_{i+i} - w_i$. Then the bandwidth Δ is set to the median value of the differences (d_i).

From the given estimated density function $\mathfrak{f}$, we extract the local extrema. Intuitively, we want to keep the most *meaningful* values, which we approximate with the likeliest values in the weight distribution. Let's note the ordered local minima m_k and maxima M_k. The likeliest values M_k will define the new pool of scalar weight values, while the local minima will define a partition of the real numbers for our hashing function $\mathfrak{h}$. Formally, we define $\mathfrak{h}$ as

$$\mathfrak{h} : \omega \mapsto \sum_k M_k \mathbb{1}_{\{\omega \in [m_k; m_{k+1}[\}} \tag{2.4}$$

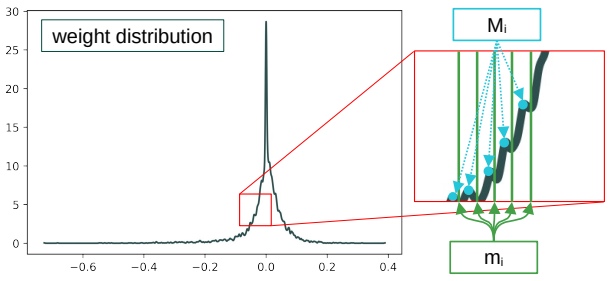


Figure 2.2: Weight kernel density estimation of a layer of a ResNet 50 trained on ImageNet. On the right side, we zoom in and highlight the local minimum (vertical green lines) and the local maxima (blue dots). Every value between two consecutive vertical green lines is set to the corresponding blue dot.

The partition and values selected by the hashing function are illustrated in Figure 2.2 for a layer of a ResNet 50 trained on ImageNet. The resulting hashing method is adaptive to the weight distribution and does not require hyperparameter tuning. While this hashing method plays the role of introducing strict redundancies among weight values, it also diminishes the number of distinct weight values. The hashed neural network can be stored on the device using less disk space with the Huffman coding [100]).

Weight Merging in RED In RED, we proposed to merge the identical neurons post-hashing. This protocol offers better performance than direct neuron merging based on the Euclidean distance, as we will show in the experiments section. However, this method offers limited flexibility, as for a given neural network the hashing function is unique and so is the pruned model. We proposed a soft relaxation of the merging criterion. We introduced a per layer parameter α_l to merge the $\alpha_l\%$ closest hashed neurons that shall be merged.

After the publication of [252], we figured another improvement over the standard redundancy-based data-free pruning: the angular distance. Instead of merging neurons based on the Euclidean distance, we can merge them based on their angular distance. If two neurons are collinear and similarly oriented

(*i.e.* positive angle) then they can be merged if and only if the deep neural network uses ReLU activation function. Formally, if we have a strict collinearity between neuron n_i and n_j then $\lambda W_i = W_j$ and $\lambda b_i = b_j$ then for $\lambda > 0$:

$$\sigma(\langle W_i; X\rangle + b_i) + \sigma(\langle \lambda W_i; X\rangle + \lambda b_i) = (1+\lambda)\sigma(\langle W_i; X\rangle + b_i). \tag{2.5}$$

This is a usage example of the previously mentioned weak form of compliance with operation folding of the ReLU activation function (see section 1.1.3). Consequently, we can merge neurons that have a low angular distance, which offers even greater compression performance than that of RED.

In summary, redundancy-based pruning is based on two components: neuron merging and a similarity measure. In this thesis, our contributions were published in [252] and consist in the introduction of a novel weight hashing to improve the similarity measurements. Following this work, we also proposed a better similar measurement for ReLU networks based on the angular distance rather than the euclidean distance.

In the journal extension [253], we theorized an upper-bound over the prediction errors introduced by the hashing mechanism. This offers strong theoretical grounding to the method and further motivates its use. In the following section, we propose to dive into this result.

2.1.1 Theoretical guarantees

For the sake of simplicity, let's first, consider a single fully-connected layer f with weight tensor $W \in \mathbb{R}^{n_1 \times n_0}$. We will extend our results to a full ReLU-based neural network afterward. The goal of the upper bound is to bound the average error on intermediate features of a single layer. We note the hashed weight values $\tilde{W}$ using the aforementioned method. Then, we can bound the absolute error between each individual scalar weight values as $|w - \tilde{w}| \leq \min_k\{m_k > w\} - \max_k\{m_k < w\}$ based on the local minima (m_k). Consequently, based on the distribution $\mathbb{P}_w$ of the weight values W, we get the following upper-bound U_{part} on the expected absolute scalar error:

$$\mathbb{E}_{\|x\|\leq 1}[|wx - \tilde{w}x|] \leq \sum_k (m_{k+1} - m_k) \int_{m_k}^{m_{k+1}} \mathbb{P}_w dw = U_{\text{part}} \tag{2.6}$$

This results gives us a first bound from the partitioning that is performed during hashing. In order to get a finer grained upper bound, we leverage properties of the density estimation process itself. The final upper-bound U will then be a combination of these two upper-bounds. Because we are using a kernel density estimation based on a Gaussian kernel, in a given interval $[m_k; m_{k+1}]$, the density if monomodal. Consequently, we can use the fact that for any monomodal distribution, the average absolute distance of a random sample to the distribution mean is $\sigma_k\sqrt{2/\pi}$. Thus, the triangular inequality gives:

$$\mathbb{E}_{\|x\|\leq 1}\left[|wx - \tilde{w}x|\right] \approx \int |w - \tilde{w}| x \mathfrak{w}(w) dx \leq \max_k \sigma_i \left(\sqrt{\frac{2}{\pi}} + \sqrt{3}\right) \leq \frac{\Delta}{\sqrt{2\pi}}\left(\sqrt{\frac{2}{\pi}} + \sqrt{3}\right) = U_{\text{monomodal}} \tag{2.7}$$

Explicitly, we now have the two upper-bounds U_{part} and $U_{\text{monomodal}}$ which enable us to define our upper-bound U as

$$\mathbb{E}_{\|x\|\leq 1}\left[|wx - \tilde{w}x|\right] \leq U = \min\left\{\frac{\Delta}{\sqrt{2\pi}}\left(\sqrt{\frac{2}{\pi}} + \sqrt{3}\right), \quad \sum_k (m_{k+1} - m_k)\int_{m_k}^{m_{k+1}} \mathbb{P}_w dw\right\} \tag{2.8}$$

The challenge that remains is to generalize this upper-bound to a full network.

To do so, we exploit several properties. First, as we study the behavior of the average error, we use the Central Limit Theorem which gives us the multiplicative term $\frac{1}{\sqrt{s_{\text{in}}}}$, where s_{in} is the input size, to generalize the scalar error $|wx - \tilde{w}x|$ to the neuron error. Second, assuming a ReLU activation function also enables us to erase about a half of the values from the equation as they would be zeroed out (this is empirically observed on modern ReLU architecture). Fortunately, we can leverage batch normalization layers in order to obtain a finer grained estimation of the average number of values that will be zeroed out by the ReLU function. Let β and γ be the expected layer mean output and standard deviation output respectively, then we can add a multiplicative term $\frac{1}{2}\left(1 - \text{erf}\left(\frac{-\beta}{\gamma\sqrt{2}}\right)\right)$ where erf is the Gauss

Table 2.1: Evaluation of our adaptive hashing in term of % removed weight values (% reduction) and test accuracy drop compared with a uniform baseline.

Cifar10												
Architecture		ResNet			W. ResNet		MobileNet v2			EfficientNet		
Model		20	56	164	28-10	40-4	0.35	1	1.4	B0	B4	B7
% Reduction	uniform	98.0	98.4	98.9	99.7	99.8	98.4	98.6	99.0	98.5	98.9	99.2
	adaptive	98.9	99.0	99.1	99.9	99.9	96.7	99.4	99.6	99.4	99.7	99.9
% Accuracy drop	uniform	0.58	0.45	0.89	0.72	0.68	**0.00**	0.00	0.00	0.00	0.01	**0.00**
	adaptive	**0.07**	**-0.03**	**0.01**	**-0.02**	**0.00**	**0.00**	**-0.01**	**-0.04**	**-0.02**	**0.00**	**0.00**

ImageNet										
Architecture		ResNet			MobileNet v2			EfficientNet		
Model		50	101	152	0.35	1	1.4	B0	B4	B7
% Reduction	uniform	99.6	99.5	99.5	21.2	24.7	22.0	98.5	98.9	99.2
	adaptive	99.9	99.9	99.9	22.0	61.3	97.4	72.3	87.8	97.8
% Accuracy drop	uniform	**0.00**	0.01	**0.00**	3.96	5.79	8.19	3.36	3.85	3.30
	adaptive	**0.00**	**0.00**	**0.00**	**0.25**	**0.60**	**0.10**	**0.44**	**0.00**	**0.01**

error function, $\text{erf} : z \mapsto \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$. Third, we use the error propagation properties of piece-wise affine compositions. Therefore, we get the following global upper bound in the whole network

$$U = \prod_{l=1}^{L} \left(\frac{\min \left\{ \frac{\Delta_l}{\sqrt{2\pi}} \left(\sqrt{\frac{2}{\pi}} + \sqrt{3} \right), \sum_k ((m_{k+1})_l - (m_k)_l) \int_{(m_k)_l}^{(m_{k+1})_l} \mathbb{P}_w dw \right\}}{2\sqrt{\text{s}_{\text{in}l}}} \left(1 - \text{erf}\left(\frac{-\beta_l}{\gamma_l \sqrt{2}} \right) \right) + \beta_l \right) - \prod_{l=1}^{L} \beta_l. \tag{2.9}$$

Intuitively, we can observe that as Δ converges to zero, so does the upper-bound. For instance, we get the expected result when no hashing ($\Delta = 0$) is performed $U = 0$. Consequently, we have defined a global upper-bound U on the expected error from the hashed deep neural network as an operator. In other words, we have

$$\mathbb{E}_{\|X\| \leq 1}[\|\tilde{f}(X) - f(X)\|] \leq U \tag{2.10}$$

More details can be found in the [253] paper and in appendix C of this manuscript. Having theoretical results offers further confidence in the method. In what follows, we present a thorough empirical evaluation.

2.1.2 Experimental Results

Hashing Evaluation In Table 2.1, we report our evaluation of the proposed hashing in its ability to reduce the number of distinct weight values while maintaining the accuracy. The baseline approach corresponds to a uniform quantization in 8 bits, which corresponds to 256 distinct values[3]. The first observation is rather straightforward, as RED systematically better preserves the accuracy of the original model on both Cifar10 and ImageNet. Furthermore, in most occurrences, the proposed hashing mechanism also offers a higher reduction of the number of parameters, which means that it better performs in every aspect. however, in instances where the reduction is lower for the hashing, e.g. EfficientNet on ImageNet, the baseline candidate only achieves higher compression by significantly degrading the accuracy of the model. In Table 2.2, we propose a second comparison of the hashing method with respect to another clustering technique: k-means, on ResNet 56 trained for Cifar10. Similarly, we observe that the proposed hashing method outperforms the pre-existing methods. This empirical result was expected by the definition of the hashing method, as well as the theoretical guarantees that we just discussed.

In order to validate the theory, we propose to measure the logits of the deep neural network at hand and its hashed counterpart. The measure of $\mathbb{E}_X[\|\tilde{f} - f\|]$ comes from the train and test sets of each task. In Table 2.3, we report our empirical study of the provided upper-bound U. While we can observe a good tightness (the lower the value the better) in most cases, there are instances where the method

[3]This baseline was considered as it is the go-to method in compression [131].

Table 2.2: Hashing performance on ResNet 56 on Cifar10 as compared to k-means approach. We highlight in bold the best trade-off in compression and accuracy.

method	-	hashing	k-means					
k	-	-	128	196	224	256	384	512
accuracy	93.46	**93.41**	90.97	91.41	93.10	93.28	93.35	93.46
compression	0%	**99.0%**	99.3	98.8	98.6	98.4%	98.1	95.3

Table 2.3: Empirical evaluation of the expected error from hashing. The tightness of this value is verified by comparing it to the empirical measure $\mathbb{E}_X[\|\tilde{f} - f\|]$.

Cifar10											
Architecture	ResNet			W. ResNet		MobileNet v2			EfficientNet		
Model	20	56	164	28-10	40-4	0.35	1	1.4	B0	B4	B7
U	3.1	1.7	3.1	3.5	3.0	3.2	4.0	2.4	4.9	6.7	2.8
Tightness to $\mathbb{E}_X[\|\tilde{f} - f\|]$	7%	54%	13%	64%	79%	22%	65%	28%	57%	65%	17%

ImageNet									
Architecture	ResNet			MobileNet v2			EfficientNet		
Model	50	101	152	0.35	1	1.4	B0	B4	B7
U	0.01	0.01	0.01	0.04	0.08	0.03	0.01	0.01	0.01
Tightness to $\mathbb{E}_X[\|\tilde{f} - f\|]$	6%	1%	1%	37%	27%	28%	11%	3%	9%

struggles. From our intuition, as this problem is mostly specific to Cifar10 (and slightly for MobileNet on ImageNet), this can be attributed to the low amount of data for the upper-bound empirical estimation. All in all, we can conclude that the upper-bound that was proposed does provide a good estimation of the actual average error introduced through the hashing process. Furthermore, a second observation can be made: the upper-bound U has a significantly smaller value than the norm of the logits $\mathbb{E}[\|f\|]$ by a significant margin. This fact and the over-confidence of modern deep neural networks explain the good accuracy preservation of the hashing method. In simpler words, DNNs are over-confident, *i.e.* their logits have a well determined maximum, plus the logits have a larger norm than the introduced error which suggests that on average, said error is unlikely to change the argmax of the logits and as such is unlikely to change the prediction.

Merging Evaluation In Figure 2.3, we propose an evaluation of the influence of the relaxation α parameter. These results show that the larger the models the more extra pruning can be performed without damaging the accuracy. However, as mentioned in the previous chapter, such evaluations should not serve as proof due to the lack of transfer to other datasets such as ImageNet. Fortunately, in this specific case, we actually observe a common trend in every field of deep compression: the more parameters we have for a given accuracy on a specific task, the easier the model is to compress. Aside from this observation, our Cifar evaluation only serves the purpose of comparison with older data-free pruning techniques.

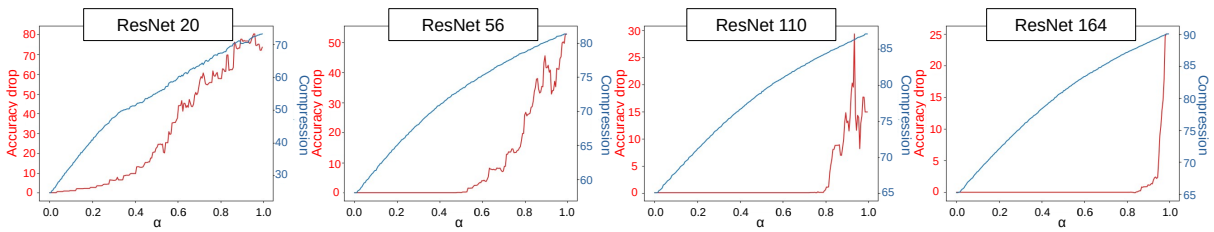


Figure 2.3: Accuracy drop (%, red) and compression (in term of % removed parameters, blue) vs. values of α for ResNet 20, 56, 110 and 164 on CIFAR-10. We can remove high number of similar neurons without impacting the accuracy, particularly for deeper networks.

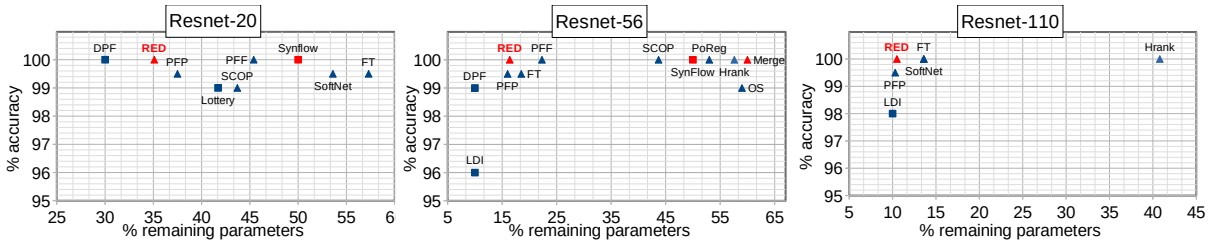


Figure 2.4: Comparison between RED and state-of-the-art methods on Cifar-10, in terms of % accuracy (measured as a percentage of the base model accuracy, the higher, the better) and % remaining parameters (the lower, the better). Each method is classified either as data-free (red) or data-driven (blue) and structured (triangle) or unstructured (rectangle). For each network, RED performs significantly better than other data-free methods, and often as well as data-driven or unstructured methods.

In Figure 2.4, we propose a comparison of the state-of-the-art pruning techniques on Cifar10 from 2020 and earlier[4]. While the proposed RED technique performs data-free structured pruning, we also considered unstructured (squares) as well as data-driven (blue) pruning. Our results simply highlight the ability of RED to find better accuracy *v.s.* compression trade-offs. However, these results should be mitigated on two aspects. First, RED does not generalize well to more challenging tasks. Second, these results were obtained using tensor decomposition on top of the redundancy-based pruning.

In the following section, we will discuss the last part of [252]: tensor decomposition as well as the last aspect of [253]: semi-structured redundancy-based pruning, which will enable good performance on ImageNet.

2.2 Tensor Decomposition and Other forms of Pruning

When this PhD started, the transformer architecture was not as trending as it may be now in computer vision. The community was focused on the ResNet, MobileNet and EfficientNet architectures. The ResNet 50 was the favored backbone for all applications such as object detection or image segmentation. However, EfficientNet and MobileNet sets of deep neural networks offered higher accuracies using significantly fewer parameters. The key to this achievement was two-fold. First, as demonstrated in [229], the use of modern training techniques such as newer optimizers, larger batch-sizes and noisy students. Second, the use of depthwise separable convolution. Consequently, in our first line of research, we tried to decompose the ResNet layers to mimic the depthwise separable convolution layer.

2.2.1 Depthwise Separable Convolution Tensor Decomposition

Let's consider a trained convolutional layer f with hashed weights $W \in \mathbb{R}^{n_o \times n_i \times s_1 \times s_2}$ using convolution kernels of size $s_1 \times s_2$. Our goal is to find a new set of weights W_d and W_p to encode the depthwise and pointwise convolutions respectively, such that $W \approx W_p W_d$ with $W_d \in \mathbb{R}^{n_i \times s_1 \times s_2}$ and $W_p \in \mathbb{R}^{n_o \times n_i}$.

In [252], we propose a new way to view the depthwise separable convolution layer. We see the depthwise layer weights as a spatial basis. Intuitively, a standard convolutional layer mixes the information both spatially and semantically (channels). In the case of the depthwise separable layer, the depthwise layer performs the spatial mixing. Furthermore, if we were to re-construct a convolutional layer from the sequence of depthwise and a pointwise layer then we would get $n_o \times n_i$ 2D spatial kernels, which would simply be the depthwise kernels weighted by the pointwise scalar kernels. In other words, the depthwise kernels constitute a basis to the re-constructed convolutional layer. In turn, the pointwise layer weights can be seen as the representation of the layer in the space defined by the aforementioned basis. Consequently, the proposed process for the decomposition consists in taking the opposite approach. First, we extract a spatial basis W_d of the convolutional weight tensor W. Then, we define z representation W_p for each output channel in that basis. This process is illustrated in Figure 2.5. The first step consists in grouping the weight values W by input channel $i \in \{1, ..., n_i\}$. From the redundancies introduced by hashing, we can expect that there exist a few tensors $B_{j,i}$ such that, for any original weight value $W_{o,i}$ corresponding to the input channel i there exists a linear combination of $B_{j,i}$ that gives $W_{o,i} = \sum_j \lambda_j B_{j,i}$. Then, the weight values of the depthwise layer in the depthwise separable decomposition are directly given by the basis $B_{i,j}$ and the pointwise layer weights are given by the values of λ.

[4]The reasons for this are two-fold. First, RED was conducted in 2020. Second, data-free pruning has seen many iterations since then.

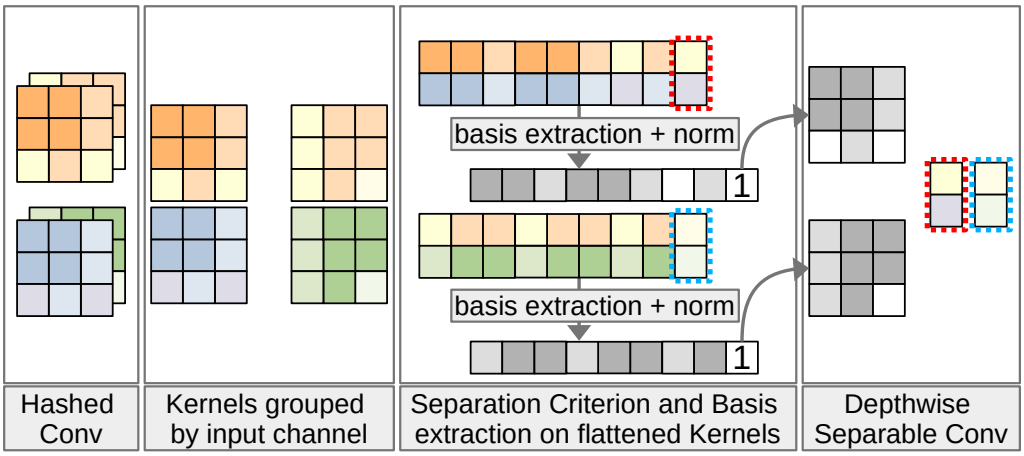


Figure 2.5: illustration of the decomposition process

Note that this decomposition is exact, *i.e.* $W = W_p W_d$. However, while the hashing process does enable a significant compression and introduces many redundancies, it very unlikely that the obtained basis will fit in $W_d \in \mathbb{R}^{n_i \times s_1 \times s_2}$. Formally, for a given input channel i, we will not have a basis of dimension 1 but rather of dimension r_i. For this reason, we call this process an uneven depthwise separable tensor decomposition, as the resulting depthwise layer will have an uneven number of 2D kernels per input channel. Thus, the resulting layer will be of the following size: $s_1 \times s_2 \times (\sum_i r_i)$ for the depthwise part and $n_o \times (\sum_i r_i)$ for the pointwise. This leads to the following criterion: the process is relevant if and only if :

$$\sum_i r_i < \frac{s_1 \times s2 \times n_i \times n_o}{s_1 \times s2 + n_o}. \tag{2.11}$$

In table 2.4, we report the ablation study of the impact of the proposed tensor decomposition on several Cifar10 neural networks. All the models in the table achieve at least 99% of the original model accuracy. Furthermore, we include the previously mentioned pruning technique (merge) for the sake of comparison, in its strict ($\alpha = 0$) and relaxed ($\alpha = \alpha^*$) forms, where α^* is tuned to achieve the highest pruning rate without degrading the accuracy. We can draw several observations:

- From the difference between column 2 and 3, we see that without hashing the depthwise decomposition does not offer any benefits, which is expected as exact redundancies have not been introduced yet. The only pruning performed comes from the relaxed version of the aforementioned redundancy-based structured pruning.
- From column 3 to 4, we see that the hashing on its own introduces enough redundancies for the strict merging to outperform the relaxed merging without hashing.
- From the difference between column 5 and 6 (the two last columns) we can see that the proposed tensor decomposition offers a significant added value as compared to the redundancy-based structured pruning.
- From the difference between the ResNet and Wide ResNet families, we can observe that larger networks with a higher number of output channels require a larger basis for the decomposition, *i.e.* $\sum_i r_i$ is higher on Wide ResNets than on ResNets. This translates to a lower added value in the last column on the Wide ResNets.

In summary, the proposed tensor decomposition does offer a non-negligible added value when focusing on already small networks for toy applications such as Cifar10. However, from our gained experience in the field, we must insist on two major limitations. First, this property fails to enable state-of-the-art compression on real world applications. Second, this is limited to ResNet-like architectures.

Stemming from these observations, all the works that followed during this thesis were designed with the goal of being applicable to any architecture and especially at any scale. In particular, in order to

Table 2.4: Ablation results in terms of % removed parameters compared to the base model.

Hashing	✗	✗	✗	✓	✓	✓
Merge	$\alpha = 0$	$\alpha = \alpha^*$	$\alpha = \alpha^*$	$\alpha = 0$	$\alpha = \alpha^*$	$\alpha = \alpha^*$
Depthwise Separation	✗	✗	✓	✗	✗	✓
ResNet 20	0.00	18.58	18.58	25.18	41.03	**65.05**
ResNet 56	0.00	61.19	61.19	58.45	77.68	**84.52**
ResNet 110	0.00	75.29	75.29	62.41	84.43	**89.64**
ResNet 164	0.00	78.61	78.61	62.73	88.87	**91.06**
Wide ResNet 16-8	0.00	31.08	31.08	19.67	38.67	**51.92**
Wide ResNet 22-2	0.00	51.17	51.17	13.27	63.67	**64.98**
Wide ResNet 28-2	0.00	49.19	49.19	11.46	61.20	**64.19**
Wide ResNet 28-4	0.00	41.79	41.79	20.99	51.99	**56.07**
Wide ResNet 28-8	0.00	33.58	33.58	19.78	41.78	**52.87**
Wide ResNet 28-10	0.00	47.25	47.25	25.59	58.79	**60.80**
Wide ResNet 40-4	0.00	49.67	49.67	43.37	61.80	**70.35**

address both of these limitations in the case of data-free pruning, we introduced split: a semi-structured, redundancy-based pruning technique.

2.2.2 A new Semi-structured Pruning Approach

Let's consider any mat-vec layer f with weights $W \in \mathbb{R}^{n_o \times n_i \times s}$ where s is equal to 1 for a fully-connected layer and equal to $s_1 \times s_2$ for a convolutional layer. Our goal is to replace computations by memory calls. To do so, we propose to perform the merging step per input channels on top of the previously introduced output channel merging. Formally, the original merging required that two output neurons n_i and $n_{i'}$ should be identical. In other words, $W_{i,j} = W_{i',j}$, for all values of j. In the proposed SPLIT method, we relax this constraint such that there exists $j \in \{1, ..., n_i\}$ with $W_{i,j} = W_{i',j}$. Intuitively, this is equivalent to a sparser version of the previously proposed redundancy-based pruning. This relaxing will offer higher pruning rates, as this method is a special case of the original merging method. However, this increased compression rate comes at a price. In order to highlight this price, we need to explain how to actually infer using the proposed method, like we did with structured approach.

With the structured approach, after the identification of the neurons to merge, we had to update the subsequent layers and edit the computational graph which gave us a method that is straightforward to leverage: the model is simply smaller after pruning. This is not true with the SPLIT method. Formally, let $\bar{W}$ be a ragged tensor containing only the unique 2D kernels (or scalar values) from the convolutional (or fully-connected) layer hashed weight tensor W. We can reconstruct the hashed weight values W from the pruning tensor $\bar{W}$ with a tensor of indices I such that for any i, j we have $W_{i,j} = \bar{W}_{I_{i,j}}$. This pruning is very similar to the semi-structured pruning proposed in [203] in that semi-structured pruning requires indices to be stored in order to represent the partial structure of the tensor. The proposed split pruning process is illustrated, for the sake of clarity, in Figure 2.6. In short, the proposed method can be summarized as replacing redundant computations by memory calls. In practice, this is challenging to implement on most hardware devices as the available kernels[5] would rather leverage parallelization and fewer memory calls than selective compute with multiple memory calls. However, this should be mitigated by two elements: first, as previously mentioned, such implementations are indeed efficient on common hardware as demonstrated by Nvidia and second, some devices like FPGAs can be specifically designed for efficient memory calls.

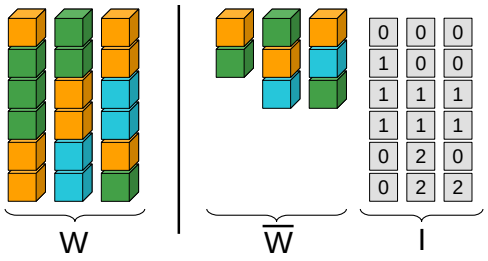


Figure 2.6: illustration of the splitting method. Step 1: identify redundant operations. Step 2: keep only the necessary compute. Step 3: duplicate results at inference.

[5]Here, we are talking about inference kernels which usually are lines of code in c++ with explicit calls to instruction sets specific to the hardware.

Table 2.5: Ablation study of the structured and semi-structured redundancy-based pruning on ConvNets for image classification. We report the percentage of removed parameters.

Hashing	✗	✗	✓	✓	✓	✓	acc drop
merge (α value)	0	α^*	0	α^*	0	α^*	
split	✓	✓	✗	✗	✓	✓	
Cifar10							
ResNet 20	0.00	18.58	25.18	41.03	65.26	**67.48**	0.07
ResNet 56	0.00	61.19	58.45	77.68	85.89	**88.81**	-0.03
ResNet 110	0.00	75.29	62.41	84.43	88.31	**91.82**	0.10
ResNet 164	0.00	78.61	62.73	88.87	90.87	**94.49**	0.01
Wide ResNet 28-10	0.00	47.25	25.59	58.79	77.49	**80.13**	-0.02
Wide ResNet 40-4	0.00	49.67	43.37	61.80	65.97	**68.59**	0.00
MobileNet V2 (0.35)	0.00	4.99	0.00	6.47	53.65	**55.48**	0.00
MobileNet V2 (1)	0.00	4.21	0.00	5.46	69.17	**71.52**	-0.01
MobileNet V2 (1.4)	0.00	2.43	0.00	3.16	77.92	**80.57**	-0.04
EfficientNetB0	0.00	2.02	1.44	2.62	61.79	**64.62**	-0.02
EfficientNetB4	0.00	3.81	1.04	4.93	73.42	**76.34**	0.00
EfficientNetB7	0.00	2.13	0.72	2.76	80.23	**83.42**	0.00
ImageNet							
ResNet 50	0.00	0.09	0.09	0.29	43.95	**44.25**	0.00
ResNet 101	0.00	0.75	0.58	0.75	44.12	**44.51**	0.00
ResNet 152	0.00	0.58	0.58	0.58	43.64	**43.68**	0.00
MobileNet V2 (0.35)	0.00	2.75	0.00	2.75	14.13	**14.88**	0.25
MobileNet V2 (1)	0.00	1.95	0.01	1.95	46.00	**46.97**	0.60
MobileNet V2 (1.4)	0.00	2.16	0.02	2.18	83.73	**85.42**	0.10
EfficientNetB0	0.00	1.68	1.23	1.91	53.52	**54.43**	0.44
EfficientNetB4	0.00	2.43	0.98	2.76	72.26	**74.58**	0.00
EfficientNetB7	0.00	2.23	2.54	2.54	87.50	**89.21**	0.01

Table 2.6: Hashing and split performance on Transformer architectures trained for ImageNet. We report the percentage of removed distinct values through hashing (hashing ratio), the resulting accuracy drop and the percentage of weights removed with split (pruning ratio).

model	$\|W\|$	hashing ratio	acc drop	pruning ratio
DeiT T	5.7M	96.016%	0.000	70.32%
DeiT S	22M	97.994%	0.100	84.93%
DeiT	87M	98.527%	0.000	93.20%
CaiT XS24	26.7M	98.028%	0.000	81.85%
CaiT S24	47M	98.992%	0.030	86.27%
CaiT M36	271M	99.443%	0.100	91.80%
LeViT 128S	7.9M	97.604%	0.000	76.91%
LeViT 256	19M	97.133%	0.000	85.93%
LeViT 384	39M	97.214%	0.000	90.55%

In order to evaluate the proposed method as compared to the structured counterpart, we report our results in Table 2.5. As we consider convolutional neural network, we only prune convolutional layers. We can make several observations:

- From columns 1 and 2, we see that without hashing, no pruning can be performed. Similarly to the tensor decomposition and structured pruning, this result is expected as prior to the hashing step, no strict redundancies have been introduced. On the flip side, some relaxed redundancies can

already be eliminated.

- From columns 3, 4 and 5, our results show that SPLIT vastly outperforms the structured pruning method in its two forms. This result is even more blatant on ImageNet, where the structured approach fails to prune over 3% of the deep neural networks.
- From columns 5 and 6, we can see that the merging relaxation offers little benefit over all and most of the heavy lifting is done by split, especially on ImageNet which was our initial target.
- From the comparison between different architectures, we observe that SPLIT performs well regardless of the model. While it may seem like this is not the case on the smallest MobileNet v2, it should be noted that we did not prune the last fully-connected layer which represents 60% of the parameters and as such, we pruned 15% of the 40% prunable values.

While split does address the limitation of RED in achieving great compression performance on both Cifar10 and ImageNet with ConvNet, in a simple data-free configuration, such architectures are getting less attractive with the rise of Transformers.

In Table 2.6, we report the performance of both the hashing method and the split pruning rates. In short, we observe the stability of the hashing method which achieves high reduction rate: at least 96% of the unique weight values are hashed leading to at least 25 times less distinct values. This hashing protocol induces negligible accuracy drops, which completes our study of the hashing method: it works on all the commonly used architectures for challenging tasks such as ImageNet. Stemming from this good performance, the split method achieves its highest pruning rates yet.

In conclusion, the performance of the proposed split does alleviate the limitations of RED at the expense of more challenging implementation for current hardware. This can also be studied from a theoretical perspective as a birthday problem (see Appendix C.3). However, there is so much that can be achieved in pruning for deep learning without data. In the following section, we will present our contributions regarding importance-based, or magnitude-based pruning, with re-training.

2.3 Importance-Based Pruning

The aforementioned redundancy-based pruning methods assume that the deep neural networks learn redundant operations, which is equivalent to deny the effectiveness of the initialization schemes in their ability to break symmetry. Formally, a well known property of deep neural networks optimized with stochastic gradient descent: if two neurons of a given layer are identical at one point, then they will remain identical through the remainder of the optimization process. This property is called symmetry. In practice, initialization schemes [71, 85] are specifically designed to avoid symmetry. From our previous results on ImageNet, it appears that these methods do work, as the redundancy-based structured pruning fails to achieve significant pruning ratios on such networks. Consequently, we advise focusing on importance-based pruning in order to achieve significant pruning, especially if structured pruning is considered.

2.3.1 Magnitude Criterion

The category of pruning techniques that we are considering are often referred to as either importance-based or magnitude-based. Before delving into the technical details, we would like to explain these two terms. In the case of redundancy-based pruning, we want to remove identical operations, as the redundancies serve no semantic purpose and induce a computational burden. On the other hand, here, our goal consists in the removal of the least necessary operations regardless of similarity to other operations. For example, let's consider a feature extractor for cats and dogs classification. The semantic extracted features are: the size of the animal, the presence of fur on the face and the number of legs. All of these features are distinct (no redundancies) but the last one is clearly irrelevant and should be pruned. Consequently, the advantage of this approach to pruning is that we have fewer constraints on candidate neurons for removal, as we do not need redundancies. The second appellation of magnitude-based comes from the naive approach, which simply consists in measure the magnitude (norm) of the weights as a proxy for importance.

In order to obtain the importance ranking of the neurons, a simple approach would simply consist in considering a labeled validation set $\mathcal{D}$ and evaluate the accuracy of all the possible combinations of pruned neurons. In other words, this method simply consists in testing every possible sub-combination

of neurons and keep the best trade-off in terms of accuracy *v.s.* compression. However, this approach has a major drawback: scalability. In fact, to consider a neural network with N neurons, then we have to evaluate 2^N architectures. With the size N of current neural networks and the trend for ever larger models, such an approach would hold no chance of scaling to larger models. Consequently, every pruning method requires the design of an efficient proxy for the importance ranking of neurons.

In order to tackle this problem, we assume that we have access to a trained neural network F with weight tensors $(W_l)_{l\in\{1,\dots,L\}}$ and a validation set $\mathcal{D}$ comprising a few unlabeled examples (in practice, we use 64 examples).

The baseline approach for importance-based pruning is the magnitude criterion C_{L^p}. This criterion stems on the following intuition: if a weight value is larger than the corresponding output features will, on average, be larger. Thus, they will be more important in the decision-making of the predictive function. Furthermore, learned features are rankable, e.g. in classification, the outputs are sorted by magnitude. Formally, we define the magnitude criterion C_{L^p} as follows,

$$C_{L^p} : (W_n, F, \mathcal{D}) \mapsto \|W_n\|_p, \tag{2.12}$$

where W_n are the weights corresponding to the output neuron n. This criterion has several advantages. First, it does not require any data to be computed, which makes it fast and simple to compute. In practice, when using this criterion, most of the computations are dedicated to fine-tuning the pruned model, in order to recover from the accuracy drop induced by pruning. Second, this pruning criterion does not require the computation of gradients which enables its computation for non-differentiable functions, e.g. integer quantized models. However, despite these advantages, we would not recommend the use of this criterion. Indeed, its limitations are numerous, with the most important one being: this criterion is blind to the previous and following computations. For instance, if the output of a neuron with large weight values is fed to low value input weights in the following layers, then their magnitude would cancel out. In other words, the magnitude criterion offers a measure that is local layer-wise, which limits its effectiveness. In order to address this limitation, we propose to draw inspiration from all the work done in the domain of attribution.

2.3.2 Adapting Attribution techniques to Pruning

In the pursuit of explainability of deep neural network predictive behaviors [239], attribution ranks the inputs of a model by their contribution to the final prediction. For example, in the case of computer vision on images (like image classification on ImageNet), an attribution method would rank the pixels with respect to their contribution to the final logits [186]. This is illustrated in figure 2.7. The most notorious attribution techniques are the GradCam [186] and integrated gradient [205] methods. The attributions methods are often evaluated using two metrics: the insertion and deletion scores. In the deletion score, we remove the least important neurons and measure the impact on the accuracy, the lower the impact the better. Intuitively, we see the similarity between pruning and attribution, as in pruning, we rank weight values while in attribution we rank inputs, both based on an estimation of their relative importance with respect to the outputs. By design, attribution techniques must solve the limitation from the magnitude criterion: the measurement is local with respect to the layer. Formally, as attribution techniques measure the importance of the input with respect to the output, they ought to account for the whole predictive function.

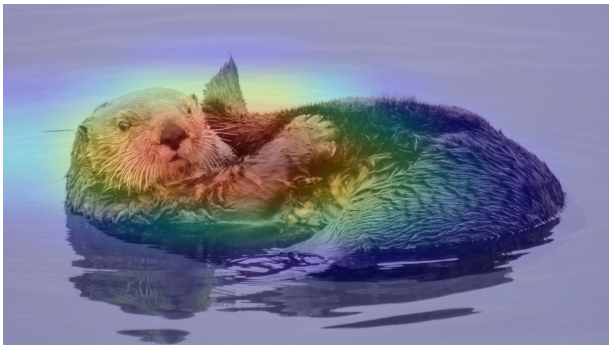


Figure 2.7: illustration of the GradCam method on the image of a beautiful sea otter grooming, through a ResNet 50 model.

The first example of attribution technique that has already been applied to pruning [25] is the gradient criterion C_{∇^p}. Formally, we define this criterion as

$$C_{\nabla^p} : (W_n, F, \mathcal{D}) \mapsto \left\|\nabla_{W_n} F(\mathcal{X} \in \mathcal{D})\right\|_p, \tag{2.13}$$

where $\nabla_{W_n} F(\mathcal{X} \in \mathcal{D})$ is the gradient of the outputs of F with respect to W_n. This criterion measures whether slight changes to the weight values lead to marginal or significant modifications in the final predictions, which addresses the aforementioned limitation of the magnitude criterion on its layer-wise

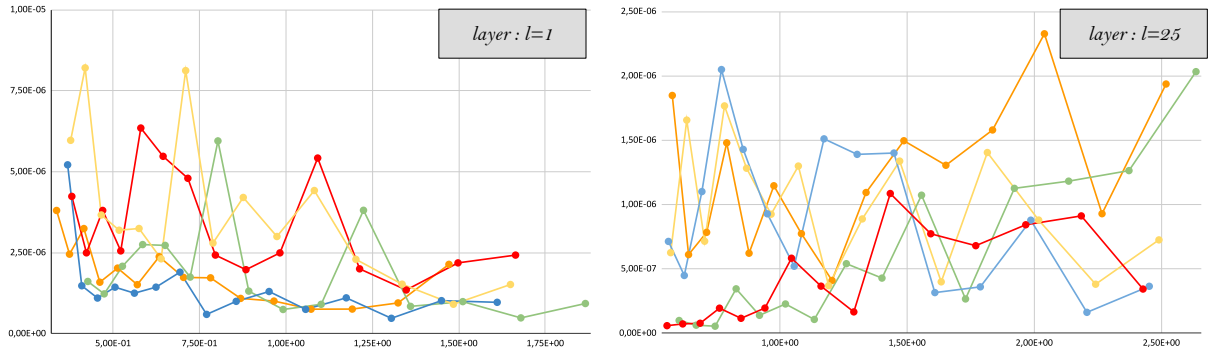


Figure 2.8: Visualization, for 5 random neurons and two different layers of a ResNet 56 trained on Cifar10, of the evolution of $\|\nabla_{\mu^s W_n} F(\mathcal{X} \in \mathcal{D})\|_p$ (y-axis) as the magnitude $\|\mu^s W_n\|_p$ (x-axis) is brought to 0.

locality. However, the gradient criterion C_{∇^p} suffers from another limitation: as the gradient offers a local measurement of slight weight changes in the weight values, this is not sufficient for larger weight values as during pruning they are not slightly changed, they are zeroed-out.

The first solution to this issue consists in mixing the two criterion into a norm $\times$ gradient criterion $C_{L^p \times \nabla^p}$ which combines the best of both worlds:

$$C_{L^p \times \nabla^p} : (W_n, F, \mathcal{D}) \mapsto \|W_n\|_p \times \|\nabla_{W_n} F(\mathcal{X} \in \mathcal{D})\|_p \,. \tag{2.14}$$

While this criterion does address the aforementioned limitation from the gradient criterion in part, the problem still remains to be solved in the sense that we use a heuristic: the gradients and weights magnitude contributions are equal. However, to the best of our knowledge, there are no empirical nor theoretical reasons to do so. What we do know is that $\nabla_{W_n} F(\mathcal{X} \in \mathcal{D})$ only holds within a neighborhood of W_l^n current value, and abruptly setting this neuron weights to zero may very well violate this locality principle.

This problem was properly addressed by the integrated gradient attribution method [205]. Its core idea consists in measuring the importance scores based using a Riemann approximation of the cost function integral from the current value to a baseline. Formally, given an input X and a baseline B, the integrated gradients method estimates the cost of going from X all the way to the baseline value B in terms of the impact on the predictive function. To do so, the method samples values on a segment between X and B and evaluates a simple pruning criterion at each of these values. However, in the case of inputs attribution, the choice of the baseline B at zero is a by-product of the deletion evaluation metric. It consists in removing inputs by setting them to zero. In practice, a missing part of an image could be anything from a default color, a block pixel or a white noise. On the other hand, in pruning, having a zero baseline is relevant as the ultimate goal is to remove computations. Consequently, we adapt the integrated gradient method to pruning and propose the criterion C_{IG^p}, formally defined as

$$C_{\text{IG}^p} : (W_n, F, \mathcal{D}) \mapsto \sum_{s=0}^{S} \|\mu^s W_n\|_p \times \|\nabla_{\mu^s W_n} F(\mathcal{X} \in \mathcal{D})\|_p \tag{2.15}$$

where $\mu \in]0; 1[$ denotes an update rate parameter, that we set to 0.9 on Cifar10 and 0.95 on ImageNet. As a result, the proposed criterion C_{IG^p}, captures the following nuance: some weight values are not sensitive to slight changes, *i.e.* small gradients, however as we start decaying their value, they become more sensitive (*i.e.* their gradient grows). This is illustrated with an empirical example from ResNet 56 in Figure 2.8. We can see that some neurons (right figure, green neuron) start with high gradient values, but end up being not costly to prune. In simpler words, some neurons are costly to slightly decay, but once this is done, going all the way to zero is not semantically expensive. On the other hand, some neurons (right figure, blue neuron) are not expensive initially, but as we go to zero, their cost increases. This highlights the importance of such criterion.

In order to further improve the pruning process, we studied the importance of decoupling the pruning and fine-tuning steps.

2.3.3 Entwining Pruning and Fine-tuning

Let's consider a ranking of the neurons by importance within a given layer, according to a criterion C. For each neuron n we have the importance score $C(W_n, F, \mathcal{D})$. Because the model is pre-trained, we have

Algorithm 1 SInGE Algorithm

Require: neural network F, hyperparameters : O, μ and $(\rho_l)_{l\in\{1,\dots,L\}}$ and dataset $\mathcal{D}$
  for $l \in \{1, \dots, L\}$ **do**
    while pruning_rate$(W_l) \leq \rho_l$ **do** ▷ wait until we reach the goal
      evaluate $M \leftarrow C_{\text{IG}^p}(W_l, F, \mathcal{D})$ ▷ magnitude estimation
      find $n = \arg\min\{M\}$ ▷ find the neuron to prune
      set $W_l^n \leftarrow 0$ ▷ the pruning is performed here
      for $o \in \{1, \dots, O\}$ **do**
        fine-tune the whole network F over a batch from $\mathcal{D}$
      end for
    end while
  end for

to first prune before doing any kind of fine-tuning, which implies that the least important neuron will necessarily be pruned. Let's assume that this neuron is neuron $n = 1$. Then the question at hand is: should we prune the second least important neuron before or after fine-tuning.

If we entwine the fine-tuning and pruning processes, this leads to an iterative method in which we prune the least important neuron among remaining neurons and then fine-tune the pruned network. This method does provide a more relevant importance measurement, as the p^{th} removed neuron is ranked without the $p-1$ least neurons that are known to be less important. However, this approach raises a question: how does this impact the fine-tuning process? There are two approaches to this issue: instead of performing S optimization steps after pruning, we either perform S steps between each pruning iteration or perform S/P steps between each of the P pruning iterations. The first solution introduces a significant overhead in terms of processing time but offers necessarily higher accuracy for the pruned network as we are allowed more optimization steps (SP instead of S) and a more relevant ranking. The second option maintains the processing cost and theoretically offers better trade-offs in terms of compression *v.s.* accuracy as compared to the naive approach. Unfortunately, in practice, this second option actually achieves very similar performance to the first solution. This can be attributed to the fact that the first optimization are by far the most critical ones. Consequently, with SInGE [254] we both designed a new pruning criterion and proposed to divide the fine-tuning steps by the number of pruning iterations and entwine the pruning and fine-tuning phases. This is summarized in the Algorithm 1. The resulting method bears similarities with attribution methods that were published afterward such as the guided integrated gradients (GIG) [106] and IDGI [242], in the sense that it does not compute the scores of all the elements to rank at once, simultaneously. In the following section, we propose to empirically validate the different aspects of the SInGE method.

2.3.4 Empirical Validation

report In Table 2.7, we provide a comparison of each of the described criterion on Cifar10. For each method, we simply remove the least important neurons without fine-tuning. We considered three pruning rates: 75%, 85% and 90%, each representing a different level of difficulty in order to highlight the difference in performance among the proposed methods. The empirical evidence shows the importance of accounting for the whole predictive function, as the first criterion to drop in accuracy is the magnitude criterion. Second, we observe that while the gradient criterion performs decently at a 75% pruning rate, it fails to keep up above 85% which shows the limitation of the gradient that is addressed by the norm $\times$ gradient and integrated gradient criteria. Furthermore, as a side note, it is worth noting that RED was doing about 75% structured pruning using the redundancy-based approach, without tensor decomposition. Overall, these results provide a strong incentive to use our adaptation of the integrated gradients criterion for magnitude-based pruning over the all the aforementioned techniques.

In table 2.8, we evaluate the influence of the fine-tuning process. As expected, the proposed entwined approach systematically outperforms the standard decoupled solution. This observation is even more important under the higher pruning rate regime. In other words, the more difficult the pruning goal, the more important it gets to entwine the pruning and fine-tuning steps. This can be attributed to two factors: first, as previously mentioned, by performing the fine-tuning after each pruning step, the following importance measurements become more relevant. Second, as we fine-tune after pruning only one neuron, the accuracy degradation is limited which leads to a simpler fine-tuning task *i.e.* it is simpler to recover from a marginal change in the predictive function. Consequently, we have empirically validated

Table 2.7: Pruning and accuracy performance of the different pruning criterion on a ResNet 56 trained on Cifar10, without fine-tuning. We also report the standard deviation over multiple runs.

Pruning target (% FLOPS / parameters)	pruning criterion	top-1 accuracy
0.0 / 0.0	baseline	93.46
73.03 / 75.00	magnitude C_{L^1}	42.01 ± 0.41
	magnitude C_{L^2}	42.35 ± 0.38
	gradients C_{∇^2}	77.68 ± 0.52
	magnitude × grad $C_{L^2\times\nabla^2}$	92.36 ± 0.17
	integrated magnitude × grad C_{IG^2}	**93.23** ± 0.23
86.46 / 85.00	magnitude C_{L^1}	19.14 ± 0.82
	magnitude C_{L^2}	19.13 ± 0.09
	gradients C_{∇^2}	28.31 ± 1.75
	magnitude × grad $C_{L^2\times\nabla^2}$	90.28 ± 0.18
	integrated magnitude × grad C_{IG^2}	**92.80** ± 0.30
88.10 / 90.00	magnitude C_{L^1}	10.00 ± 1<
	magnitude C_{L^2}	10.00 ± 1<
	gradients C_{∇^2}	10.00 ± 1<
	magnitude × grad $C_{L^2\times\nabla^2}$	10.00 ± 1<
	integrated magnitude × grad C_{IG^2}	**84.54** ± 0.91

Table 2.8: Comparison between post-pruning and entwined pruning and fine-tuning on a ResNet 56 on Cifar10.

% Pruning target (% FLOPS / parameters)	fine-tuning	# steps	top-1 accuracy
86.46 / 85.00	post-pruning	1000	92.59
	entwined	1000	**93.18**
	post-pruning	2000	92.66
	entwined	2000	**93.25**
	post-pruning	5000	93.13
	entwined	5000	**93.31**
88.10 / 90.00	post-pruning	1000	77.2
	entwined	1000	**85.38**
	post-pruning	2000	80.89
	entwined	2000	**87.52**
	post-pruning	5000	86.39
	entwined	5000	**90.02**

each of the two components that define the SInGE method.

In Table 2.9, we propose to evaluate the proposed method with respect to other structured pruning techniques on the canonical benchmark: ResNet 50 trained for ImageNet. We consider two pruning rate setups: high fidelity in terms of accuracy (at least 75% accuracy) and the high compression rates (at least 50% of the parameters are removed). At first glance, SInGE excels in both setups. In the high accuracy, SInGE manages to achieve both the highest fidelity and the highest compression rates in terms of parameters as well as FLOPs removed.

In Table 2.10, we further evaluate SInGE as compared to other existing structured pruning techniques. We consider a more challenging architecture: MobileNet v2. We considered three setups: 30%, 40% and 50% pruning. It is worth noting that, as we do not prune the last layer, only 64% of the total parameters have a chance to be pruned. Consequently, our results at 50% pruning are quite remarkable with an accuracy of 70.01%, *i.e.* with a 1.79 points accuracy drop. Furthermore, we can see that, contrary to some other pruning techniques, our implementation of SInGE reduces the number of parameters and FLOPs in similar proportions, not favoring one over the other.

Table 2.9: Comparison between existing structured pruning performance on ResNet 50 on ImageNet. In both the low (< 50% parameters removed) and high (> 50%) pruning regimes, SInGE achieves remarkable results.

Method	% params rm	% FLOPS rm	accuracy
baseline	0.00	0.00	76.15
Hrank (CVPR 2020) [133]	36.67	43.77	74.98
RED (NeurIPS 2021) [252]	39.6	42.7	76.1
HAP (WACV 2022) [244]	44.59	33.82	75.12
SRR-GR (CVPR 2021) [227]	-	45	75.76
SOSP (ICLR 2021) [162]	49	45	75.21
SRR-GR (CVPR 2021) [227]	-	55	75.11
SInGE	**50.80** ± 0.02	**57.35** ± 0.11	**76.05** ± 0.07
RED (NeurIPS 2021) [252]	54.7	55.0	71.1
SOSP (ICLR 2021) [162]	54	51	74.4
GDP (ICCV 2021) [80]	-	55	73.6
HAP (WACV 2022) [244]	**65.26**	59.56	74.0
OTO (NeurIPS 2021) [33]	64.1	65.2	73.3
GFP (ICML 2021) [141]	-	65.0	73.94
SInGE	63.78 ± 0.01	**65.96** ± 0.21	**74.7** ± 0.31

Table 2.10: Comparison with existing structured pruning methods on MobileNet V2 backbone for ImageNet.

goal	Method	% params rm	% FLOPS rm	accuracy
-	baseline	0.00	0.00	71.80
30%	CBS (arxiv 2022) [245]	30.00	-	71.48
	Adapt-DCP (TPAMI 2021) [140]	**35.01**	30.67	71.4
	ManiDP-A (CVPR 2021) [211]	-	**37.2**	71.6
	SInGE	30.96	31.54	**71.67** ± 0.06
40%	CBS (arxiv 2022) [245]	40.00	-	69.37
	MDP (CVPR 2020) [78]	**43.15**	-	69.58
	SInGE	40.90	42.30	**70.47** ± 0.09
50%	CBS (arxiv 2022) [245]	50.00	-	62.96
	Adapt-DCP (TPAMI 2021)	-	45.0	64.13
	ManiDP-A (CVPR 2021)	-	48.8	69.62
	Accs (arxiv 2021) [152]	50.00	-	69.76
	GFP (ICML 2021) [141]	-	50.0	69.16
	SInGE	**50.13**	**48.90**	**70.01** ± 0.22

In Table 2.11, we report our last evaluation of SInGE: unstructured pruning. The proposed method can be adapted for scalar importance ranking in a straightforward fashion. The only change that is required is the entwined fine-tuning and pruning. In short, we cannot fine-tune between each scalar pruning due to their number. Consequently, we apply fine-tuning once every 100 scalar pruning values and choose this number in order to have the approximately same total number pruning iterations as in the structured case (and change the value of $\mu = 0.8$). The resulting methods lead to state-of-the-art performance on ResNet 50.

In summary, importance-based pruning offers greater performance as compared to redundancy-based pruning. Such pruning benefits from the use of sophisticated importance proxies that we called pruning criteria from the domain of attribution. Furthermore, the fine-tuning step plays a central role in the final accuracy and overall performance of the method. This step is better entwined between the pruning iterations than performed once post-pruning.

Table 2.11: Comparison with existing unstructured pruning techniques on ResNet 50 on ImageNet.

Method	% params rm	% FLOPS rm	top1 accuracy
DS (NeurIPS 2021) [204]	80.47	72.13	76.15
GMP (arxiv 2019) [67]	80.08	-	76.15
STR (ICML 2020) [117]	79.69	81.17	76.00
RigL (ICML 2020) [59]	80.08	58.92	75.00
SInGE	80.00	82.21	75.12
SInGE	90.00	86.96	73.77

While SInGE offers great performance for a given target global pruning rate, it does suffer from some limitations. In the following, we propose to address one: how to properly assign a pruning rate to each layer individually from a global pruning rate.

2.4 Other Applications Related to Pruning

The core idea introduced in this section is to estimate the relative importance of each layer of a deep neural network with respect to the predictive function outputs. In other words, while the previous methods aimed at ranking neurons within a layer, we now have to rank layers within the network. A naive approach to this problem would consist in simply using the neuron-level importance estimation and reduce it to a scalar in order to get the whole layer importance. However, this approach may not make sense for several pruning techniques, e.g. the redundancy-based pruning. Consequently, we separate these two problems. Furthermore, even when a pruning method can be applied to both the neurons and layers it may only be optimal for one. For example, SInGE is designed for neuron importance estimation and assumes that the object of study is zeroed-out which is never the case for a whole layer during pruning which limits the relevance of the neuron pruning criterion as a layer importance estimator.

This problem of layer ranking was first suggested to us at Datakalab, by members of the French Confiance-AI program, from Valeo. However, the problem was formulated in a significantly different manner: to achieve robust inference.

2.4.1 Robust Inference

In this specific case, robust inference is defined as robust to hardware failures. The prime example of hardware failures to detect is the bit-flip. A bit-flip is an occurrence of a random memory bit being modified (from 0 to 1 or 1 to 0). This kind of robustness is of paramount importance for critical applications [206, 76]. Such hardware failures can even be deliberately provoked as system attacks in order to lead to incorrect predictions and give rise to system failures. Such failures can be catastrophic for some critical systems. For instance, some methods [135] have demonstrated that it was possible to induce a bit flip every 350ms on a data stream of 500Mbit/s. Another example is the DDR2 memory sticks that, according to hardware manufacturers would suffer from an average of 22696 errors occur every year.

The standard approach [137] to tackle the detection of hardware failures and discard the corresponding computations requires running the inference twice and check that the outputs are strictly identical, confirming that no bit flip occurred.

In order to address the challenge of robust inference, we propose to narrow the duplicated operations to the most important layers. This circles back to the initial problem at hand which can also be applied to neural network pruning. Let's define the candidate importance layer-wise criteria. The first set of candidates are zero and first order metrics which were previously[6] introduced alongside SInGE.

Zero and First Order Criteria

Weights (magnitude criterion): the data-free, zero order estimation simply measures the weight norms. In our previous study, we considered the l_1 and l_2 norm. However, from our empirical search,

[6]For the sake of section independence, we re-define all the criteria here.

in the context of layer-wise evaluation, it appears the infinite norm $\|\cdot\|_\infty$ offers a systematically better estimation. The resulting criterion, denoted W, offers the advantage of being fairly simple but does not account for inter-layer relationships.

$$W : (f, \mathcal{X}) \to (\|w_1\|_\infty, ..., \|w_L\|_\infty) \tag{2.16}$$

Gradients: the gradient criterion has already been introduced and is actually a direct adaptation of the GradCam [186] attribution technique which computed the gradients of function F w.r.t. each pixel of the image or feature map. We adapted this criterion to compute the infinite norm of the gradients of F w.r.t. each *weight* instead:

$$\nabla : (f, \mathcal{X}) \to \left(\left\| \mathbb{E}_{\mathcal{X}} \left[\frac{\partial f}{\partial w_1} \right] \right\|_\infty, ..., \left\| \mathbb{E}_{\mathcal{X}} \left[\frac{\partial f}{\partial w_L} \right] \right\|_\infty \right) \tag{2.17}$$

Weight $\times$ gradients: In order to circumvent the intuitive limitations of the previous gradient methods, their combination has been studied [189, 34] and usually leads to a slight improvement in practice for attribution:

$$\mathrm{W} \times \nabla : (f, \mathcal{X}) \to \left(\left\| \mathbb{E}_{\mathcal{X}} \left[w_l \times \frac{\partial f}{\partial w_l} \right] \right\|_\infty \right)_{l \in \{1,...,L\}} \tag{2.18}$$

Higher Order Criterion

GradCam++: based on GradCam, several iterations of the method have been proposed and GradCam++ [30] appears to be the most popular one. It uses third order derivatives and can also be adapted to weight values as follows:

$$\mathrm{GCam{+}{+}} : (f, \mathcal{X}) \to \left(\left\| \mathbb{E}_{\mathcal{X}} \left[\frac{\left(\frac{\partial f}{\partial w_l} \right)^2}{2\left(\frac{\partial f}{\partial w_l} \right)^2 + w_l \left(\frac{\partial f}{\partial w_l} \right)^3} \right] \right\|_\infty \right)_{l \in \{1,...,L\}} \tag{2.19}$$

Integrated Gradients Criteria

Integrated gradients (IG): similarly to neuron importance estimation, the integrated gradient attribution technique can be adapted to layer-wise importance:

$$\mathrm{IG} : (f, \mathcal{X}) \to \left(\left\| \mathbb{E}_{\mathcal{X}} \left[\sum_{\lambda \in [0;1]} \frac{\partial f}{\partial \lambda w_l} \right] \right\|_\infty \right)_{l \in \{1,...,L\}} \tag{2.20}$$

Guided integrated gradients (GIG) similarly to the other famous attribution technique GradCam, the integrated gradients have been widely studied, and several iterations have been proposed. A first example is GIG [106] which consists in decaying, for each integrated gradient iteration, only the least important values, as defined by their gradient magnitudes $||\frac{\partial f}{\partial \lambda w_l}||$.

Important direction guided integrated gradients (IDGI) [242] is the latest improvement over the IG method, as this thesis concludes. Its core idea consists in using the direction of the gradients, weighted by the difference between the outputs at each integrated gradients iteration.

Statistical Criteria

The aforementioned approaches are deterministic with respect to the weight values. However, we know that the loss landscape of deep neural networks can be very chaotic and despite all best efforts, trained neural network never actually converge to zero gradients on challenging tasks. Consequently, statistical criteria evaluate noised weight values rather than the original weights.

SmoothGrad [193] criterion simply evaluate the expected gradients with respect to the weights perturbed by a Gaussian white noise:

$$\mathrm{Smooth}\nabla : (f, \mathcal{X}) \to \left(\left\| \mathbb{E}_{X_{\text{test}}, \mathcal{N}} \left[\frac{\partial f}{\partial w_l + \mathcal{N}} \right] \right\|_\infty \right)_{l \in \{1,...,L\}} \tag{2.21}$$

VarGrad [2] leverages the intuition that the more sensitive the weight values and their corresponding gradient, the higher the variance of said gradients with respect to a small additive noise:

$$\text{Var}\nabla : (f, \mathcal{X}) \rightarrow \left(\left\| \mathbb{V}_{X_{\text{test}}, \mathcal{N}} \left[\frac{\partial f}{\partial w_l + \mathcal{N}} \right] \right\|_\infty \right)_{l \in \{1, \dots, L\}} \tag{2.22}$$

Black Box Criterion

All the aforementioned methods assume that the neural network behaves as a white box, as we can use the chain rule in order to compute the gradients and higher order derivative analytically. On the other hand, black box approaches in attribution assume that no intermediate operations are known. In practice, such methods iteratively measured the sensitivity of the predictive function to partially modified inputs. These methods can be seen as resolving an inverse problem and have in part struggled to keep the interest of the AI community [37].

HSIC [163] was introduced at NeurIPS 2022[7] and is a recent attempt at designing faster black-box attribution methods which narrow the gap in performance with white box methods. The core idea consists in modelling the dependencies between images regions or patches and variations of the predictive function:

$$\text{HSIC} : (f, \mathcal{X}) \rightarrow \left(\mathbb{E}_{\mathcal{X}} \left[\text{hsic}(w_1) \right], \dots, \mathbb{E}_{\mathcal{X}} \left[\text{hsic}(w_L) \right] \right) \tag{2.23}$$

2.4.2 Empirical Robustness Evaluation

We propose to evaluate the capacity of the proposed criteria to rank layers properly. To do so, we built a toy dataset for which we could define a ground truth for the layer rankings.

Ranking Dataset

In order to have a tractable problem in layer ranking, we need our models to achieve a high accuracy with a low number of layers. To do so, we considered MOON dataset for binary classification [168]. For the sake of diversity and to represent as many real world case scenarii, the dataset of models and layer rankings should contain different architecture archetypes:

- The vanilla architecture: a sequential neural network comprising fully-connected layers and ReLU activations in the same fashion as the VGG [190] architecture.
- The skip connection architecture: a refinement of the vanilla architecture, which takes inspiration from the ResNet family [86].
- The stochastic depth architecture: which leverages skip connections [98] and is likely to influence the role of each layers as it forces the first layers to extract refined semantics earlier in the network.
- The transformer architecture: based on the trending transformer block, comprising a multi-head self-attention module followed by a feed forward network.

For each of these possible architectures, the number of layers (or blocks) was randomly sampled, uniformly between 2 and 6. The layer widths uniformly range from 8 to 128 output neurons. We trained each network using the ADAM optimizer with learning rate 0.01 for 6 epochs, which led every network to approximately 100% test accuracy.

In order to obtain the ground truth regarding the layers' importance rankings, we apply different types of noise perturbations to the weights and activations to measure the impact on the accuracy. For each noise distribution, we varied the signal-to-noise ratio in order to evaluate the behavior of the model w.r.t. more or less difficult settings.

- Multiplicative impulse (pepper), denoted $\mathcal{U}$: we randomly prune some weights and activations under a uniform prior. This corresponds to unstructured and structured pruning at the weight and activation levels, respectively.

[7] Paul Novello and Thomas Fel, the authors of HSIC, presented their work at NeurIPS and NeurIPS in Paris, they were great. We met and had a nice conversation about the interactions between explainable AI and compression. It had some perks to participate in the organization of NeurIPS in Paris !

Table 2.12: For each family of architectures (Vanilla, skip, skip+SD and transfo) and distribution prior ($\mathcal{U}$, Gaussian $\mathcal{N}$ and $\mathcal{D}$, applied to weights (W) or activations (act)), we report the proportion of correct rankings of each criterion (with l_∞ as the reduction method). The scores are averaged over 1000 sampled architectures per row.

Archi	Noise	W	∇	W $\times \nabla$	GCam++	IG	GIG	IDGI	Smooth∇	Var∇	Hsic
Vanilla	$\mathcal{U}$ on W	25	76	76	76	59	49	71	71	28	38
Vanilla	$\mathcal{N}$ on W	42	68	68	68	59	56	66	66	36	47
Vanilla	$\mathcal{D}$ on W	32	60	60	33	41	60	48	60	43	60
Vanilla	$\mathcal{U}$ on act	82	82	82	33	67	78	78	53	29	77
Vanilla	$\mathcal{N}$ on act	82	82	82	33	67	78	78	53	29	63
Vanilla	$\mathcal{D}$ on act	82	82	82	33	67	77	77	55	28	49
skip	$\mathcal{U}$ on W	46	47	43	64	44	43	47	52	77	28
skip	$\mathcal{N}$ on W	23	31	32	71	27	33	28	28	39	34
skip	$\mathcal{D}$ on W	30	30	32	72	28	31	27	31	47	16
skip+SD	$\mathcal{U}$ on W	42	42	42	25	41	42	42	17	10	22
skip+SD	$\mathcal{N}$ on W	21	24	24	53	23	23	22	22	32	40
skip+SD	$\mathcal{D}$ on W	20	21	17	47	19	19	20	20	29	12
transfo	$\mathcal{U}$ on W	0	0	0	22	0	0	0	6	12	0
transfo	$\mathcal{N}$ on W	1	1	1	10	0	1	1	18	12	7
transfo	$\mathcal{D}$ on W	0	0	0	7	0	0	0	5	14	0
avg		35	43	43	43	36	39	40	37	31	33

- Additive Gaussian noise $\mathcal{N}(0, \sigma)$, with $\sigma \in]0, max(w \in W_l)]$ applied to either weights or activations. This setting simulates a quantization or hashing process as a small, additive perturbation.
- Additive impulse or Dirac $\mathcal{D}$ noise, where a large perturbation (between 0 and $max(w \in W_l)$) is applied to a proportion between 0 and 100% of uniformly drawn random weights or activations. This simulates a random bit flip.

The ground truth is then derived from the accuracy degradation induced by these perturbations. The whole dataset is available on GitHub[8].

Ranking Evaluation

In Table 2.12, we report the performance of the proposed criterion on our toy dataset. We can make several observations. First, it appears that on average, the GradCam++ and simple first order criteria (gradients and weights $\times$ gradients) offer the highest performance across the board. However, a second observation gives us that the GradCam++ actually performs best on more complex architectures with perturbations on the weight values, while the first order methods particularly shine on the vanilla architecture with noise applied to the activations. However, the GradCam++ results on such architectures, in particular on the transformer architecture, are not satisfactory. This leads us to the conclusion that the first order methods are the most likely to achieve the best performance in general.

ImageNet Evaluation

In order to evaluate the proposed criteria with respect to bit-flip detection, we simulate actual bit-flips and measure the induced accuracy degradation. We made the code for evaluation available on GitHub[9]. The metric is as follows: we monitor the l most important layers according to a criterion, we apply random bit-flips to the others and measure the accuracy. We go from all layers but one, *i.e.* $l = L - 1$, all the way down to $l = 0$. In Figure 2.9, we report the evolution of the accuracy of a ResNet 50 trained on ImageNet. Similarly to what precedes, we observe the good performance of the $W \times \nabla$ and GradCam++ criteria. However, in this case, other criteria such as ∇ under perform. This observation can be explained by the fact that gradients only measure local changes and specifically target the weights. Consequently, they are unlikely to account for perturbations to both weights and activations. Meanwhile, the bit-flips

[8] https://github.com/publicanonymoussubmission/LayersRanking
[9] https://github.com/publicanonymoussubmission/bitswapdetection

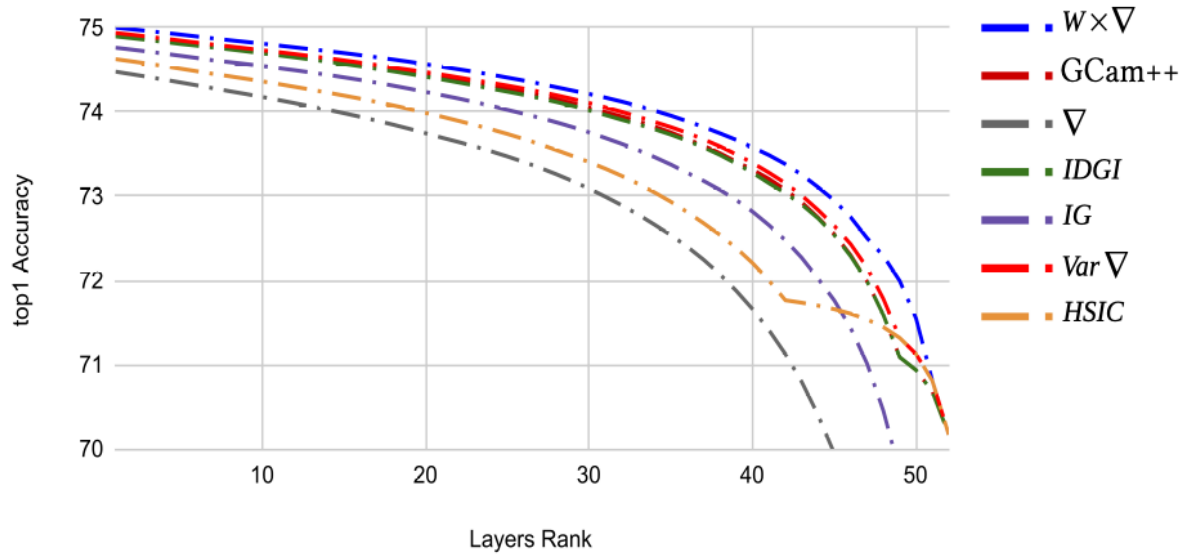


Figure 2.9: Evolution of the accuracy of a ResNet 50 trained on ImageNet as we progressively left important layers unchecked for random bit-swaps. All curves trend downwards as less bit-swap are detected, hence increasing the damage to the predictive function. The higher the accuracy, the better.

are randomly applied to both weights and activations and can induce huge changes that break this locality principle. Nonetheless, our results demonstrate that it is possible to preserve the accuracy without more than 1% accuracy loss at a fraction of the cost of running the model twice. As a result, we only need to monitor 17 layers out of 52 which significantly reduces the redundancy overhead.

Stemming from these results, we deduce that the proposed criteria are indeed well suited for layer ranking, which leads to their evaluation for pruning.

2.4.3 Layer Relative Importance

Given a global target pruning rate γ for trained neural network F, of weights $(W_l)_{l\in\{1,\dots,L\}}$. In other words, our goal consists in removing $\gamma \sum_l \#(W_l)$ parameters where $\#W_l$ denotes the number of scalar weight values in weight tensor W_l. We propose to leverage the proposed criteria C to assign per-layer pruning rate goals γ_l and then use pruning criteria to identify the γ_l proportion of neurons to remove. Formally, we define the γ_l as:

$$\gamma_l = \alpha * \gamma * C(F, \mathcal{D}, W_l) \tag{2.24}$$

with α a normalizing constant such that $\gamma \sum_l \#w_l = \sum_l \gamma_l \#w_l$.

Based on the number of possible combinations of layer-wise and neuron-wise criteria, we performed the evaluation on the TinyML perf challenge [42] which consists in the compression of a pre-trained ResNet-8, on Cifar10. Our global goal consists in removing $\gamma = 20\%$ for structured pruning without fine-tuning. In Table 2.13, we report the accuracies of the pruned models using all the criteria previously introduced for inter-layer relevance (rows) *v.s.* intra-layer importance (columns). Our first observation is the overall wide discrepancy in among the averaged accuracies from the columns (last row) as well as among rows (last column). This suggests that, while some methods are effective at neuron ranking, they may not be suited for layer ranking. Unfortunately, the community has been focusing on intra-layer neuron pruning and may have overlooked the inter-layer relevance and budgetization which plays a significant role. Second, we observe that ∇ and $Var\nabla$ performs the best overall for layer-wise relevance in that context. In particular, for layer-wise importance ranking the ∇ criterion combined with IDGI (closest to the SInGE criterion) for neuron pruning offers the best performance overall, with close results from GradCam++, $Smooth\nabla$ and $Var\nabla$ with integrated gradient based (IG, GIG, IDGI) neuron selection criteria, further confirming our results with SInGE [254]. In practice, we would recommend the ∇ for layer ranking due to its simplicity and lesser processing cost.

Table 2.13: Results (%acc) obtained for TinyML perf challenge ResNet-8 on Cifar-10 with various inter-layer (rows) and intra-layer (columns) relevance criteria. Accuracies are reported for a global pruning rate of $\gamma = 20\%$ of a ResNet-8 trained on Cifar-10 averaged over 10 runs and without post-pruning fine-tuning.

	W	∇	W $\times\nabla$	GCam++	IG	GIG	IDGI	Smooth∇	Var∇	Hsic	avg
W	31	63	18	41	75	75	73	32	21	32	46
∇	31	71	79	67	80	80	82	55	13	74	63
W $\times\nabla$	31	41	47	32	75	75	75	37	37	50	50
GCam++	31	27	82	70	80	80	80	12	52	22	54
IG	31	32	64	23	73	75	75	37	65	39	51
GIG	31	32	66	59	75	73	76	32	47	38	53
IDGI	23	21	55	64	76	73	76	28	50	59	53
Smooth∇	31	17	69	33	50	80	80	62	47	65	53
Var∇	31	77	75	75	80	80	80	63	28	19	61
Hsic	31	36	57	44	73	75	75	11	26	39	47
avg	30	42	61	51	74	77	77	33	43	44	

In summary, inspired from the challenge of robustness to hardware failures, layer-wise importance ranking can lead to significant performance improvements for pruning techniques. However, the existing methods fail to work o, the more challenging transformer architectures, leaving plenty of room for future research.

During this thesis, we had the opportunity to contribute to the domain of deep neural network pruning both redundancy and importance based, providing new pruning criterion, pruning budgetizations and even a new pruning granularity. However, in its current state, pruning still remains less effective than quantization. In the next section, we propose a list of challenges which currently hinder the performance of pruning and are, at least in part, responsible for the limited impact of pruning.

2.5 Future Challenges for Pruning

2.5.1 Hardware Aware Pruning

As previously discussed, the reason why most compression methods are compared with respect to metrics such as the number of parameters or FLOPs removed instead of actual latency and throughput is for the sake of fair comparison, both latency and throughput are sensitive to every aspect of the hardware device: inference engine version, the OS, the GPU, CPU, motherboard, RAM and so on. However, the pruning methods may have converged to local minimum where pruning more parameters does improve the memory footprint but does not necessarily lead to improved latency. In [187], the authors proposed a solution to this, specific to their hardware. The idea consists in maximizing the importance estimated by a pruning criterion with a computational cost constraint from empirical measurements of latency.

At Datakalab, one of our intern engineers has implemented their own approach which showed that removing fewer neurons can actually lead to better performance in terms of latency. However, it appears that this aspect of pruning is still overlooked in the community and a lot of work should be done in order to allow for both relevant evaluation (with respect to actual performance) and still allow for fair comparisons. In our opinion, the release of well grounded, empirically-based, latency estimators could alleviate this limitation. Such evaluators could come in the form of look-up tables, which, for a given architecture, output the corresponding latency on a set of hardware. Such work would require a wide range of hardware in order to offer diversity of end applications and the evaluation of various architectures.

2.5.2 Pruning Granularity

A second key component of pruning which would strongly benefit from a more thorough look into is the pruning granularity. The two most studied pruning granularities are structured and unstructured pruning. The former is the most straightforward to leverage but is too constrained for high pruning rates. The latter is usually very challenging to leverage. To offer other trade-offs in terms of compression and accuracy, researchers have introduced semi-structured formats. However, these granularities almost

systematically fall in-between the structured and unstructured pruning, without actually revisiting the pruning field. In this discussion, we would like to suggest the study of a completely different approach to deep neural network inference pruning, focused on latency. Instead of pruning neurons or weight values, we would prune computations. Intuitively, for a given matrix multiplication algorithm, pruning is an indirect way to omit steps of the original algorithm. In the new paradigm, we would directly prune steps of said algorithm. A more thorough discussion is provided in Appendix C.4.

These are, to me, the most important aspects over which pruning would benefit from further improvements. Still, pruning in its current form is usually less effective than quantization as a compression technique. During this PhD, we work extensively on the latter, and this constitutes the second track of our contributions to the compression community.

Chapter 3

Deep Neural Network Quantization

In chapter 2, we introduced the main dichotomy around the pruning paradigms: redundancy-based *v.s.* importance-based. Regarding quantization, we propose to follow a dichotomy regarding data usage. Following this axis, quantization is often divided in three sub-categories: data-free, gradient-based post-training quantization (GPTQ) and quantization-aware training (QAT).

Data-free quantization was first introduced at ICCV 2019 [157]. Stemming from the ever-growing range of pre-trained neural networks available off the shelf, the authors proposed a whole process which does not require any data to quantize such deep neural networks *i.e.* to map high precision floating point representations to low precision fixed point representations. The core challenge initially consists in figuring a solution for the activation quantization which requires a scaling adapted to their support. However, in a data-free setup, the estimation of their support requires workarounds, which we will thoroughly discuss in the upcoming dedicated section. Such quantization approaches bare numerous advantages. First, their only requirement being a pre-trained neural network, they are not computationally intensive and scale efficiently to any model size. For instance, on the largest LLMs we worked with, almost all data-free quantization techniques took under 10 seconds to operate. Second, they are, by design, privacy compliant. This has been of paramount importance at Datakalab in our discussions with the French watchdog for privacy rights and technologies (CNIL). During this PhD thesis, we mostly contributed to data-free quantization, leading to three publications, each of which address a shortcoming of the first iterations of data-free quantization techniques.

A set of quantization techniques fall in between data-free and GPTQ: data-generative quantization [243, 129]. Such techniques generate synthetic data from a pre-trained neural network. The core idea is fairly simple: assume an input white noise X, then optimize X such that the prediction of the trained neural network over X corresponds to a fictitious label. For instance, for a neural network trained on ImageNet, we would optimize X such that the output gives a specific class, e.g. an otter. The different iterations over this generic principle provided several regularization terms based on input-specific heuristics. The quantization part of these methods is generally limited to already existing data-driven technique. While the whole process does not actually involve real data, we do not classify these methods as data-free, as they involve a very computationally expensive data-generation. However, we do not classify them as GPTQ either as they do not require real data and as such they are privacy compliant by design. During this thesis, we did not work on these methods, but still considered these for the sake of performance comparison.

Gradient-based post-training quantization or GPTQ, was initially proposed as an intermediate quantization configuration between data-free quantization and QAT. However, as the community and literature grew, it became clear that GPTQ is an entire field of quantization. In its most general formulation [156], the idea consists in leveraging a calibration set, which corresponds to a fraction of the training data, in order to optimize the quantized weight values such that they preserve the original predictive function. During this PhD thesis, we also worked quite extensively on this subject and made a few contributions, to which we dedicate an entire section of this chapter.

Quantization-aware training (QAT), was the original quantization approach to deep neural network acceleration. The most significant breakthroughs were two-fold. First, the introduction of the straight-through estimation [22] (STE) addressed a problem that remains to be the center of attention with regard to quantization and optimization: the derivative of the rounding operation. As discussed in the first chapter, the quantization process involves a rounding step, which has a zero derivative. Thus, it requires a workaround in order to actually optimize with stochastic gradient descent. The STE simply consists

in omitting the rounding operation in the backward pass, which is equivalent to redefining the gradient operator of the rounding transformation to be the identity. The second breakthrough was published by Courbariaux *et al.* [45] with the introduction of the first binary neural networks. To this day and to the best of our knowledge, this remains the most impressive feat in the compression community as the quantization to xnor operations leads to a division by 32 of the memory footprint and multiplies by 58 the inference speed [173] without hindering the generalization abilities of the resulting trained neural network. However, such quantization methods are computationally intensive during training. Consequently, it is only late in this PhD that we had the opportunity to work on them. Still, we propose to wrap this chapter on a discussion on the current state of QAT and future work, which, to me, may pave the way towards the deployment of generative AI on edge devices.

3.1 Data-Free Quantization

Following the proposed general outline, we will start with our contributions to data-free quantization. In order to get a full grasp on the work that was done, we will start with a detailed description of the starting point of all modern data-free quantization methods.

3.1.1 Fundamental Work

Let F be a trained neural network comprising L layers $(f_l)_{l\in\{1,\dots,L\}}$ with some of them using weight tensors W_l and bias tensors b_l. In order to quantize any weight tensor W, we require a scaling function s, an optional transformation of the weight tensor t and a rounding step such that the general definition of a quantization operator is

$$Q : W \mapsto \left\lceil \frac{t(W)}{s(W)} \right\rfloor . \tag{3.1}$$

In the early iterations of post-training quantization, both a data-free and GPTQ context, the transformation t is simply the identity operation. On the other hand, the weight scaling function has to satisfy that, for a target $\mathfrak{b}$ bit-width, the quantized weight values cover the full range $[\![-(2^{\mathfrak{b}-1}-1); 2^{\mathfrak{b}-1}-1]\!]$. We have full access to the weight values, contrary to intermediate features. This enables us to define a proper weight scaling function. As a result, the general definition of the scaling function is, to the best of our knowledge, systematically:

$$s : W \mapsto \frac{\max_{w\in W}\{|t(w)|\}}{2^{\mathfrak{b}-1}-1}. \tag{3.2}$$

As a result, we can generalize this quantization process to both weights and activations for a given layer f with weights W and input intermediate feature, the resulting quantized layer f_Q reads

$$f \approx f_Q : X \mapsto Q_x^{-1}(Q_x(X)) \times Q_w^{-1}(Q_w(W)) + b \tag{3.3}$$

where Q^{-1} refers to the de-quantization operation defined as $Q^{-1} : A \mapsto s(A)A$. While the weight quantization process can be explicitly defined from the trained weights only, we face the first challenge of data-free quantization: the input quantization. Formally, we need to figure a way to derive the scaling function for activation quantization. The solution to this challenge was one of the two main contributions of the fundamental work in data-free quantization: DFQ [157].

Scales from Batch-Normalization Layers

Let's consider a sequence of two layers f_1 and f_2 such that the forward pass reads $f_2 \circ f_1$ where f_1 is a batch-normalization layer and f_2 a fully-connected layer. Then, by definition of the batch normalization layer, we have access to statistics on the intermediate features of $f_2 \circ f_1$. Let's note X the input tensor, Y the output $f_1(X)$, then we derive the statistics from the batch-normalization layer f_1 parameters μ, σ, γ and β:

$$\begin{aligned} \mathbb{E}[X] &= \mu \\ \mathbb{V}[X] &= \sigma^2 \\ \mathbb{E}[Y] &= \beta \\ \mathbb{V}[Y] &= \gamma^2 \end{aligned} \tag{3.4}$$

Consequently, once the batch-normalization layer f_1 is folded in the fully-connected layer f_2 to get the fused fully-connected layer f, we can deduce the input range as $[\mu - \lambda\sigma; \mu + \lambda\sigma]$ with $\lambda \in \mathbb{R}_+$. In the

original article, the authors recommended a default value $\lambda = 6$ (keep over 99.99% of the values from the distribution) in order to achieve the best performance in int8 quantization. As a result, the quantized layer reads

$$f_Q : X \mapsto \frac{\max\{\mu - 6\sigma, \mu + 6\sigma\}}{2^{\mathfrak{b}-1} - 1} \text{clip}^{2^{\mathfrak{b}-1}-1}_{-(2^{\mathfrak{b}-1}-1)} \left\lceil \frac{X}{\frac{\max\{\mu-6\sigma,\mu+6\sigma\}}{2^{\mathfrak{b}-1}-1}} \right\rfloor \times \frac{\max_{w \in W} \{|t(w)|\}}{2^{\mathfrak{b}-1} - 1} \left\lceil \frac{W}{\frac{\max_{w \in W}\{|t(w)|\}}{2^{\mathfrak{b}-1}-1}} \right\rfloor + b, \tag{3.5}$$

where $\text{clip}^{2^{\mathfrak{b}-1}-1}_{-(2^{\mathfrak{b}-1}-1)}$ is a clipping function that enforces the $[\![-(2^{\mathfrak{b}-1} - 1); 2^{\mathfrak{b}-1} - 1]\!]$ support. The per-neuron ranges $\mu - \lambda\sigma$ and $\mu + \lambda\sigma$ are reduced (using the maximum) to a single scalar value in order to perform per-tensor quantization. Consequently, we can quantize weights and activations, which leaves the biases represented in high precision. However, the quantization process is lossy and may induce a bias. The second main contribution of the original DFQ paper was a solution to this problem: the bias correction.

Bias Correction

Let's define the bias introduced by quantization as a shift in the expected per-neuron output post quantization. Formally, for a given input distribution $\mathcal{X}$, then the bias shift reads

$$\mathbb{E}[f(\mathcal{X}] - \mathbb{E}[f_Q(\mathcal{X}]. \tag{3.6}$$

Intuitively, a bias shift will lead to a shift in the intermediate features range support which, in turn, will affect the quantization ranges. Formally, if the activations shift from their training ranges, the scaling function will lead to the clipping of the intermediate activations. Consequently, we have to ensure that we do not allow for a bias shift induced by the quantization process. In other words, we need to find an update $\tilde{b}$ to the bias tensor b such that

$$\mathbb{E}[f(\mathcal{X}] - \mathbb{E}[f_Q(\mathcal{X}] - \tilde{b} = 0. \tag{3.7}$$

In order to compute the expectations $\mathbb{E}[f(\mathcal{X}]$ and $\mathbb{E}[f_Q(\mathcal{X}]$, we use the linearity of the expectation with $\mathbb{E}[\mathcal{X}] = \mu$. Then the bias correction can be analytically derived using

$$\tilde{b} = W\mu - Q_w^{-1}(Q_w(W))\mu. \tag{3.8}$$

Thus, applying bias correction consists in setting the bias of the quantized layer as $b + \tilde{b}$. Stemming from this strong data-free quantization baseline, the authors of [157] proposed to perform per-tensor quantization on both weight values and activations. Such a quantization scheme would apply the same scaling factor to all output dimensions of the weight tensor, and as such would be very sensitive to the inter-channels discrepancies. To circumvent this limitation, the authors of [157] propose to perform cross-layer equalization.

Cross-Layer Equalization

The core idea of cross-layer equalization is to leverage the property of the ReLU activation: $\lambda\text{ReLU}(x) = \text{ReLU}(\lambda x)$ for $\lambda \in \mathbb{R}_+$. This property holds for every piece-wise affine activation function. However, in practice, only the ReLU satisfies this property, among commonly used activation functions.

Stemming from this observation, the cross layer equalization method consists in computing a set of per-channel scalings S_l such that, for all layers, we update the current weight tensor $W_l \leftarrow S_l W_l S_{l+1}^{-1}$. As a result, over several iterations of this process, the weight values converge to a less unbalanced set of tensors. However, according to the qualitative analysis provided in DFQ, it appears that, on top of being bounded to ReLU networks, the cross layer equalization never fully converges. This can be attributed to the difficulty of enforcing a cross-neuron balance post-training. Consequently, we proposed to take the opposite approach in SPIQ [256]: instead of trying to enable per-tensor quantization over the whole network, SPIQ unlocks per-channel quantization for both weights and activations.

3.1.2 Quantization Granularity

The definition of the activation scaling $s(X)$ from equation 3.5 induces a dimensionality constraint. We need to apply $s(X)$ to both X, which has n_i channels, and $Q_x(X)Q_x(W)$ which has n_o channels, *i.e.* if $s(X)$ is a vector then the dimension of $s(X)$ is constrained to be either n_{l-1}, n_l or 1. In order to be

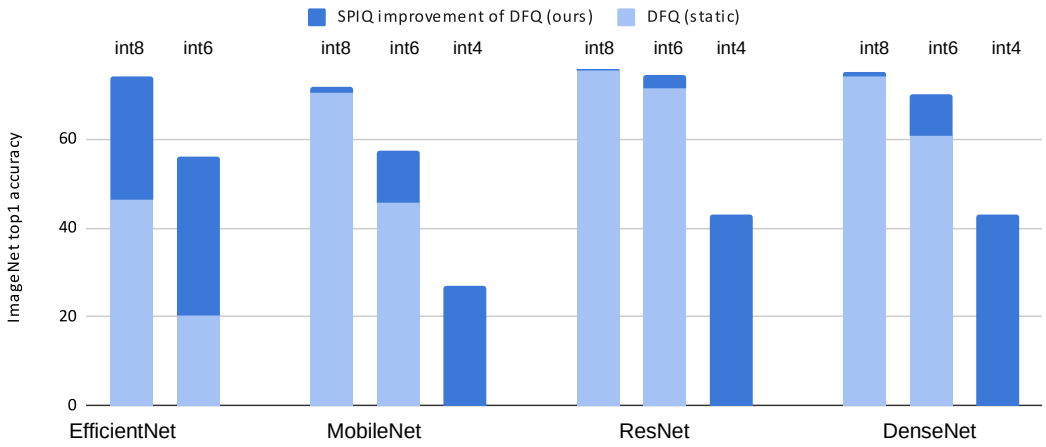


Figure 3.1: Illustration of the accuracy drop attributable to the input and activation quantization. We perform input quantization as defined in [157] as well as SPIQ (ours) but keep the DFQ quantized weight values, *i.e.* per-tensor weight quantization is applied. The results show that input quantization is paramount to the network accuracy preservation, most notably on already compact designs (e.g. MobileNet and EfficientNet). On all tested configurations, SPIQ significantly improves over DFQ [157].

Table 3.1: Comparison of the inference time on the ImageNet validation set for different architectures, quantized with the static (same runtime as SPIQ) and the dynamic methods. We report the boost induced by using the proposed static method.

Method	ResNet	MobNet V2	DenseNet	EffNet B0
dynamic	79s	50s	93s	59s
SPIQ	63s	41s	77s	51s
boost	20.2%	18.0%	17.2%	13.6%

applied to both X and $Q_x(X)Q_x(W)$, it should either be a scalar or of size n_{l-1} and n_l simultaneously. Consequently, $s(X)$ is bounded to be a single, scalar value.

As previously mentioned, the input scaling factor $s(X)$ is derived from the batch normalization layer. Such approach offers a static input scaling factor, *i.e.* the scaling terms are set once and for all. However, this approach leads to significant accuracy degradation, especially when considering low bit-widths as shown in Figure 3.1. In order to alleviate this accuracy degradation, some researchers have introduced dynamic quantization, which is adaptive to the current individual input. The goal is to compute $s(X)^{\text{dynamic}} \in \mathbb{R}$ based on the inferred input X at the cost of overhead computations at inference. Consequently,

$$s(X)^{\text{dynamic}} = \frac{\max_{x \in X}\{|x|\}}{2^{\mathfrak{b}-1} - 1} \tag{3.9}$$

The computation of the maximum is performed at each inference, which adds a significant computational overhead as shown in Table 3.1. However, with respect to the resulting accuracy, the dynamic scaling factor is necessarily tighter to the original distribution, by definition.

Nevertheless, with SPIQ, we propose a solution to achieve both an even tighter fit than the dynamic approach, on average, and preserve the inference properties of static input quantization. We define the scaling vector $s(X)^{\text{spiq}} \in \mathbb{R}^{n_i}$ using the batch normalization layers. Formally, instead of reducing the scaling vector $\frac{\max\{\mu-6\sigma,\mu+6\sigma\}}{2^{\mathfrak{b}-1}-1}$, we simply keep it as is. Thus, we get a scaling vector instead of a scalar one: $s(X)^{\text{spiq}} \in \mathbb{R}^{n_i}$. However, we are no longer able to perform the de-quantization as described in equation 3.5 because of dimensionality constraints. In other words, while the scaling vector $s(X)^{\text{spiq}}$ can be applied to X, it does not match the dimensions of the tensor product output $Q_x(X)Q_w(W)$. To tackle this limitation, we propose to decompose the quantization in two steps. Intuitively, we propose to fold the input de-quantization operation within the weight values, which would not only perform their learned transformation but also de-quantize the input tensors. In practice, this bears similarities with the aforementioned cross-layer equalization process where the scaling factor S is given by the activation ranges and where we keep a per-channel quantization inference. Formally, first, we update the weight tensors W_l such that they apply both the inverse of the rescaling $s(X)^{\text{spiq}}$ to the inputs X and the

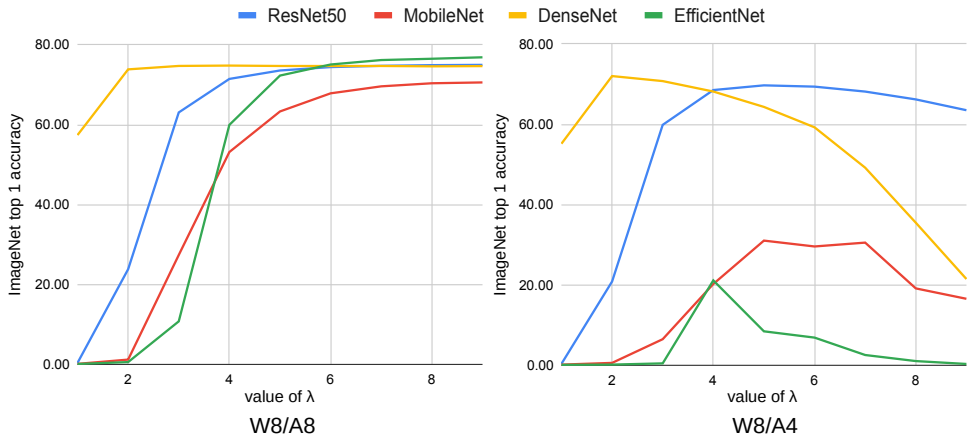


Figure 3.2: Influence of hyperparameter λ on top1 accuracy for weights quantized in int8 using the naive per-channel quantization and inputs quantized either in int8 or int4 for input quantization, on ResNet 50, MobileNet V2, DenseNet 121 and EfficientNet B0 for classification on ImageNet.

operation originally defined by W_l. Then we note,

$$W_l^{\text{spiq}} = \text{diag}(s(X)^{\text{spiq}}) \times W_l \tag{3.10}$$

where diag is the transformation of a vector in a diagonal matrix. Second, we scale the new value W_l^{spiq} as a single weight tensor. Consequently, equation 3.5 becomes:

$$f_l^q : I_l \mapsto s(W_l{}^{\text{spiq}}) \odot \left(\left\lfloor \frac{W_l^{\text{spiq}}}{s(W_l^{\text{spiq}})} \right\rceil \times \left\lfloor \frac{X}{s(X)^{\text{spiq}}} \right\rceil \right) \tag{3.11}$$

The resulting method offers a solution to the full per-channel quantization of both weights and activations for any weight quantization operator. In the following section, we evaluate the proposed SPIQ method as compared to per-tensor static and dynamic quantization alternatives, as well as to other data-free state-of-the-art quantization techniques.

Empirical Validation

In Figure 3.2, we provide an empirical validation of the selection of the λ parameter from the definition of input support derived from statistics in batch-normalization layers. Based on our results on ImageNet, we observe that the selection of the lambda parameter is very sensitive to the target bit-width. In higher precision, such as in 8 bits activation quantization, $\lambda = 9$ is optimal across the studied architecture. In particular, we can observe that the DenseNet architecture faces a decrease in accuracy as λ grows past the value of 8. On the flip side, when working with low precision bit-widths, it appears that lower values of λ enable higher accuracies. As we work in a data-free and low-bit setup, we recommend the use of $\lambda = 4$ for the highest average performance across deep neural networks. Stemming from these guidelines, we evaluate the proposed input quantization schemes.

In Figure 3.3, we evaluate the three approaches: per-tensor static, per-tensor dynamic and per-channel static input quantization, on several architectures trained on ImageNet. We observe that the proposed SPIQ approach offers better trade-offs than both per-tensor approaches. In particular, on the ResNet, DenseNet and EfficientNet families, it appears that SPIQ systematically outperforms the other methods by a significant margin, especially on the more challenging, low bit-widths. On the other hand, on the MobileNet v2 family, it seems that the dynamic quantization achieves a slightly higher accuracy at the expense of inference speed. These results rise to a question: why don't we propose to perform dynamic per-channel quantization to further improve accuracy? The answer is the lack of feasibility. In short, in order to perform dynamic per-channel quantization, we would have to re-quantize the weight tensors at each inference pass which would slow down the inference to the point where the quantized model would be slower than the original model, which would defeat the initial purpose of quantization. Consequently, we can assert that, overall, the proposed SPIQ method offers the best trade-offs in terms of accuracy *v.s.* latency for input quantization.

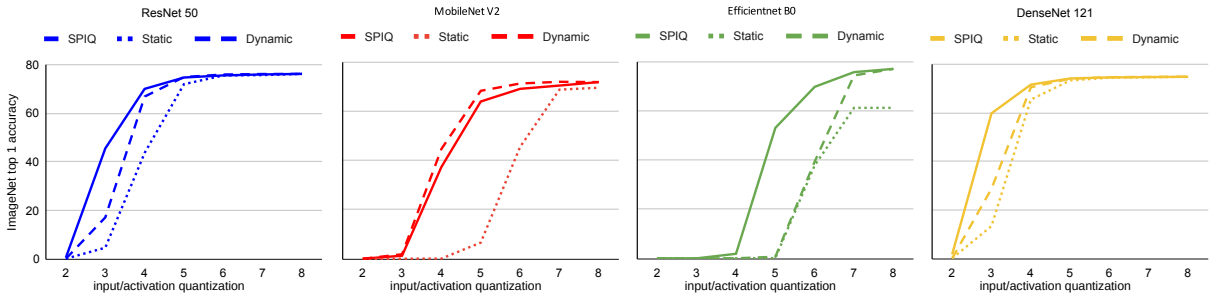


Figure 3.3: Comparison between SPIQ *v.s.* static and dynamic inputs quantization. The weight quantization is fixed to 8 bits, and we vary the input bit range from int2 (ternary quantization) to int8. We report the top1 accuracy on ImageNet for ResNet 50, MobileNet V2, EfficientNet B0 and DenseNet 121.

Comparison to State-of-the-Art Data-free Quantization

In order to achieve state-of-the-art performance in uniform data-free quantization, we propose to combine SQuant [43] with SPIQ. SQuant focuses on weight quantization by minimizing the signed quantization error instead of the absolute quantization error[1]. In Table 3.2, we provide a thorough comparison between data-free and data-generative quantization methods on the canonical benchmark: ResNet 50 trained on ImageNet. We draw several observations:

- in W8/A8, it appears that most data-free quantization technique achieve near full-precision performance. For instance, [43] achieves 76.04 accuracy, which corresponds to a 0.11 accuracy drop. Still, SPIQ enables [43] to fully recover and achieve the exact full-precision accuracy. This suggests that the main degradation was induced by the intermediate features quantization (fig 3.1).
- in W6/A8, all data-free quantization methods start to suffer a more significant accuracy degradation. Meanwhile, SPIQ improves the performance of [43] and limits the accuracy drop to merely 0.01 points.
- in W4/A8, SPIQ increases the accuracy of [43] by 1.10 points, almost reaching 70% accuracy.

It is worth noting that we report 8 bits activation, while in the original articles of SQuant [43] and SPIQ [256] such quantization would have been noted as 4 bits. This is a consequence of the mistake that we highlighted in the first chapter (section 1.4.3 at page 24), where the activation quantization is not properly enforced on all layers. As a result, the experiments were conducted in a kind of mixed-precision setup where some layers are quantized in W4/A4 and some are quantized in W4/A8. In order to further highlight this error (especially as compared to works where we corrected this mistake), we propose to note it as 8 bits activation quantization. Still, our conclusion holds as: SPIQ enables higher accuracy on the ResNet family without having a massive impact on the latency.

In table 3.3, we extend this evaluation to more deep neural network architectures. Our first observation is that SPIQ shows some significant accuracy improvements on MobileNet v2, DenseNet and EfficientNet. This result, on its own, motivates the use of the proposed method. For instance, on MobileNet in W6/A8, the proposed SPIQ achieves 7.86 accuracy improvement. On EfficientNet, in W6/A8, we see an 20.16 increase in accuracy. This phenomenon can be explained by several factors. First, the EfficientNet architecture leverages the squeeze and excite computational block which leads to significant discrepancies across channels, to which SPIQ offers robustness for the quantization process. Regarding the MobileNet architecture, the explanation is empirical: stemming from the observations made in DFQ, it appears that these compact deep neural networks are particularly prone to significant per-channel discrepancies. As a result, the proposed SPIQ methods is of great relevance for the most challenging architectures from a quantization viewpoint. Furthermore, as stated in chapter 1, these results on ImageNet are expected to hold true for other computer vision tasks.

In Table 3.4 and 3.5, we propose to empirically confirm this assertion in the case of SPIQ on both report image segmentation on CityScapes [44] and object detection on VOC [60], respectively. As announced, we observe the same behavior as on ImageNet, with SPIQ significantly improving over both the original SQuant (with per-tensor static quantization) and also over the dynamic input quantization. Furthermore,

[1]Intuitively, if we had a vector $(0.8 \quad 0.8 \quad 0.7)$ its quantized counterpart would be $(1 \quad 1 \quad 1)$ with the signed quantization errors $(+0.2 \quad +0.2 \quad +0.3)$. However, with SQuant, we get $(1 \quad 1 \quad 0)$ and get a total signed error of -0.3 instead of the initial 0.7.

Table 3.2: Comparison between state-of-the-art, data-free, post training quantization techniques with ResNet 50 on ImageNet. We distinguish methods requiring data generation or not (No DG). In SPIQ the weight quantization method is SQuant [43].

	Method	No DG	W-bit	A-bit	Accuracy
ResNet 50	Baseline	-	32	32	76.15
	DFQ [157]	✓	8	8	75.45
	ZeroQ [28]	✗	8	8	75.89
	DSG [264]	✗	8	8	75.87
	GDFQ [240]	✗	8	8	75.71
	SQuant [43]	✓	8	8	76.04
	SPIQ [256]	✓	8	8	**76.15**
	DFQ [157]	✓	6	8	71.36
	ZeroQ [28]	✗	6	8	72.93
	DSG [264]	✗	6	8	74.07
	GDFQ [240]	✗	6	8	74.59
	SQuant [43]	✓	6	8	75.95
	SPIQ [256]	✓	6	8	**76.14**
	DFQ [157]	✓	4	8	0.10
	ZeroQ [28]	✗	4	8	7.75
	DSG [264]	✗	4	8	23.10
	GDFQ [240]	✗	4	8	55.65
	SQuant [43]	✓	4	8	68.60
	SPIQ [256]	✓	4	8	**69.70**

it appears that these two benchmarks are actually less challenging than ImageNet, as we achieve near full-precision accuracy at lower bit-widths. This further highlights the importance of per-channel quantization for data-free methods in order to preserve the information carried over by the intermediate features of deep neural networks.

In Figure 3.4, we conclude the study of the SPIQ method with a qualitative analysis of the resulting intermediate activations. In short, we observe an enhanced information preservation through both SPIQ and dynamic quantization. This is visually blatant with deeper layers (right side of the figure) where the per-tensor static approach contains blank (full black) feature maps.

In summary, the granularity of the activation quantization has a massive impact on the accuracy of the quantized neural networks. In particular, we showed that compact architectures are significantly more sensitive to this aspect of quantization. The proposed SPIQ method offers the performance of per-channel quantization of both weights and activations without hindering the inference speed by folding the activation de-quantization in the quantized weights.

As a result, we designed a novel state-of-the-art data-free, uniform quantization method. It offers a tighter fit to the original distribution. However, these distributions are often non-uniform. Consequently, uniform quantization can only achieve so much. In order to address this limitation, non-uniform quantization proposes different approaches to quantization. During this thesis, we contributed to this side of the field with PowerQuant [255].

3.1.3 Non-Uniform Quantization

Let F be a trained deep neural network. Considering the aforementioned quantization techniques, the quantized counterpart to F would still perform matrix multiplications, simply using more efficient data representations. On the contrary, in prior works on non-uniform quantization [18, 99, 104, 234, 260, 153, 268], the full precision tensors were mapped to a quantized space using non-uniform transformations, which led to a change in the nature of the mathematical operation to perform at inference. A prime example of such quantization scheme is the logarithmic quantization [263] which uses the log function to quantize the weights and converts the quantized weights to encode bit shifts instead of scalar multiplications.

In order to alleviate the need for the support of new instruction sets, we searched for a non-uniform

Table 3.3: Comparison between state-of-the-art data-free, post training quantization techniques with MobileNet V2, DenseNet 121 and EfficientNet B0 on ImageNet. We focused on data-free post training quantization methods that don't involve back-propagation (no BP).

	Method	No BP	W-bit	A-bit	Accuracy
MobileNet V2	Baseline	-	32	32	71.80
	DFQ [157]	✓	8	8	70.92
	SQuant [43]	✓	8	8	71.68
	SPIQ [256]	✓	8	8	**71.79**
	DFQ [157]	✓	6	8	45.84
	SQuant [43]	✓	6	8	55.38
	SPIQ [256]	✓	6	8	**63.24**
	DFQ [157]	✓	4	8	0.10
	SQuant [43]	✓	4	8	0.21
	SPIQ [256]	✓	4	8	**1.28**
DenseNet 121	Baseline	-	32	32	75.00
	DFQ [157]	✓	8	8	74.75
	OCS [267]	✓	8	8	74.10
	SQuant [43]	✓	8	8	74.70
	SPIQ [256]	✓	8	8	**75.00**
	DFQ [157]	✓	6	8	73.47
	OCS [267]	✓	6	8	65.80
	SQuant [43]	✓	6	8	73.62
	SPIQ [256]	✓	6	8	**74.54**
	DFQ [157]	✓	4	8	0.10
	OCS [267]	✓	4	8	0.10
	SQuant [43]	✓	4	8	47.14
	SPIQ [256]	✓	4	8	**51.83**
EfficientNet B0	Baseline	-	32	32	77.10
	DFQ [157]	✓	8	8	46.43
	SQuant [43]	✓	8	8	76.93
	SPIQ [256]	✓	8	8	**77.02**
	DFQ [157]	✓	6	8	20.29
	SQuant [43]	✓	6	8	54.51
	SPIQ [256]	✓	6	8	**74.67**
	DFQ [157]	✓	4	8	0.11
	SQuant [43]	✓	4	8	0.12
	SPIQ [256]	✓	4	8	**0.62**

Table 3.4: Performance (mIoU) on semantic segmentation on CityScapes dataset.

	method	W4/A8	W6/A8	W8/A8	-
DeepLab V3+	baseline	-	-	-	70.71
	DFQ + static	6.51	45.71	70.11	-
	DFQ + dynamic	7.51	66.65	70.22	-
	SQuant + static	7.69	66.77	70.21	-
	SQuant + dynamic	28.87	66.98	70.42	-
	SPIQ	**36.14**	**68.69**	**70.66**	-

quantization method that would map scalar multiplications to scalar multiplications and proposed a novel data-free quantization operator dubbed PowerQuant [255]. Formally, stemming on the previously introduced definition of a quantization operator Q in equation 3.1, we are actually searching for a transformation $t \in \mathcal{T}$ such that,

$$\forall t \in \mathcal{T}, \quad \forall x, y \in \mathbb{R}_+^*, \ (t(x) \times t(y)) = t(x \times y) \tag{3.12}$$

In other words, we are searching for a quantization operator from the set of automorphisms of the group $(\mathbb{R}_+^*, \times)$. Formally, such automorphisms are the power functions $x \mapsto x^a$.

Table 3.5: Performance (mAP) on object detection on Pascal VOC 2012 dataset with SSD MobileNet.

	method	W4/A8	W6/A8	W8/A8	-
SSD MobileNet	baseline	-	-	-	68.56
	DFQ + static	3.94	53.52	67.91	-
	DFQ + dynamic	15.95	62.31	67.52	-
	SQuant + static	14.98	61.29	68.43	-
	SQuant + dynamic	35.47	66.72	**68.56**	-
	SPIQ	**37.88**	**68.01**	**68.56**	-

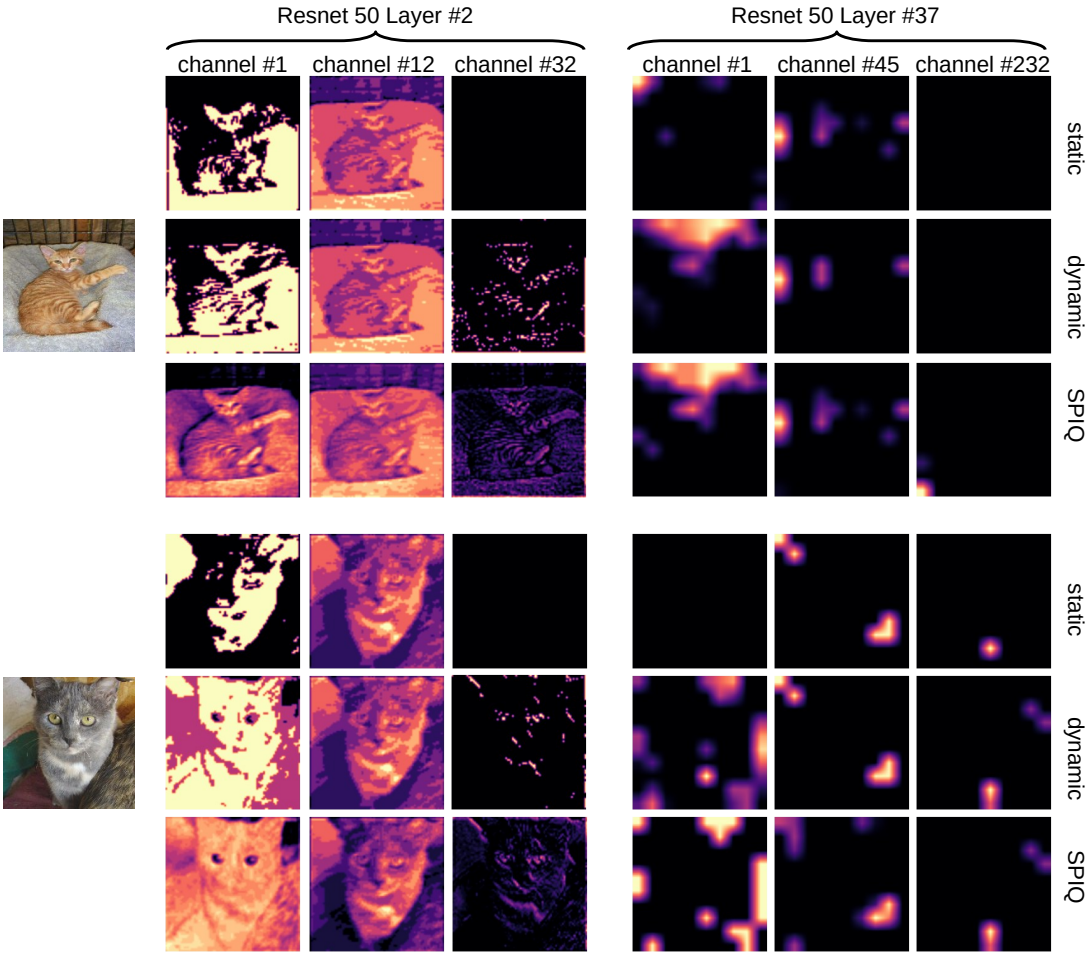


Figure 3.4: Illustration of different feature map channels of a quantized (static, dynamic and SPIQ) ResNet 50.

Lemma 3.1.1. *The set of continuous automorphisms of* $(R_+^*, \times)$ *is defined by the set of power functions* $\mathcal{Q} = \{Q : x \mapsto x^a | a \in \mathbb{R}\}$.

We provide proof in Appendix D.1. Consequently, the set $\mathcal{Q}$ of candidate quantization operators derived from these power functions is formally defined as

$$\mathcal{Q} = \left\{ Q_a : W \mapsto \left\lceil (2^{b-1} - 1) \frac{\text{sign}(W) \times |W|^a}{\max |W|^a} \right\rfloor \middle| a \in \mathbb{R} \right\}. \tag{3.13}$$

Intuitively, we introduce the absolute value for the weight tensors and their sign in order to generalize the power functions to the entire real numbers set. In Figure 3.5, we illustrate the influence of the power exponent parameter a on the quantization process and quantized space:

- For larger values of $a > 1$, the power quantization will assign more precision (lower quantization error) to the tails of the support of the tensor. Intuitively, this comes from the fact that the step

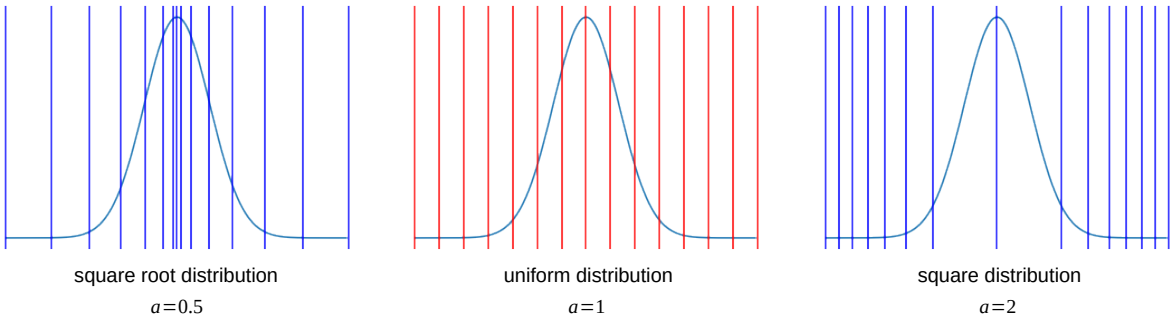


Figure 3.5: Influence of the power parameter a on the quantized distribution for weights distributed following a Gaussian prior. In such a case, the reconstruction error is typically minimized for $a < 1$.

size, in the quantized space, will increase as values get near zero and decrease when values get larger.

- For $a = 1$, the power quantization corresponds to the standard uniform quantization operator.
- For $a < 1$, the power quantization will achieve a higher precision in the center of the tensor distribution and induce larger quantization errors on the tails.

In order to assign the proper power exponent a to the deep neural network we are quantizing in a data-free fashion, we propose to minimize the average quantization error over the weight values. Formally, we define this error ϵ as

$$\epsilon(F, a) = \sum_{l=1}^{L} \left\| W_l - Q_a^{-1}(Q_a(W_l)) \right\|_p . \tag{3.14}$$

where $\| \cdot \|_p$ denotes the L^p vector norm (in practice $p = 2$ and the de-quantization operator Q_a^{-1} is defined as:

$$Q_a^{-1}(W) = \text{sign}(W) \times \left| W \times \frac{\max |W|}{2^{b-1} - 1} \right|^{\frac{1}{a}} \tag{3.15}$$

This optimization problem bares strong mathematical properties which enable us to solve it using the Nelder–Mead method [158]. Such solvers work on problems for which derivatives may not be known or, in our case, are almost-surely zero (due to the rounding operation). In practice, more recent solvers are not required in order to reach the optimal solution.

Lemma 3.1.2. *The minimization problem $\epsilon(F, a)$ with respect to a is locally convex and has a unique global solution.*

We provide proof of the two elements of this lemma in Appendices D.2 and D.3. The question that these properties raise is: is the quantization error correlated enough with respect to the actual accuracy of the quantized model? If the answer is yes, then we would have a well-defined optimization problem. Furthermore, this problem would have theoretical guarantees to achieve higher accuracies than the standard uniform quantization. We introduced the resulting method, which searches for the appropriate power function to define the quantization operator, under the name: PowerQuant [255].

Empirical Validation

In Figure 3.6, we draw the graph of the accuracies and quantization error ϵ with respect to the value of a for ResNet 50 and DenseNet 121 both trained for ImageNet. Our observations are three-fold:

- We observe a strong anti-correlation between the accuracy and quantization error, which indicates that it defines an appropriate optimization proxy. This answers the previous question.
- We empirically confirm the theoretical results, which read: the optimization problem (red curve) is locally convex around the solution and the solution is unique[2].

[2]On a personal note, it is always a pleasure to find a convex problem in deep learning

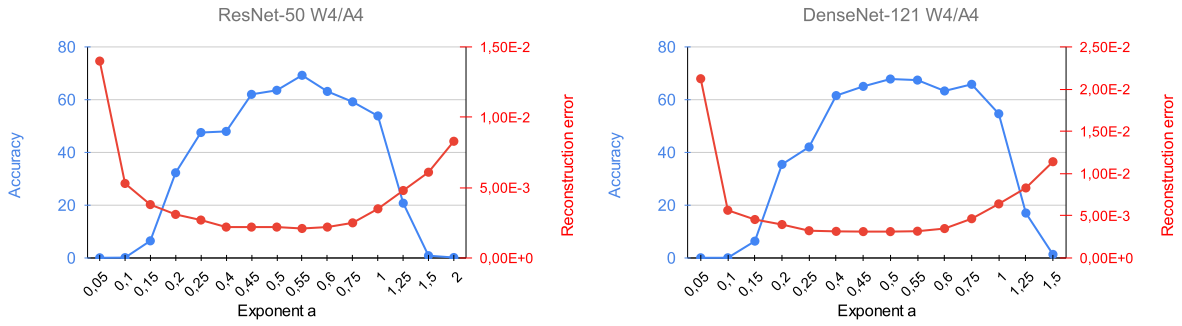


Figure 3.6: Accuracy/reconstruction error relationship for ResNet and DenseNet quantized in W4/A4.

- We also observe that the optimal power exponent a^* has a value near 0.5 which corresponds to the use of square root (Q) and square (Q^{-1}) power functions, which are simpler to implement and optimize for. Consequently, we will use this default value for our comparison to the state-of-the-art later on. Furthermore, this result provides a novel insight on deep neural network compression. In unstructured pruning in particular, we often assume that many low values can be pruned (magnitude-based). However, our results with PowerQuant and the near optimal low value of a suggest that putting more emphasis on low values is of paramount importance in quantization.

In order to complete our study, we propose a comparison between PowerQuant and the other most commonly used non-uniform, fixed point, quantization scheme: the log quantization. In Table 3.6, we report our results on both ResNet 50 and DenseNet 121. It appears that PowerQuant systematically outperforms the latter which is expected as the uniform quantization is an element of the quantization operator search space ($a = 1$). Furthermore, as compared to log quantization, PowerQuant not only achieves higher accuracy, it does so by a massive margin. For example, on DenseNet, PowerQuant adds 62.76 points over log quantization in W4/A8. A second key takeaway is the limitation of the quantization error as an indicator for the quantized model accuracy. Across bit-width, we can see that the lower the bit-width the larger the quantization error (which is intuitive), however, for similar accuracies, we can observe huge discrepancies in terms of errors. For example, on DenseNet quantized in W4/A8, the log and uniform quantization share a similar quantization error with an almost 50 points difference in accuracy. This limitation will be a key element to our extension of PowerQuant, called NUPES which we discuss in the GPTQ section (3.2).

In order to evaluate the proposed PowerQuant method, we conducted several experiments on a wide range of convolutional neural networks and vision transformers. Furthermore, as we will showcase, PowerQuant is effective enough to tackle the compression of all of these networks and even that of large language models.

Comparison to State-of-the-art Quantization Methods

In Table 3.7, we draw comparison between PowerQuant and SOTA data free compression techniques, on the canonical benchmark ResNet 50 trained on ImageNet. Similarly to SPIQ, we see that the PowerQuant method reaches the full-precision accuracy in W8/A8 quantization. We can explain this observation by the fact that the power quantization is applied to both the activation and weight tensors. This enables for a better preservation of the information carried over through the forward pass, like in SPIQ. Regarding low bit-width quantization (W4/A8), PowerQuant further improves the performance over SQuant, also outperforming SPIQ.

As can be empirically observed on the ResNet family of architectures, the activation and weight tensor distributions follow a bell-shaped curve, which is better captured by the PowerQuant method. However, these distributions are not peaky enough for PowerQuant to particularly shine. In order to further highlight the strength of the proposed method, we evaluated it on vision transformers.

In Table 3.8, we provide a thorough comparison of PowerQuant as compared to other data-free quantization techniques and PSAQ [130], which was designed specifically for transformers quantization. It appears that transformers are significantly different to quantize as compared to ConvNets. For instance, DFQ [157] which was the least effective method on ResNets, EfficientNets and DenseNets as compared to SQuant, is outperforming both SQuant and PSAQ. On the other hand, we observe the generalization

Table 3.6: Comparison between logarithmic, uniform and the proposed quantization scheme on ResNet 50 trained for ImageNet classification task. We report both the top1 accuracy and the reconstruction error (equation 3.14) for different quantization configuration (weights noted W and activations noted A).

Architecture	Method	W-bit	A-bit	a^*	Accuracy	Reconstruction Error
ResNet 50	Baseline	32	32	-	76.15	-
	uniform	8	8	1	76.15	1.1×10^{-4}
	logarithmic	8	8	-	76.12	2.0×10^{-4}
	PowerQuant	8	8	0.55	**76.15**	**1.0** $\times 10^{-4}$
	uniform	6	8	1	75.07	8.0×10^{-4}
	logarithmic	6	8	-	75.37	4.6×10^{-4}
	power (ours)	6	8	0.55	**75.95**	**4.3** $\times 10^{-4}$
	uniform	4	8	1	54.68	3.5×10^{-3}
	logarithmic	4	8	-	57.07	2.1×10^{-3}
	PowerQuant	4	8	0.55	**70.53**	**1.9** $\times 10^{-3}$
DenseNet 121	Baseline	32	32	-	75.00	-
	uniform	8	8	1	75.00	2.8×10^{-4}
	logarithmic	8	8	-	74.91	2.5×10^{-4}
	PowerQuant	8	8	0.60	**75.00**	**2.2** $\times 10^{-4}$
	uniform	6	8	1	74.47	1.1×10^{-3}
	logarithmic	6	8	-	72.71	1.0×10^{-3}
	power (ours)	6	8	0.55	**74.84**	**0.7** $\times 10^{-3}$
	uniform	4	8	1	54.83	4.7×10^{-3}
	logarithmic	4	8	-	5.28	4.8×10^{-3}
	PowerQuant	4	8	0.55	**68.04**	**3.1** $\times 10^{-3}$

Table 3.7: Comparison between state-of-the-art post training quantization techniques on ResNet 50 on ImageNet. We distinguish methods relying on data (synthetic or real) or not. In addition to being fully data-free, our approach significantly outperforms existing methods.

Architecture	Method	Data	W-bit	A-bit	Accuracy	gap
ResNet 50	Baseline	-	32	32	76.15	-
	DFQ [157]	No	8	8	75.45	-0.70
	ZeroQ [28]	Synthetic	8	8	75.89	-0.26
	DSG [264]	Synthetic	8	8	75.87	-0.28
	GDFQ [240]	Synthetic	8	8	75.71	-0.44
	SQuant [43]	No	8	8	76.04	-0.11
	PowerQuant [255]	No	8	8	**76.15**	**0.00**
	DFQ [157]	No	4	8	0.10	-76.05
	ZeroQ [28]	Synthetic	4	8	7.75	-68.40
	DSG [264]	Synthetic	4	8	23.10	-53.05
	GDFQ [240]	Synthetic	4	8	55.65	-20.50
	SQuant [43]	No	4	8	68.60	-7.55
	PowerQuant [255]	No	4	8	**70.29**	**-5.62**

capacity of the proposed PowerQuant which achieves remarkable results on all the tested transformer architectures. In particular, on ViT, we can see that not only PowerQuant outperforms other methods, but it does so in a setup involving a much smaller bit-width.

The rise of large language models has brought a new challenge for data-free quantization and a renewed interest for these methods which scale well with the model size. The difficulty to quantize such models comes from the presence of outliers: values that are significantly further from the distribution mean in terms of standard deviations [50]. When performing previously introduced state-of-the-art quantization schemes such as DFQ [157] or SQuant [43], the quantization process simply rounds to zero almost every value because of the scaling factor. Formally, the scaling factor is derived from the tensor support, which is stretched out by the outliers. This is particularly true for weight tensors. However, this stretch leads to all the remaining values behind concentrated around zero and erased by the rounding step. This phenomenon is illustrated in Figure 3.7. We observe that LLMs have outliers that are over 17 standard

Table 3.8: Comparison of data-free quantization methods on ViT and DeiT trained on ImageNet.

model	method	W / A	accuracy
ViT	baseline	-/-	78.05%
	DFQ (ICCV 2019)	8/8	70.33%
	SQuant (ICLR 2022)	8/8	68.85%
	PSAQ (arxiv 2022)	8/8	37.36%
	PowerQuant	8/8	**77.46%**
	DFQ (ICCV 2019)	4/8	66.63%
	SQuant (ICLR 2022)	4/8	64.62%
	PSAQ (arxiv 2022)	4/8	25.34%
	PowerQuant	4/8	**75.24%**

(a) Evaluation for ViT Base

model	method	W / A	accuracy
DeiT T	baseline	-/-	72.21%
	DFQ (ICCV 2019)	8/8	71.32%
	SQuant (ICLR 2022)	8/8	71.11%
	PSAQ (arxiv 2022)	8/8	71.56%
	PowerQuant	8/8	**72.23%**
	DFQ (ICCV 2019)	4/8	67.71%
	SQuant (ICLR 2022)	4/8	67.58%
	PSAQ (arxiv 2022)	4/8	65.57%
	PowerQuant	4/8	**69.77%**

(b) Evaluation for DeiT Tiny

model	method	W / A	accuracy
DeiT S	baseline	-/-	79.85%
	DFQ (ICCV 2019)	8/8	78.76%
	SQuant (ICLR 2022)	8/8	78.94%
	PSAQ (arxiv 2022)	8/8	76.92%
	PowerQuant	8/8	**79.33%**
	DFQ (ICCV 2019)	4/8	76.75%
	SQuant (ICLR 2022)	4/8	76.61%
	PSAQ (arxiv 2022)	4/8	73.23%
	PowerQuant	4/8	**78.16%**

(c) Evaluation for DeiT Small

model	method	W / A	accuracy
DeiT B	baseline	-/-	81.85%
	DFQ (ICCV 2019)	8/8	80.72%
	SQuant (ICLR 2022)	8/8	80.60%
	PSAQ (arxiv 2022)	8/8	79.10%
	PowerQuant	8/8	**81.26%**
	DFQ (ICCV 2019)	4/8	79.41%
	SQuant (ICLR 2022)	4/8	79.21%
	PSAQ (arxiv 2022)	4/8	77.05%
	PowerQuant	4/8	**80.67%**

(d) Evaluation for DeiT Base

Table 3.9: Evaluation on data-free quantization methods on large language models in W4/A16 quantization for common sense reasoning tasks.

model	method	OBQA	ARC-E	ARC-C	WinoGrande	HellaSwag	PIQA	BoolQ	Average
Dolly v2 3B	-	27.600	61.742	34.044	59.274	49.861	73.885	58.315	52.103
	DFQ	26.000	55.808	27.730	57.616	43.567	71.273	53.456	47.921
	SQuant	26.200	55.934	28.328	57.301	43.587	71.491	53.700	48.077
	PQ	**27.200**	**61.880**	**33.253**	**58.950**	**48.560**	**73.905**	**57.609**	**51.622**
Dolly v2 7B	-	30.600	64.141	37.713	61.010	52.778	74.755	64.862	55.123
	DFQ	23.600	53.956	33.020	54.775	44.633	69.260	64.801	49.149
	SQuant	24.200	54.167	33.106	54.854	44.573	69.260	64.801	49.280
	PQ	**30.400**	**62.386**	**35.214**	**60.537**	**52.542**	**74.776**	**65.034**	**54.413**
OPT 13B	-	27.000	61.953	33.020	65.746	52.390	76.714	64.954	54.540
	DFQ	25.400	59.975	29.836	63.062	49.044	75.734	49.786	50.405
	SQuant	25.547	60.145	29.911	63.097	49.032	75.734	49.786	50.465
	PQ	**27.000**	**61.816**	**32.741**	**64.088**	**51.115**	**76.354**	**67.217**	**54.333**
OPT 30B	-	30.600	64.941	34.471	68.272	54.272	77.911	70.061	57.218
	DFQ	27.400	56.860	30.119	63.931	50.119	75.680	67.339	53.064
	SQuant	27.400	57.077	30.231	63.967	50.140	75.676	67.339	53.119
	PQ	**30.500**	**63.552**	**34.276**	**67.930**	**53.625**	**78.183**	**69.966**	**56.862**

deviations away from the mean (more than the number of values that can represented by an int4) while ResNet architectures are bounded by 8 standard deviations. As a result, a quantization method suited for large language models quantization should account for this phenomenon. As our results will showcase, this is the case with PowerQuant.

In order to evaluate quantization models, we measure the zero-shot performance on common sense reasoning datasets: BoolQ [40], PIQA [24], HellaSwag [259], WinoGrande [182], ARC easy and challenge

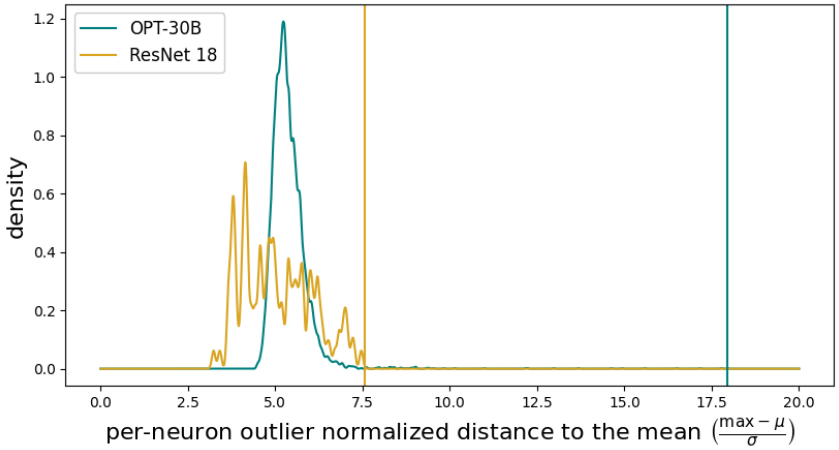


Figure 3.7: Distribution of outlier weight values defined by their distance to the mean value in terms of standard deviations for a ResNet 18 and OPT model. This highlights the specific challenge introduced by LLMs in terms of quantization ranges.

Table 3.10: Evaluation of the proposed method with group-wise quantization on LLMs in W3/A16 and grouping of size 128. We report the average score on common sense reasoning tasks like in Table 3.9.

LLM	DFQ	OPTQ	SmoothQuant	PowerQ + Group-Wise
OPT 13B	35.210	54.01	53.66	**54.472**
OPT 30B	32.184	54.132	53.758	**56.377**

[41] and OpenBookQA [150]. In Table 3.9, we report the performance of data-free quantization techniques for 4 bits quantization of the weight values. These results showcase the ability of PowerQuant to almost perfectly preserve the full-precision accuracy of LLMs in 4 bits, which corresponds to a memory footprint divided by 4 as compared to fp16. In other words, for an OPT 30B model, we go from the requirement of a high grade a100 GPU with 80G of VRAM to load and run the model, down to only needing a consumer grade GPU such as an RTX 3090 with 24G of VRAM to infer the quantized model.

Most recent works [51, 66, 236] that tackle weight quantization of LLMs, all leverage a new quantization granularity, dubbed group-wise quantization, introduced in nuQmm [167]. Its core idea of group-wise quantization consists in using a finer-grained scaling mechanism than per-channel quantization. Formally, instead of using a vector to scale the weight tensor, we use a tiled matrix of scaling factors. If we note W the weight tensor and $W_{i,j:j+128}$ the weight values defining neuron i computations with the j to $j+128$ inputs. Then, per-channel quantization would use a single scaling factor s_i for all values of j, *i.e.* $\frac{W_i}{s_i}$. On the other hand, group-wise quantization uses a scaling $s_{i,j}$ specific to a group (often of size 128), such that we get $\frac{W_{i,j:j+128}}{s_{i,j}}$. However, this approach only works as we only quantize the weight values. Consequently, we hypothesize that may turn in a problem for the quantization community[3] as any work relying on group-wise quantization can never be leveraged for full LLMs quantization. Still, we provide an evaluation, in Table 3.10, of the data-free methods in combination with group-wise quantization for the sake of comparison. All in all, PowerQuant significantly outperforms the more recently introduced quantization techniques in the challenging W3/A16 setup.

In summary, PowerQuant leverages power functions which corresponds to the automorphisms of $(\mathbb{R}_+^*, \times)$. The optimization of the power exponent with respect to the quantization error induced is a locally convex problem with a unique solution which empirically lies near the value of 0.5 enabling for an effective approximation using simpler power functions: the square and square root functions. During this PhD thesis, this method has been the one achieving the most impressive results and never failed to work on any usecase it was tested on.

Nevertheless, while PowerQuant has shown strong results on a wide range of applications and architectures, it could benefit from further improvements as it suffers from several limitations.

[3] more on that in chapter 4.

PowerQuant Limitations

The first limitation of PowerQuant is the optimization of the power exponent a. From our experimentation, while the optimization of a common value for the whole network leads to higher accuracy, it appears that optimizing an exponent per layer only decreases the quantization error without increasing the accuracy (quite to the contrary). The second limitation of the PowerQuant method is that, like all non-uniform quantization methods, it is not straightforwardly compatible with gradient-based post training quantization optimization (as we will discuss in section 3.2.2). This limitation hinders the performance of PowerQuant on smaller networks such as ResNets and ViTs as compared to GPTQ methods.

We address all the aforementioned limitations in our extension work called NUPES and make PowerQuant compatible with the GPTQ framework, while solving the problem of the per-layer optimization of the power exponent in the meantime.

However, before jumping into GPTQ methods, we propose to address a common pitfall shared by all data-free quantization schemes: hardware bit-width support. In short, we have seen that data-free quantization is very effective at 6 and 8 bits weight quantization, yet struggles when considering 4-bits quantization. However, most hardware devices only support 4 and 8 bit formats. Consequently, data-free methods are bound to either degrading the accuracy too much or only use 8 bits quantization. In our work REx [246], we propose a solution to this issue which enables the use of data-free quantization to their fullest while remaining compliant with the hardware bit-widths support.

3.1.4 Hardware Limitations

Let's consider a trained neural network F with L layers with weight tensors $(W_l)_{l\in\{1,\dots,L\}}$. Given a well-supported target bit-width $\mathfrak{b}$, e.g. $\mathfrak{b}=4$, our goal is to design a quantization process that enables high fidelity to the original predictive function. Stemming from previous work on quantization such as DFQ [157], SQuant [43] or our contributions SPIQ [256] and PowerQuant [255], let's assume that we have implemented a quantization operator Q. As previously mentioned (eq 3.14), the quantization error on the weight values is defined by $W - Q^{-1}(Q(W))$. In order to leverage the quantization operator without increasing the bit-width, we proposed to exploit residual expansions (REx) [246] of the quantization errors.

Residual Expansion

Let's note $R^1 = Q^{-1}(Q(W))$ the first order residual. We define the second order term R^2 of the residual expansion from the quantization error such that:

$$R^2 = Q^{-1}(Q(W - R^1)). \tag{3.16}$$

As a result, the residuals are computed recursively from the previous lower order residual terms. However, for inference, we leverage the following property: $R^1X + R^2X \approx WX$, *i.e.* the operations are performed in parallel which enables for efficient inference. Consequently, we can generalize the residual expansion process to any expansion order K, leading to the following:

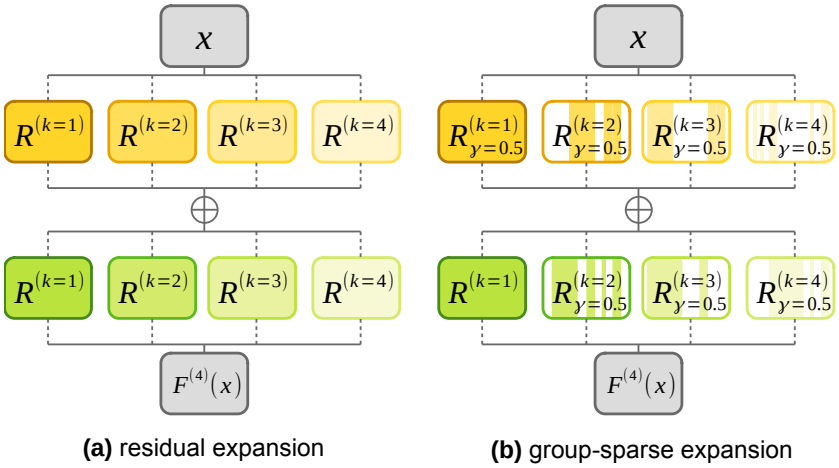
$$R^K = Q^{-1}\left(Q\left(W - \sum_{k=1}^{K-1} R^k\right)\right) \tag{3.17}$$

In turn, we get an expanded layer with K times the weight tensors quantized to $\mathfrak{b}$ bits, as illustrated in Figure 3.8 (a) in the case $K=4$. Intuitively, similarly to other expansions, the proposed REx methods provides an increasingly accurate approximation of the weight tensors such that $\sum_{k=1}^{K} R^k$ converges exponentially fast to W, with respect to K. Formally, we get the following lemma on the expansion convergence:

Lemma 3.1.3. *Let f be a layer with weights $W \in \mathbb{R}^n$ with a symmetric distribution. We denote $R^{(k)}$ the k^{th} quantized weight from the corresponding residual error. Then the error between the rescaled $W^{(K)} = Q^{-1}(R^{(K)})$ and original weights W decreases exponentially, i.e.:*

$$\left| w - \sum_{k=1}^{K} w^{(k)} \right| \leq \left(\frac{1}{2^{b-1}-1}\right)^{K-1} \frac{\left(s_{R^{(K)}}\right)_i}{2} \tag{3.18}$$

where w and $w^{(k)}$ denote the elements of W and $W^{(k)}$ and $\left(s_{R^{(k)}}\right)_i$ denotes the row-wise rescaling factor at order k corresponding to w, as defined in equation 3.1.



(a) residual expansion **(b)** group-sparse expansion

Figure 3.8: Illustration of the proposed method for a two-layers neural network. **(a)** residual expansion at order 4: the intensity of the color map indicates the magnitude of the residual error. **(b)** group-sparse expansion for orders $k \geq 1$ ($\gamma = 50\%$ sparsity).

Stemming on this result, we expect the quantization error to decrease in such a fast pace, which should induce an increase of the fidelity of the expanded quantized neural network to the original predictive function. This property is non-negligible in the data-free setup, where we cannot evaluate the model on a validation set in order to measure the accuracy degradation. Nonetheless, we can derive an upper bound from lemma 3.1.3, on the maximum error $\epsilon_{\max}$ introduced by the residual expansion on the predictions as

$$\epsilon_{\max} \leq U = \prod_{l=1}^{L} \left(\sum_{i=1}^{l} \left(\frac{1}{2^{\mathfrak{b}-1}-1} \right)^{K-1} \frac{s(R^i)}{2} + 1 \right) - 1 \tag{3.19}$$

where $s(R^i)$ is the scaling factor from equation 3.1 applied to each residue (see Appendix E.2 for the detailed derivations from the REx article). These theoretical results suggest that, in practice, a network can be quantized with high fidelity with only a few expansion orders. In terms of hardware support, this implies that for a limited number of bit-widths support, REx offers multiple trade-offs with a protocol that remains agnostic to the quantization operator. Furthermore, this method can be generalized to activation tensors.

Input Expansion

The weight tensor quantization with the aforementioned expansion method leads to the exhibition of new trade-offs in terms of accuracy *v.s.* speed. However, in order to further decrease the memory footprint, we can apply the same process to the activations. Let's consider an intermediate input feature tensor X, then using the generic quantization notation, we get $I^1 = Q^{-1}(Q(X))$ and the generalized to any order K expansion

$$I^{(K)} = Q^{-1}\left(Q\left(X - \sum_{k=1}^{K-1} Q^{-1}(I^{(k)}) \right) \right) \tag{3.20}$$

Now, we get two sets of K expansion orders, one for the weights and one for the activations. As a result, if we were to compute all the possible combinations, the resulting inference would suffer from a massive overhead. Consequently, we only perform the combinations such that $k_1 + k_2 < K$ where k_1 gives the activation and k_2 the weight expansion order respectively, the reminder of the computations being negligible by virtue of the exponential convergence (lemma 3.18).

$$f : X \mapsto \sum_{k_1,k_2 \in \{1,\dots,K\}^2}^{k_1+k_2 \leq K+1} R^{(k_2)} \otimes I^{(k_1)} \tag{3.21}$$

where $\otimes$ is the base operation of the layer, e.g. a convolution for a convolutional layer or a matrix multiplication for a fully-connected layer. However, with formulations from equations (3.17) and (3.20),

the overhead computations induced by the expansion is non-negligible. In the following section, we provide a solution to tackle this issue.

Sparse Expansion

The expansion definition in equation 3.17 does not account for the interneuron discrepancies. To circumvent this limitation, we propose to reduce the overhead cost by only expanding a fraction of the weight values. This sparse expansion can be either structured (*i.e.* it consists of removing output channels or neurons) or unstructured. For the sake of clarity, aside from LLMs, the sparsity will always be structured. The remaining task consists in identifying the most important neurons. However, in a data-free compression setup, we do not have access to activations or gradients. Thus, we measure the relative importance of a neuron in a layer simply by the magnitude, similarly to pruning [154]. The resulting sparse expanded layer is illustrated in Figure 3.8 (b). Formally, given a target budget γ (in %) of allowed overhead computations, we only expand the $\frac{\gamma}{K-1}\%$ most important weight values with the $K-1$ term comes from the fact that we always fully expand the first order.

$$\left(R_{\gamma}^{(k)}\right)_i = (R^{(k)})_i \cdot \mathbb{1}_{\gamma}^{(k)} \tag{3.22}$$

where $\mathbb{1}_{\gamma}^{(k)}$ indicates the indices of the most important neurons.

Similarly to the standard residual expansion, the sparse expansion offers strong theoretical guarantees with respect to the convergence and bounded quantization error (see Appendix E.1). Furthermore, in the sparse expansion, because the residual terms are computed recursively, the method can re-consider neurons that were previously considered unimportant. Consequently, the sparse expansion lowers the upper bound on the maximum quantization error with respect to the overhead computations. In other words, assuming that we properly identify the most important neurons, the sparse expansion will systematically outperform the standard residual expansion. In practice, this has always been the case through our experiments. Formally,

Lemma 3.1.4. *Let f be a layer of real-valued weights W with a symmetric distribution. Then, for $K' < K$ two integers, we have:*

$$Err\left(R^{(1)} + \sum_{k=2}^{K'} R_{\gamma_1}^{(k)}\right) \geq Err\left(R^{(1)} + \sum_{k=2}^{K} R_{\gamma_2}^{(k)}\right) \tag{3.23}$$

where Err is the quantization error (i.e. the absolute difference between the quantized and original weights, as in Equation 3.18) and $K' \times \gamma_1 = K \times \gamma_2 = \beta$.

Proof of this result can be found in Appendix E.3. As a result, this provides an insight on how to efficiently set the parameters K and γ. Theoretically, the lower γ (and proportionally increased K) the better. Fortunately, in practice, the performance converges fast with respect to the sparsification. As a result, we can suggest that in 4 bits quantization, one should use $\gamma \in [25\%; 50\%]$, meaning, we should keep 25% to 50% of each residue with $K = 2$. The method for computing the weights of the expanded model is summarized in Algorithm 2.

As a result, the expanded model can achieve better trade-offs in terms of accuracy *v.s.* number bit operations (BOPS), using only the well-supported bit-widths for the target hardware device. Furthermore, it is worth recalling that all the overhead operations are performed in parallel which further reduce their cost in practice, allowing for higher accuracies than previous methods at equivalent memory and runtime costs, as we will showcase in the upcoming experiments.

Comparison to State-of-the-art

In Table 3.11, we report our results on convolutional neural networks trained on ImageNet. As suggested by our results, the sparse expansion provided by REx enables to find significantly better accuracy *v.s.* compression trade-offs. For instance, on ResNet 50, we achieve a higher accuracy than other data-free quantization using only 4 and 8 bits formats which are two of the three supported bit-widths for modern GPUs (the remainder being 1 bit quantization). Furthermore, it is worth noting that while SPIQ leveraged a sophisticated quantization method, SQuant, in order to achieve state-of-the-art performance. In our results, we showcase REx leveraging the naive quantization and still outperforming SPIQ. As a

Algorithm 2 Expansion Algorithm

Require: trained DNN f with L layers, hyperparameters : K and γ, operator Q
  initialize γ^l and initialize $f^{(K)}$ as a clone of f with K per-layer kernels
  for $l \in \{1, \ldots, L\}$ **do**
    $W \leftarrow$ base kernel of layer l in f
    $W_{\text{acc}} \leftarrow 0$ accumulated quantization error
    for $k \in \{1, \ldots, K\}$ **do**
      $R^{(k)}_{\gamma^l} \leftarrow Q(W - W_{\text{acc}})\mathbb{1}^{(k)}_{\gamma}$ ▷ equation 3.22
      set k^{th} kernel of layer l of $f^{(K)}$ with $R^{(k)}_{\gamma^l}$
      $W_{\text{acc}} \leftarrow W_{\text{acc}} + Q^{-1}(R^{(k)}_{\gamma^l})$
    end for
  end for
  return $f^{(K)}$

Table 3.11: Comparison at equal BOPs with existing methods in W6/A6 and REx with W4/A6 +50% of one 4 bit residue.

DNN	method	year	bits	Accuracy
ResNet 50	full-precision			76.15
	DFQ [157]	ICCV'19	W6/A8	71.36
	ZeroQ [28]	CVPR'20	W6/A8	72.93
	DSG [264]	CVPR'21	W6/A8	74.07
	GDFQ [240]	ECCV'20	W6/A8	74.59
	SQuant [43]	ICLR'22	W6/A8	75.95
	SPIQ [256]	WACV'23	W6/A8	75.98
	REx	-	150% × W4/A8	**76.01**
MobNet v2	full-precision			71.80
	DFQ [157]	ICCV'19	W6/A8	45.84
	SQuant [43]	ICLR'22	W6/A8	61.87
	SPIQ [256]	WACV'23	W6/A8	63.24
	REx	-	150% × W4/A8	**64.20**
EffNet B0	full-precision			77.10
	DFQ [157]	ICCV'19	W6/A8	43.08
	SPIQ [256]	ICLR'22	W6/A8	54.51
	REx	-	150% × W4/A8	**57.63**

Table 3.12: Bert [52] quantized in W4/A8, on the GLUE tasks. We provide the original performance from the article (original) of BERT on GLUE as well as our reproduced results (reproduced). REx is applied to the weights with 3 bits + 33% sparse expansion.

task	original	reproduced
CoLA	49.23	47.90
SST-2	91.97	92.32
MRPC	89.47/85.29	89.32/85.41
STS-B	83.95/83.70	84.01/83.87
QQP	88.40/84.31	90.77/84.65
MNLI	80.61/81.08	80.54/80.71
QNLI	87.46	91.47
RTE	61.73	61.82
WNLI	45.07	43.76

uniform [113]	log [153]	SQuant [43]	SPIQ [256]	REx
45.60	45.67	46.88	46.23	**47.02**
91.81	91.53	91.09	91.01	**91.88**
88.24/84.49	86.54/82.69	**88.78**/**85.24**	88.78/85.06	88.71/85.12
83.89/83.85	84.01/83.81	83.80/83.65	83.49/83.47	**83.92**/**83.85**
89.56/83.65	90.30/84.04	90.34/84.32	90.30/84.21	**90.50**/**84.35**
78.96/79.13	78.96/79.71	78.35/79.56	78.52/79.86	**79.03**/**79.96**
89.36	89.52	**90.08**	89.64	**90.08**
60.96	60.46	60.21	60.21	**61.20**
39.06	42.19	42.56	42.12	**42.63**

result, we observe that, on MobileNet v2, REx improves over the DFQ starting point by 18.36 points. Consequently, we can assert that REx works properly on convolutional neural networks. Similarly to PowerQuant, we value the ability of REx to enable significant performance improvements on a wide variety of architectures and tasks. To show this ability, we evaluated REx on transformer architectures.

In Table 3.12, we provide a comparison of the sparse expansion method using 3 bits plus a 33%

Table 3.13: Average GLUE performance of data-free quantization methods.

Uniform	SQuant	SPIQ	REx
74.16 ± 0.08	74.68 ± 0.19	74.48 ± 0.35	**75.00** ± 0.16

Table 3.14: Evaluation on Common sense reasoning benchmarks for OPT-13B [262] LLM quantized in W4/A16. For each quantization operator DFQ [157], SQuant [43] and PowerQuant [255], we share performance with and without REx (noted with check marks). We also provide the original full-precision (FP) performance.

	FP	DFQ [157]		SQuant [43]		PowerQuant [255]	
Use REx	-	✗	✓	✗	✓	✗	✓
HellaSwag	52.43	49.25	**50.14**	49.23	**50.21**	**51.29**	50.98
OpenBookQA	27.20	**25.80**	25.40	25.40	**26.20**	25.80	**27.80**
ARC-E	61.91	59.93	**61.91**	59.97	**61.95**	**60.82**	60.52
ARC-C	32.94	30.2	**32.42**	30.12	**32.34**	31.57	**32.94**
Winogrande	65.04	64.56	**64.72**	64.48	**64.88**	64.88	**65.04**
PiQA	76.88	75.84	**76.17**	75.84	**76.30**	75.90	**76.93**
BoolQ	65.90	54.71	**65.54**	54.28	**65.38**	**70.43**	69.45
Average Score	54.61	51.47	**53.76**	51.33	**53.91**	54.38	**54.81**

Table 3.15: We report the different trade-offs achieved with REx expanding over different proposed quantization operators in W4/A8 as compared to their performance in W8/A8, on a MobileNet V2.

method	W4/A8	W4$_{+\ 25\%}$/A8	W4$_{+\ 50\%}$/A8	W4$_{+\ 75\%}$/A8	W6/A8	W8/A8
naive [113]	0.1	53.11	64.20	**71.61**	51.47	70.92
SQuant [43]	4.23	58.64	67.43	**71.74**	60.19	71.68
SPIQ [256]	5.81	59.37	68.82	**71.79**	63.24	**71.79**
BrecQ [128]	50.130	61.30	69.80	**71.77**	68.71	**71.75**
AdaRound [156]	65.31	70.94	71.28	**71.76**	70.45	**71.76**

sparse expansion in order to provide a fair comparison, *i.e.* at equal Binary Operations (BOPS), with other methods quantized in 4 bits. At first glance, it appears that REx significantly improves over previous techniques. However, in order to evaluate the BERT architecture [52], we consider the glue set of tasks [225] which induces variance in the results. In Table 3.13, we provide the average scores with their corresponding standard deviation. These results suggest that our previous observation bared significance. This motivates the assertion that REx generalizes to multiple architectures and tasks. However, in order to fully leverage the strengths of REx on large language models, we proposed to adapt the method to specifically target one of the core challenges of LLMs quantization, namely the presence of outlying weight values.

In Table 3.14, we provide the evaluation of REx in combination with other data-free quantization approaches on LLMs evaluated with common sense reasoning tasks. As showcased, REx systematically improves the performance of these methods by a significant margin, to the point where PowerQuant + REx even outperform the original model. To achieve these results with a negligible overhead, we use REx to only quantize outliers in a sparse (unstructured pruning) manner with a binary encoding. Formally, we identify the outliers as any weight scalar value that is at least 6 standard deviations away from the average within a weight tensor. We quantize these outliers in a binary sparse tensor. We recall that binary values are $\{-1, 1\}$ enabling support for both positive and negative outliers. The resulting expansion is not only binary but over 99.8% sparse, which leads to a marginal overhead that translates in less than 0.26% latency overhead. All in all, these results highlight the ability of REx to help improve the performance of quantization methods on LLMs. Furthermore, they provide first results in the ability of REx to work with other quantization operators.

In Table 3.15, we provide thorough evaluation of the proposed REx method with other quantization operators, including data-driven ones. We can observe that regardless of the quantization operator, the REx sparse expansion enables to achieve significantly higher accuracy in W4/A8 with up to 50% int4 added weight values than in direct 6 bits quantization. Similarly, using only 75% extra 4 bits operations, we can outperform the 8 bit quantization which concludes our demonstration of the generalized

performance of REx.

In summary, the proposed sparse expansion is designed to enable any quantization method to achieve better trade-offs in terms of accuracy *v.s.* compression rate on any deep neural network architecture trained for any task.

On this note, we end our study of data-free quantization. In order to achieve higher accuracies at lower bit-widths, we worked on GPTQ, at the expense of the privacy by design and extreme scalability of data-free quantization. During this PhD thesis, we were able to achieve several improvements over existing GPTQ methods, which we shall highlight in the following section.

3.2 Gradient-Based Post-Training Quantization

Following the general outline, GPTQ methods offer an intermediate setup between data-free quantization and quantization-aware training, both in terms of compression *v.s.* accuracy trade-offs and in terms of processing cost. Similarly to the previous section, we will start by a description of AdaRound [156], the fundamental work in GPTQ.

3.2.1 Rounding Up or Down?

Let's consider a trained neural network F comprising a sequence of layers $(f_l)_{l\in[\![1;L]\!]}$ such that for any l all the input layers to f_l are among the $f_{l'}$ with $l' < l$, *i.e.* the layers are ordered. Each layer is assumed to already be quantized with their corresponding input scales set. The goal of GPTQ methods in general is to leverage a small, unlabeled calibration set which is empirically set to 1024 data points.

To do so, AdaRound [156] performs the optimization layer per layer in a sequence. Intuitively, the goal is to set differentiable objectives with unlabeled data. This is achieved by exploiting the fact that the full-precision model provides a ground truth for each individual layer. In other words, for any layer f, we optimize the quantized layer f_Q such that its output features match the output features of f. In practice, we extract the full-precision input features X and their counterparts X_Q from the quantized model, using the calibration set $\mathcal{D}$ using the previous layers in F. Then we extract a ground truth Y from f and optimize the weight values using stochastic gradient descent with batches of size 32 and the Adam optimizer (with default hyperparameters). The optimization goal $\mathcal{L}$ is then a simple Euclidean distance on the features:

$$\mathcal{L} = \|f_Q(X_Q) - f(X)\| = \|f_Q(X_Q) - Y\| \,. \tag{3.24}$$

In order to focus the optimization on relevant values, [156] includes the activation functions in the layers to optimize. Formally, if the mat-vec operation in f is followed by an activation act, then the layer reads $f : X \mapsto \text{act}(WX+b)$ and the quantized counterpart gives $f_Q : X_Q \mapsto \text{act}(Q^{-1}(Q(W))Q^{-1}(Q(X_Q))+b)$. Intuitively, for a ReLU deep neural network, some features will be zeroed out by the ReLU anyway, thus we should not waste any effort (and gradients) trying to match such features and rather focus on the information that will be carried over.

By design, this process is supposed to be sequential. Intuitively, we want the input features X over which we perform the optimization to represent data that would actually be processed at inference. In other words, we do not want to optimize for examples that are sure not to occur in practice, similarly that we did not want to preserve information that would not be carried by over anyway. For this reason, we need to first optimize the layers from 1 to $l-1$ before tackling the l^{th}. Similarly, we want to quantize the input features during the optimization in order to get the following optimization goal:

$$\mathcal{L} = \left\|\text{act}(Q^{-1}(Q(W))Q^{-1}(Q(X_Q)) + b) - f(X)\right\| . \tag{3.25}$$

While all the aforementioned constraints make the optimization more challenging, especially as the process goes on and reaches deeper layers, the calibration set is so small that overfitting is bound to happen. In order to alleviate this shortcoming, AdaRound does not fully optimize the weight tensor, but rather learns whether to round up or down the weight values. Formally, let's consider a layer f with weight tensor W, then we infer the quantized counterpart as

$$f_Q : X_Q \mapsto \text{act}\left(s(W)s(X) \times t^{-1}\left(\left\lceil \frac{t(W)}{s(W)} + \epsilon \right\rfloor\right) \times t^{-1}\left(\left\lceil \frac{t(X_Q)}{s(X_Q)} \right\rfloor\right) + b\right) \tag{3.26}$$

Table 3.16: Evaluation, in W4/A8, of PowerQuant (PQ) with AdaRound (AdaR) in a naïve combination.

	ResNet 50	RetinaNet	ViT
PQ	74.892	21.562	80.354
AdaR	75.322	27.258	79.590
PQ + AdaR	72.384	20.346	77.214

where ϵ is the tensor that encodes whether we should round up or down each scalar value of W independently. At the beginning of the optimization process of 10,000 steps, we set the value of ϵ such that $s(W) \times t^{-1}\left(\frac{t(W)}{s(W)} + \epsilon\right) = W$, *i.e.* we initialize ϵ such that, up to the rounding operation, the quantization process does not alter the value of W. As the optimization goes through, the epsilon value will encode whether we should ceil or floor the weight tensor. After the completion of the 10,000 steps, we update the weight values to their quantized counterparts: $W \leftarrow s(W) \times t^{-1}\left(\frac{t(W)}{s(W)} + \epsilon\right)$. In order to ensure that the values of ϵ tensor are bounded to $[0; 1]$, Nagel *et al.*, applied a variant $\tilde{\sigma}$ of the sigmoid function σ that reads

$$\tilde{\sigma} : a \mapsto \text{clip}_0^1\left(1.2 \times \sigma(a) - 0.1\right) \tag{3.27}$$

Furthermore, they added a regularization term to the loss function in order to push ϵ to the boundaries of its support, *i.e.* to force $\tilde{\sigma}(\epsilon)$ to be either 0 or 1. Consequently, the final optimization goal is

$$\mathcal{L} = \left\| \text{act}\left(s(W)s(X) \times t^{-1}\left(\left\lceil \frac{t(W)}{s(W)} + \epsilon \right\rfloor\right) \times t^{-1}\left(\left\lceil \frac{t(X_Q)}{s(X_Q)} \right\rfloor\right) + b \right) - f(X) \right\| + 1 - |2 \times \tilde{\sigma}(\epsilon) - 1|^{\beta} \tag{3.28}$$

where the β parameter influences the convergence pace of the rounding regularization term. For high values of β, a huge loss is put on ϵ values that are far from 0 and 1. The scheduler for β will be discussed in detail in the following sections.

In summary, the grounding for all GPTQ methods, called AdaRound, proposed to optimize the weight values in a self-distillation, sequential process which minimizes the distance between the intermediate features of the original and quantized models. To do so, we optimize during 10,000 stochastic gradient optimization steps over 1024 data points a variable ϵ which encode whether we should up or down the current weight tensor.

Our first contribution to the field of gradient-based post-training quantization was to bridge the gap between it and non-uniform quantization. At first, this work was done in NUPES [250], the journal extension of PowerQuant [255] and later on, we demonstrated the generalization to all non-uniform quantization methods in [249].

3.2.2 GPTQ and Non-Uniform Quantization

In [250], we proposed to address the two main shortcomings of the aforementioned PowerQuant data-free quantization operator: the optimization of the weight values and the optimization of the exponent parameter per-layer independently. First, let's detail how we integrate power quantization within the GPTQ framework.

Gradient Based Optimization of the Power Quantized Weights

In Table 3.16, we provide performance results from a straightforward combination of power quantization with AdaRound. We observe a significant accuracy drop. This phenomenon can be intuitively explained by the distortion of the quantized space from non-uniform operators. For instance, the power quantized space assigns a small step size to all values near 0 which limits the impact from the optimization of the ϵ tensor. As a result, ϵ can only induce marginal corrections. On the other hand, values that lay further away from 0 are stretched away and as such a change in the rounding operation would lead to a massive change in the predictive function which leads to instability in the GPTQ process.

In order to alleviate this limitation, we proposed to leverage the differentiable soft quantization (DSQ)

[73] activation function. This function emulates the rounding operation in a differentiable fashion,

$$\text{dsq}(\epsilon) = \frac{\tanh\left(\beta \times \left(\epsilon - \frac{1}{2} - \lfloor \epsilon \rfloor\right)\right)}{2\tanh\left(\frac{\beta}{2}\right)} + \lfloor \epsilon \rfloor + \frac{1}{2} \tag{3.29}$$

where the hyperparameter β plays a role similar to the one introduced in AdaRound in equation 3.28. Intuitively, the β parameter encodes the steepness of the DSQ function. For high values of β, we get a very steep function, which increases the cost for gradient-based optimization to change the value by more than ± 1. This phenomenon is illustrated in Figure 3.9. This leads to a change in the objective function, as the quantization itself performs the regularization:

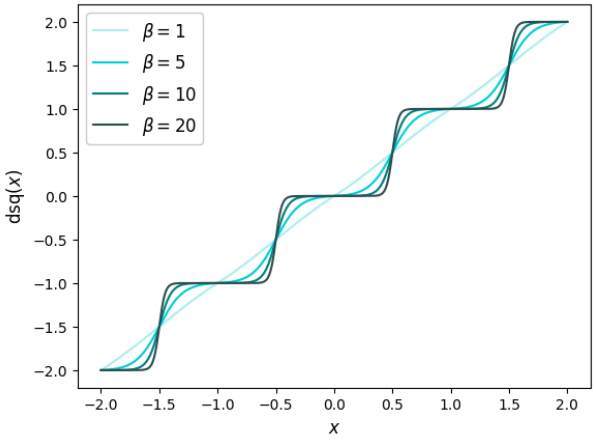


Figure 3.9: Illustration of the graph of the DSQ function for different values of the steepness hyperparameter β.

$$\mathcal{L} = \|f(X, W) - f_l\left(X_Q, \text{dsq}(W)\right)\|_2^2 . \tag{3.30}$$

The new objective function bares several advantages over the previous one. First, as we only need to compute one term, the runtime is slightly diminished. Furthermore, as we only use W and no longer need the ϵ tensor, the resulting optimization has a significantly lower memory footprint. This reduction is particularly important on transformer architectures, for which the weight tensors typically represent most of the memory usage as compared to the activations and intermediate features. Ultimately, the new optimization problem enables values to be shifted by more than one while maintaining a strong constraint in order to avoid overfitting.

As a result, this aspect of NUPES [250] addresses the first limitation of PowerQuant: we can learn weight values in a GPTQ manner. Furthermore, NUPES also allows learning the power exponent parameter, a which we propose to describe now.

Gradient based Optimization of the Power Exponent

In [255], we optimized a single power exponent a using the quantization error as an objective. The optimization, through gradient descent, of a per-layer power exponent parameter requires the computation of the following derivative

$$\frac{\partial X^a}{\partial a} = X^a log(X). \tag{3.31}$$

However, this operation introduces three numerical challenges:

- The power quantization transformation is applied to the weights and to their scale which means that the power operation is performed both at the numerator t and denominator s in equation 3.5. However, the scaling s can be analytically derived from the full-precision weights support and as such shall not be optimized through stochastic gradient descent. Consequently, in NUPES, we only account for the derivatives from the numerator in the backward propagation and derive the new scaling value (**update scale**) from the updated value of a.
- The naive implement of the gradient $\nabla_a X^a$ is not numerically stable. We identified two major pitfalls that cause this problem: strict zeros and values near zero of X. For strict zero values, the logarithm is not properly defined, which prevents the computation of $\nabla_a X^a$. In practice, this happens almost systematically for all networks, a naive example would be the intermediate features of a ReLU network. Regarding values near zero, the logarithm would compute an infinite value, due to the limitations of the floating point representation. In order to address both of these issues at once, we propose to clip the values of X (**num. stability**) to be at least 10^{-6} in magnitude.
- For a given layer, the gradient update comes from two transformations: the quantization of the weights and the quantization of the inputs. Before adding the two contributions, we propose to balance them out by averaging them along all the dimensions. This process introduces robustness to the batch-size and balances the contributions with respect to the weights and the inputs (**balanced grads**).

Algorithm 3 Learn the exponent parameter a

Require: a layer f_l with weights $W_l \in \mathbb{R}^{N\times M}$ and inputs $X \in \mathbb{R}^{B\times N}$
$\text{scale}_x \leftarrow s(sign(X) \times |X|^a)$ ▷ update scale (no ∇)
$\text{scale}_w \leftarrow s(sign(W_l) \times |W_l|^a)$ ▷ update scale (no ∇)

Forward Pass
$X \leftarrow \left\lfloor \frac{\text{sign}(X)\times|X|^a}{\text{scale}_x} \right\rceil$
$W_l \leftarrow \left\lfloor \frac{\text{sign}(W_l)\times|W_l|^a}{\text{scale}_w} \right\rceil$
$Y \leftarrow f_l(X, W_l)$

Backward Pass
$X_{\text{clipped}} \leftarrow \text{clip}(|X|, 10^{-6}, \infty)$ ▷ num stability
$W_{\text{clipped}} \leftarrow \text{clip}(|W|, 10^{-6}, \infty)$ ▷ num stability
$\nabla_{\text{from inputs}} \leftarrow \frac{X^a_{\text{clipped}} log(X_{\text{clipped}})}{B\times N}$
$\nabla_{\text{from weights}} \leftarrow \frac{W^a_{\text{clipped}} log(W_{\text{clipped}})}{N\times M}$
$\nabla_a \leftarrow \nabla_{\text{from inputs}} + \nabla_{\text{from weights}}$ ▷ balanced grads

Table 3.17: Evaluation of the different β schedulers for W4/A4 quantization. We also provide the performance of the PowerQuant method as a reference (ref).

model	AdaRound	Const_2	Const_{10}	Const_{20}	Power_1	Power_3	PowerQuant
ResNet 18	17.366	19.840	54.500	**64.528**	14.600	29.134	56.386
ResNet 34	26.934	38.940	59.402	**68.236**	31.708	31.150	62.904
ResNet 50	6.254	14.578	56.320	**68.758**	8.276	8.408	62.142
ResNet 101	4.842	9.980	53.328	**71.736**	22.622	23.054	64.562
RetinaNet	23.118	21.472	23.652	**32.692**	19.486	19.392	3.618
ViT b16	55.454	58.258	59.004	**79.578**	59.026	57.788	74.134
ViT l16	11.144	23.624	32.750	**34.544**	14.072	6.842	33.310
ViT h14	53.219	32.362	84.796	**87.190**	36.490	36.364	85.906

This protocol is summarized in Algorithm 3. However, it is worth noting that NUPES is also compatible with quantization-aware training [268]. All in all, NUPES solves the two main limitations of PowerQuant as we will showcase empirically in the upcoming sections.

Empirical Validation

As previously discussed, NUPES changes the exact role of the β parameter. In order to set it properly, we tested several scheduling strategies *i.e.* how to vary the value of the parameter β with respect to the current optimization step $\mathfrak{s}$. The first candidate is the scheduling applied in AdaRound [156] to control their approach to the β parameter, which has been leveraged in numerous subsequent works [128, 228, 139]. On top of this, we proposed our own candidates. Formally, they are defined as

$$\begin{cases} \text{AdaRound}(\mathfrak{s}) = 20 + \frac{-18}{2}\left(1 + \cos\left(\frac{\mathfrak{s}}{S}\pi\right)\right), \\ \text{Const}_c(\mathfrak{s}) = c, \\ \text{Power}_c(\mathfrak{s}) = 20\left(\frac{\mathfrak{s}}{S}\right)^c. \end{cases} \tag{3.32}$$

These schedulers are illustrated in Figure 3.10. In order to empirically validate the best suited scheduler, we evaluated them on several deep neural network architectures, as reported in Table 3.17. We clearly observe that the constant strategy vastly outperforms the other candidates in every task (classification and object detection) as well as architectures (convolutional neural networks and transformers). This suggests that, although it is important to allow the weights more flexibility as compared to AdaRound approach (recall Table 3.16), the weights should remain highly constrained throughout the optimization protocol. Stemming on this result and from now on, the NUPES method will be assumed to be performed with a constant steepness parameter $\beta = 20$. All in all, the NUPES GPTQ method requires less memory footprint and is more robust to hyperparameter tuning.

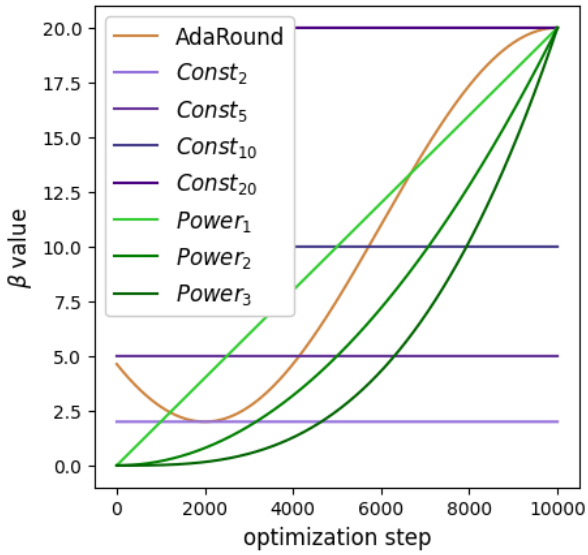


Figure 3.10: Graph plots of the β schedulers with respect to the number of optimization steps.

Table 3.18: Evaluation of the impact of learning the exponent (and not the weights) in W4/A4 quantization. We provide the full-precision original performance (accuracy or MAP) of the pre-trained model.

model	num. stability	balanced grads	update scale	Accuracy
ResNet50 (76.15)	✗	✗	✗	-
	✓	✗	✗	46,552
	✓	✓	✗	58.950
	✓	✗	✓	61.997
	✓	✓	✓	**62.468**
RetinaNet (37.294)	✗	✗	✗	-
	✓	✗	✗	4.084
	✓	✓	✗	4.128
	✓	✗	✓	19.838
	✓	✓	✓	22.424
ViT b16 (78.05)	✗	✗	✗	-
	✓	✗	✗	39.282
	✓	✓	✗	62.142
	✓	✗	✓	73,794
	✓	✓	✓	**74.552**

In Table 3.18, we evaluate the ability of NUPES to learn the power exponent parameter through stochastic gradient descent, for each layer independently and actually improve the accuracy of the quantized model. We considered the challenging int4 quantization configuration (W4/A4) and initialize the power values at 0.5. Our observations are two-fold. First, the absence of numerical stability safety nets simply prevents the learning process to ever complete (noted as "-"). More precisely, for ReLU networks, the process stops after the first optimization step of the second layer. For non-ReLU the process can complete about 4 optimization steps after introducing its first computational error. Second, across all tasks and architectures, balancing the contributions of the weights and activations, combined with the analytical scale update, systematically improve the final accuracy. The most impressive result are achieved on RetinaNet for object detection.

Overall, our empirical validation confirms the ability of NUPES to tackle PowerQuant limitations regarding weight and power exponent optimization. However, these result raise a question: how does NUPES compare with other GPTQ methods?

Table 3.19: Comparison to state-of-the-art GPTQ techniques. We report the W4/A4 quantized accuracies across convolutional neural networks and transformers. The first set of quantization methods are data-free while the second and third sets leverage a calibration set.

method	ResNet 18	ResNet 50	MobNet v2	EffNet B0	RetinaNet	ViT b16	ViT h14
full-precision	69.674	76.150	72.074	77.618	37.294	80.978	88.434
DFQ	29.602	28.548	0.232	0.112	0.256	3.354	0.176
SQuant	48.126	52.042	0.398	0.104	0.191	3.280	0.100
SPIQ	50.257	52.752	0.572	3.623	0.382	4.007	0.514
PowerQuant	56.386	62.142	0.348	3.618	2.241	74.134	85.906
AdaRound	60.258	61.656	8.840	0.102	21.392	29.906	23.070
BrecQ	28.650	63.782	38.230	0.110	18.922	22.228	25.686
QDrop	63.448	65.766	40.984	0.100	20.812	-	-
PDQuant	63.471	66.440	41.464	0.100	21.562	-	-
NUPES (learn a)	57.524	62.468	5.902	15.241	22.424	74.552	86.600
NUPES (learn W)	64.528	68.758	42.239	18.132	32.692	79.578	87.190
NUPES (learn W & a)	**65.876**	**70.684**	**42.386**	**45.902**	**33.078**	**80.100**	**87.204**

Comparison to State-of-the-art Quantization Methods

In Table 3.19, we report an extensive evaluation of post-training W4/A4 quantization techniques with proper implementation (as discussed in Chapter 1). We considered convolutional neural networks within the ResNet, MobileNet and EfficientNet families trained on ImageNet as well as RetinaNet for object detection on COCO [134]. We also included the smallest and largest ViT transformer architectures. We chose int4 quantization as this extreme setup highlights the performance and limits of the studied methods. Among data-free quantization techniques, even SQuant [43] and SPIQ [256] struggle to offer decent accuracy on the easier deep neural networks to quantize such as ResNet 50. On the other hand, PowerQuant offers a strong performance stepping stone for NUPES, especially on transformer architectures.

Among GPTQ methods, NUPES achieves state-of-the-art performance on all benchmarked neural network using weight optimization only. The most impressive results are obtained on RetinaNet and EfficientNet B0 where NUPES improves the precision of the model by 11.130 and 14.514 points, respectively. Furthermore, on ResNet architectures, PowerQuant already achieves results close to AdaRound and BrecQ [128], which enables NUPES to outperform both methods by a significant margin. On the other hand, the more recent extensions of AdaRound in QDrop [228] and PD-Quant [139] offer strong improvements on MobileNet architectures but are less effective on other architectures. The fact the gap is narrower here, can be explained by several elements: first, we applied the same optimization as in AdaRound with 10,000 optimization steps while BrecQ, QDrop and PD-Quant leveraged 20,000 steps, which hinders the scaling of these methods to larger models. Second, PD-Quant and QDrop leverage optimization tricks that require to run the entire model while optimizing each individual layer, which we also did not opt for easier scalability. Still, NUPES manages to outperform both QDrop and PD-Quant on every benchmark.

In particular, on transformer architectures, it appears that PowerQuant and NUPES both vastly outperform every other method, which reflects the much peaker distributions that characterize their weights and intermediate features. Furthermore, NUPES enables us to achieve near full-precision accuracy quantized in W4/A4 quantization. Consequently, these results show the strength of NUPES as a GPTQ technique.

However, NUPES is non-uniform, which makes it more complex to leverage than uniform quantization. During this thesis, we also worked on improving gradient-based post-training uniform quantization by studying each of the implicit assumptions of AdaRound. In the following section, we propose to discuss this work [249] in more details.

3.2.3 Best Practices

In the description of the AdaRound process, we made several assumptions that were well motivated. Prime examples of such priors were the sequential approach and the use of the full-precision model as a teacher. On the other hand, some priors were empirically validated in the initial article, such as the calibration set size and the optimization batch-size.

The following iterations over AdaRound [156] such as BrecQ [128], QDrop [228] and PD-Quant [139], questioned some other implicit assumptions. For instance, BrecQ investigated the choice of working at the layer level. Their findings confirmed that better accuracies were achieved using sub-parts of the network as compared to using the whole network. However, they also showed that using computational blocks such as residual blocks in ResNets or transformer blocks in transformers led to higher performance. In [228], the authors studied the importance of always quantizing the input features of the optimization block and figured that randomly drop the quantization of the features held a slight accuracy improvement. Ultimately, in [139], the authors leveraged extra loss terms including guidance from the batch-normalization (to enforce the preservation of the learned first and second order statistics) with further guidance from the final layers, *i.e.* optimize each block with its own similarity objective plus the similarity objectives with respect to the final outputs of the model.

While these studies helped to clarify the influence and role of some implicit priors in AdaRound, there remains numerous unanswered questions. In the following sections, we will present all the priors that we tested, starting with the ones that did not lead to any significant performance improvement[4].

Properly set Priors for GPTQ

The first set of priors that we are questioning are: the relevance of focusing on optimizing all the weight values in the weight tensor W for a given layer, the choice of the optimizing the bias term, the use of feature augmentation and the definition of the similarity objective.

In order to evaluate every prior, we propose a systematic evaluation with AdaRound and BrecQ on ResNet 50, MobileNet v2 and ViT. Intuitively, we propose to focus on learning to round up or down the most ambiguous weight values with respect to the quantization process. Stemming on the definition of AdaRound, we initialize the ϵ tensor using

$$\epsilon = \tilde{\sigma}^{-1}\left(t\left(\frac{W}{s(W)}\right) - \frac{t(W)}{s(W)}\right) \tag{3.33}$$

Weight ambiguity: From this definition, we can define the ambiguity of a scalar weight value by the distance of ϵ to 0.5. In Table 3.20, we provide the empirical results from focusing on the most ambiguous weight values. The provided results were obtained after tuning the proportion of weight values to study. However, whether we should focus especially on the most ambiguous or least ambiguous values would lead to a performance degradation. Intuitively, this highlights the fact that optimization, in the GPTQ context, leverages every ϵ values regardless of its initialization. However, it is worth clarifying that initializing ϵ randomly leads to a systematic 0.1 accuracy for all networks and methods. In other words, the initialization is important but is not a good indicator of whether the optimization will change the rounding operation or not.

Weight magnitude: In Table 3.21, we proposed to focus on the values ϵ based on the weight scalars magnitude. Intuitively, our previous work showed that focusing the quantization efforts on values based on their magnitude led to significant accuracy improvements NUPES [250] and PowerQuant [255]). Based on our empirical evidence, it appears that the latter does not hold. Formally, only optimizing low values to preserve the semantic like in AdaRound does not lead to an improved accuracy, to the contrary.

Biases: Following the previous priors evaluation, we studied the optimization of the bias tensor during the GPTQ process. In Table 3.22, we proposed to constrain the bias values in optimization with respect to their magnitude. This holds similarity with the constraint imposed on the weight values with $\tilde{\sigma}(\epsilon)$ being contrived to $[0; 1]$ (from equations 3.27 and 3.28). Through our experiments, it appeared that regardless of the constraint set on the biases, it always leads to an accuracy degradation as compared to not optimizing the bias. This can be attributed to the fact that bias are prone to overfitting, especially on a small calibration set.

[4]It is not common to publish or comment on research that "does not work" but we believe that these elements could help future researcher not focus on leads that are unlikely to be successful.

Table 3.20: Performance of GPTQ methods with respect to their focus on weights based on the ambiguity notion. We either optimize the 0.5% least ambiguous (non-ambiguous column) or the 0.5% most ambiguous (ambiguous column) weight values. We use W4/A4 for ResNet 50 and ViT while we use W4/A8 for MobileNet v2.

AdaRound			
Architecture	non-ambiguous	ambiguous	baseline
ResNet 50	61.002	60.452	61.318
MobileNet v2	64.547	64.712	65.314
ViT b16	31.103	31.045	31.256
BrecQ			
Architecture	non-ambiguous	ambiguous	baseline
ResNet 50	62.972	62.808	63.644
MobileNet v2	49.761	50.002	50.130
ViT b16	57.904	56.180	57.952

Table 3.21: Performance of GPTQ methods with respect to the focus on weights by their magnitude. We either optimize the 10% smallest values (low values) or the 10% highest values (high values). We use the same quantization as in Table 3.20.

AdaRound			
Architecture	low values	high values	baseline
ResNet 50	60.402	59.481	61.318
MobileNet v2	64.836	64.119	65.314
ViT b16	30.614	30.166	31.256

BrecQ			
Architecture	low values	high values	baseline
ResNet 50	63.216	62.515	63.644
MobileNet v2	50.077	49.653	50.130
ViT b16	57.090	56.991	57.952

Table 3.22: Performance of GPTQ methods with different constraint on the optimization of the biases. For a constraint $\alpha = 1$, we do not update the biases which corresponds to the baseline methods.

	AdaRound			
Architecture	$\alpha = 0$	$\alpha = 0.33$	$\alpha = 0.66$	$\alpha = 1$
ResNet 50	61.318	30.423	0.100	0.100
MobileNet v2	65.314	11.612	0.100	0.100
ViT b16	31.256	9.742	0.100	0.100
	BrecQ			
Architecture	$\alpha = 0$	$\alpha = 0.33$	$\alpha = 0.66$	$\alpha = 1$
ResNet 50	63.849	26.836	0.100	0.100
MobileNet v2	50.130	5.010	0.100	0.100
ViT b16	57.952	7.930	0.100	0.100

Data augmentation: In order to reduce the risk of overfitting, a standard approach in deep learning consists in using data augmentation. In the case of GPTQ optimization, we propose to perform feature augmentation [223] including the most commonly used ones: dropout [197], mixup [261], cutout [53] and noise [69]. In Table 3.23, we show that while some augmentation techniques sometimes slightly improve the final accuracy, this result is not stable across different architectures, especially transformers.

Similarity loss: Ultimately, we wanted to assert whether the proposed definition of the similarity between intermediate features using the Euclidean norm was best. As a comparison, we considered the l_1 loss, which puts less emphasis on extreme values. Similarly, we also considered two other distribution similarity metrics, e.g. the cosine and the Kullback-Leibler divergence (KL) [115]. From our empirical results, in Table 3.24, it appears that optimizing over l_1 objective enables to find different trade-offs in terms of performance *v.s.* architecture, the Euclidean norm remains the best default solution in the general case.

While these aspects of the GPTQ methods do not appear to bear interesting results, they enable us to confirm some intuitions, e.g. the biases are too prone to overfitting, and also refute some wrong intuitions, e.g. the initialization is not a good indicator of the importance a weight will play in the optimization.

Table 3.23: Performance of GPTQ methods in combination with intermediate features augmentation. We use the same quantization as in Table 3.20. We highlight in bold the results that show an improvement over the baseline without augmentations.

	AdaRound			
Architecture	+ Dropout	+ Mix-up	+ Cut-out	+ Noise
ResNet 50	**61.632**	60.336	61.306	55.316
MobileNet v2	65.178	**65.788**	**65.864**	62.506
ViT b16	29.780	30.422	29.874	**31.364**
	BrecQ			
Architecture	+ Dropout	+ Mix-up	+ Cut-out	+ Noise
ResNet 50	**63.689**	62.396	**63.843**	57.870
MobileNet v2	**50.321**	**50.151**	**51.052**	46.974
ViT b16	56.331	57.453	56.161	57.944

Table 3.24: Influence of the similarity goal in GPTQ methods. We use the same quantization as in Table 3.20. We highlight in bold the results that show an improvement over the baseline l_2.

	AdaRound			
Architecture	l_1	cosine	KL	l_2
ResNet 50	42.716	16.398	59.738	61.318
MobileNet v2	**65.630**	0.100	0.100	65.314
ViT b16	**34.460**	0.100	**38.714**	31.256

	BrecQ			
Architecture	l_1	cosine	KL	l_2
ResNet 50	46.586	25.864	60.818	63.644
MobileNet v2	49.932	0.100	0.100	50.130
ViT b16	57.400	0.100	**60.524**	57.952

Table 3.25: Evaluation of the proposed mixed-precision method.

	AdaRound		BrecQ	
Architecture	fixed	mixed	fixed	mixed
ResNet 50	61.318	**64.675**	63.849	**66.181**
MobileNet v2	65.314	**67.402**	50.130	**56.756**
ViT b16	31.256	**35.179**	57.952	**59.000**

Still, the study of these priors did not lead to any significant improvements. In the following section, we propose two aspects from which systematic improvement can be derived for all GPTQ methods.

Priors to Improve GPTQ

Mixed-precision: In order to improve GPTQ performance, we want to assign the appropriate precision to each operation, *i.e.* assign the appropriate bit-width to each output neuron. To do so, we proposed a novel approach to mixed-precision with GPTQ methods, stemming on our previous work on importance estimation from attribution techniques. For a given output neuron n of a layer f, we measure their accumulated gradients g_n with respect to the model outputs. We use the g_n as an estimation for the sensitivity of the predictive function to the given neuron. In order to assign the appropriate bit-width to each neuron, we compute the mean μ and standard deviation σ among gradients g_n. The final bit-widths $\mathfrak{b}_n$ of each neuron n is given by the target average number of bits $\mathfrak{b}$ and the distance to the mean μ in terms of standard deviations, *i.e.* $\mathfrak{b}_n = \mathfrak{b} + \left\lfloor \frac{g_n - \mu}{\sigma} \right\rceil$ where $\lfloor \cdot \rceil$ rounds towards zero (e.g. $\lfloor 0.9 \rceil = 0$, $\lfloor -0.9 \rceil = 0$ or $\lfloor 2.1 \rceil = 2$). As showcased in Table 3.25, the proposed mixed-precision scheme improves the performance of both AdaRound and BrecQ on all the tested architectures, by a significant margin.

Optimizer: On the other hand, a second prior which can be improved over is the optimizer choice. In Table 3.26, we report the performance of several of the most commonly used optimizers. Empirically, using AdaMax [111] allows to systematically improve the baseline Adam optimizer. AdaMax leverages the infinite norm of the gradients rather their Euclidean norm to normalize the update steps. We argue that the AdaMax process is best suited for noisy and slightly unstable optimization processes such as quantization. While this result holds true for uniform, we need a method to apply GPTQ to all non-uniform quantization as illustrated in Figure 3.11. This issue was addressed in NUPES for power quantization. In Table 3.27, we provide empirical evidence to the ability of the proposed to generalize to other non-uniform

Table 3.26: Performance of GPTQ methods with different gradient-based optimizers (we recall that the default option is Adam). For all the considered setups, we use the default parameters in order to avoid hyper-tuning and unfair comparisons. We use the same quantization as in Table 3.20. We highlight in bold the results that show an improvement over the baseline.

AdaRound								
Architecture	SGD	Nesterov	Adam	AdamW	Adamax	AdaGrad	AdaDelta	RMSProp
ResNet 50	60.616	60.384	61.318	**61.434**	**62.830**	**62.102**	49.984	61.924
MobileNet v2	38.430	31.962	65.314	**65.374**	**65.824**	14.232	1.304	**66.236**
ViT b16	8.326	8.530	31.256	30.462	**39.234**	9.916	4.654	30.328
BrecQ								
Architecture	SGD	Nesterov	Adam	AdamW	Adamax	AdaGrad	AdaDelta	RMSProp
ResNet 50	62.653	62.428	63.644	**63.849**	**65.374**	**64.677**	53.293	63.625
MobileNet v2	22.807	17.014	50.130	**50.279**	**50.616**	0.100	0.100	**50.184**
ViT b16	34.849	35.516	57.952	56.804	**59.325**	36.627	31.517	56.274

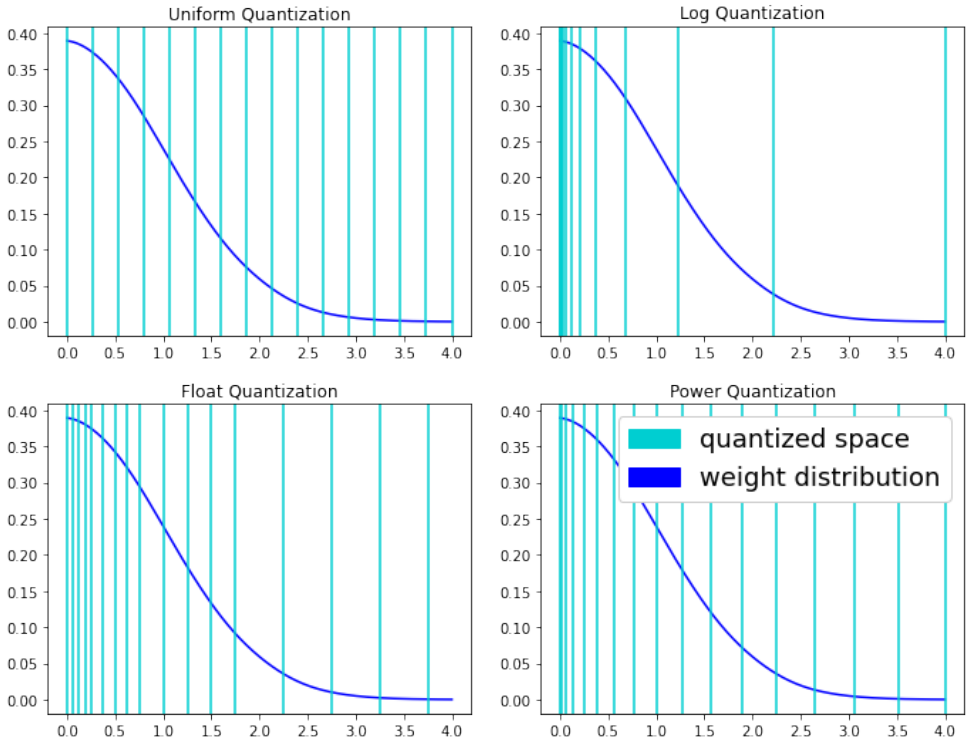


Figure 3.11: Illustration of the quantization in 4 bits of the positive part of a Gaussian distribution in uniform, logarithm, floating point and power quantization (from left to right, top to down). This highlights the balance between precision for very low bit values and larger values that is achieved with PowerQuant *versus* other quantization formats.

quantization. Furthermore, we even show that it can be applied to uniform quantization as well for the memory footprint benefits.

As a result, through our study of the gradient-based post-training quantization process, we highlighted elements over which improvements could be achieved, such as the optimizer choice and mixed-precision. In Table 3.28, we provide a summary of the performance benefits that are achieved through these best practices. These results should motivate further investigation in those directions. Finally, we wanted to test a more industrial oriented aspect of GPTQ: the choice of the calibration set.

GPTQ robustness

In academic evaluation, the calibration set is assumed to be randomly sampled from the training set. As such, it is in-distribution with respect to the trained neural network. However, in practice, we may

Table 3.27: Adaptation of GPTQ methods to non-uniform quantization such as logarithmic quantization (log) low-bit floating point representation (4 bits float) and power quantization [255]. We propose to also apply the adaptation to uniform quantization as well for its memory footprint benefits. We use the same quantization setup as in Table 3.20.

	AdaRound				NUPES			
Architecture	uniform	log	float	power	uniform	log	float	power
ResNet 50	61.318	0.304	45.758	0.100	61.396	60.688	60.170	68.546
MobileNet v2	65.314	0.100	22.134	0.100	64.790	64.866	65.028	66.014
ViT b16	31.256	0.138	6.056	0.100	33.018	35.364	32.082	79.578

Table 3.28: Summary of the added value from the best practices we propose.

	AdaRound		BrecQ		NUPES	
Architecture	standard	best	standard	best	standard	best
ResNet 50	61.31	65.10 (+**3.78**)	63.64	66.25 (+**2.60**)	68.54	70.29 (+**1.75**)
MobileNet v2	65.31	68.38 (+**3.06**)	50.13	56.75 (+**6.62**)	66.01	67.34 (+**1.33**)
ViT b16	31.25	38.07 (+**6.81**)	57.95	59.89 (+**1.94**)	79.57	80.20 (+**0.62**)

Table 3.29: We measure the impact of the data "quality". We consider training on a subset of the test set of ImageNet (ImNet val), the standard train set (ImNet train), some adversarial [87] and out-of-distribution [196] sets for ImageNet models. Finally, we considered two extreme scenarios: MNIST [49] (rescaled to 224×224) and white noises. We use the same quantization as in Table 3.20.

AdaRound						
Architecture	ImNet (val)	ImNet (train)	ImNet (adv)	ImNet (ood)	Mnist	White Noise
ResNet 50	62.908	61.318	64.174	66.030	18.118	6.944
MobileNet v2	66.428	65.314	67.068	67.084	37.494	33.466
ViT b16	33.258	31.256	27.262	26.740	12.762	29.974
BrecQ						
Architecture	ImNet (val)	ImNet (train)	ImNet (adv)	ImNet (ood)	Mnist	White Noise
ResNet 50	64.996	63.644	66.644	68.476	20.939	8.987
MobileNet v2	51.200	50.130	51.888	51.940	22.071	18.071
ViT b16	60.006	57.952	54.249	53.047	39.308	56.391

have to compress a model which was trained on proprietary data. In order to avoid privacy concerns, we wanted to know the importance of using in-distribution data as compared to out-of-distribution data. In Table 3.29, we evaluated AdaRound and BrecQ using several examples of out-of-distribution datasets. Empirical evidence suggests that up to the use of extreme examples, such as MNIST and white noise, the data distribution is not crucial. This has significant implications for industrial applications, as it suggests that using default out-of-distribution data does not hinder GPTQ methods. Furthermore, it is known that GPTQ methods outperform data-generative methods. Furthermore, if freely available data leads to both higher accuracy and lower process costs, then this results significantly hinders the motivation to study and leverage data-generative quantization.

Stemming on all the aforementioned results, we provide a set of best practices for post-training quantization regardless of the granularity, data availability and support for non-uniform quantization.

While both data-free quantization and GPTQ are convenient in terms of scalability, processing costs and achieve strong compression trade-offs in terms of accuracy *v.s.* speed, higher compression rates can be achieved through quantization-aware training at the expense of higher training costs.

3.3 Quantization-Aware Training

While data-free quantization and gradient-based post training quantization offer scalable and relatively low cost compression methods, they are unable to achieve decent accuracies in the highest compression format: binary quantization. This is performed through quantization-aware training and results in most computations being represented using only two values: ± 1. In the following section, we will detail the current state-of-the-art QAT methods: ReActNet [142] and PokeBNN [265].

3.3.1 ReActNet and PokeBNN

The general challenge of quantization-aware training is the rounding operation, which introduces zero gradients almost everywhere and is undefined elsewhere. As discussed in the first chapter, the most common solution to gradient-based optimization of rounded values, in deep quantization is the straight-through estimator (STE) which simply omits the rounding step during the back-propagation.

In recent research on binary neural networks (BNNs), various elements have been introduced in order to narrow the gap between binary and full-precision models. The first set of techniques revolve around the introduction of non-binary operations: the idea is to use as many binary operations as possible while allowing for a few well selected operations to take place in higher precision in order to preserve the accuracy. Formally, in ReActNet [142], the authors proposed to add an identity shortcut to the MobileNet v1 blocks in order to carry over higher precision features. This technique, on its own, improved the accuracy by over 10 points on ImageNet as compared to previous work on BNNs. Similarly, in PokeBNN [265], the BNN benefits from 4 bits squeeze-and-excite (SE) modules [96] which helps select the relevant features for a given input and are computationally cheap by design. The use of SE modules has been tested in other works on BNNs [144] but with less success. In PokeBNN, the authors copy the architecture of a MobileNet v3, a well known compact neural network. As a result, current BNNs do not implement every operation using the binary format, but rather add workarounds using higher precision.

Furthermore, a common trademark of recent BNNs is the use of zero padding for convolutional layers. As a result, these layers are not proper binary layers but rather ternary, as binarization only uses $\{-1, 1\}$ values. In the community and the literature, we do not have a specific term to refer to these methods. For clarity, it would perhaps be preferable to say: binary quantization for tensors properly quantized to $\{-1, 1\}$ values, padded binary tensors for works in ReActNet and PokeBNN and ternary tensors for other tensors quantized to $\{-1, 0, 1\}$ values. The difference between the two last is the constraint on the positions of the zeros, which is non-existent for standard ternary quantization. This issue has been studied in [79] and is addressed by using an alternate padding of $\{-1, 1\}$. However, in practice, almost no BNN works actually evaluate their method with this padding technique.

While padding has a non-negligible impact on the input distribution, in the context of binary quantization, it is far less significant than the distribution of the features themselves. Formally, binary quantization is implemented *via* the sign function which maps $\mathbb{R}^*$ to $\{-1, 1\}$ with the acceptance that a strict zero is so rare that most implementations do not account for it (this is debatable but widely adopted). This makes BNN incompatible with the ReLU activation function. Furthermore, during back-propagation, every value outputted by the activation function that are binarized will have the same gradients magnitude. In order to alleviate these shortcomings, ReActNet and PokeBNN introduce their own activation functions: RPReLU and DPReLU respectively:

$$\begin{aligned} \text{RPReLU} : x \mapsto \begin{cases} x - \gamma + \zeta & \text{if } x > \gamma \\ \beta(x - \gamma) + \zeta \end{cases} \\ \text{DPReLU} : x \mapsto \begin{cases} \alpha(x - \gamma) + \zeta & \text{if } x > \gamma \\ \beta(x - \gamma) + \zeta \end{cases} \end{aligned} \tag{3.34}$$

where γ, ζ, α and β are learned, real valued, parameters. We uniformized the notations from PokeBNN and ReActNet in order to highlight the fact that the DPReLU from PokeBNN has more flexibility and can encode the RPReLU from ReActNet. These activation functions enable a finer control over the optimization process. In PokeBNN, the authors add a gradient clipping in $[-3; 3]$, to further stabilize the training process.

Overall, BNNs are trained using large batch-sizes (512 and 2048 as compared to 128 for the full-precision model) over more epochs (512 and 750 as compared to 200). Regarding the optimization process, there is one remaining key specificity of BNNs: the loss function. In order to benefit from the soft labels learned by a full-precision model, modern BNNs are trained using the Kullback-Leibler [115]

divergence from a pre-trained ResNet or MobileNet v3. Consequently, the whole training process requires the full pre-training of floating point model and uses knowledge distillation. It is followed by a longer and computationally more intensive training of the final BNN.

In summary, the current iterations of BNNs use higher precision shortcuts in order to preserve the model performance and a non-binary padding. These models are trained using knowledge distillation over a computationally intensive training process.

However, the most important pitfall for these methods is their specificity to convolutional architectures. While some research are conducted for binary transformers, these are usually limited to the straightforward adaptation of previous research and do not leverage transformer-specific operations. In the following section, we outline the leads that we would follow for transformer binarization.

3.3.2 Leads on Binary Transformers

In the early months of this PhD thesis, we wondered what was the root cause of the difficulty from binarization. Our insight was that BNNs struggle to assign a distinct importance to their inputs. Formally, let F^b be a BNN such that

$$F^{(b)} = f_L^{(b)} \circ \cdots \circ f_1^{(b)} \tag{3.35}$$

with

$$\forall l \in [\![1; L]\!], \quad f_l^{(b)} : x \mapsto \sigma(W_l^{(b)} x + b_l^{(b)}) \tag{3.36}$$

where σ is the sign function, $W_l^{(b)} \in \{-1, 1\}^{n_l \times n_{l-1}}$ the kernel matrix and $b_l^{(b)} \in \{-1, 1\}^{n_l}$. For each layer, we can immediately assert that the output dimension of the layer l is lower than n_{l-1}

$$f_l^{(b)} : \{-1, 1\}^{n_{l-1}} \to E_l^{(b)} \subset \{-1, 1\}^{n_l} \tag{3.37}$$

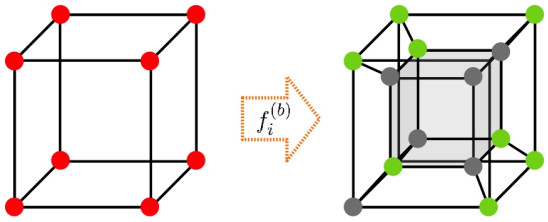


Figure 3.12: In this figure, we illustrate how all the vertices of the cubes aren't enough to map all the vertices of the tesseract. This corresponds to $f_l^{(b)} : \{-1, 1\}^3 \to E_l^{(b)} \subset \{-1, 1\}^4$.

Because $W_l^{(b)} \cdot + b_l^{(b)}$ is an affine function, we get up to $|\{-1, 1\}^{n_{l-1}}| = 2^{n_{l-1}}$ different outputs. σ is surjective, thus $|E_l^{(b)}| \leq 2^{n_{l-1}}$ i.e. $E_l^{(b)} \sim \{-1, 1\}^{\tilde{n}_l}$ with $\tilde{n}_l \leq n_{l-1}$. Therefore, recursively, we can state that for any l we have $\dim(E_l^{(b)}) \leq n_0$. This result is only due to the nature of the inputs and the basic properties of the layers. This is illustrated in fig 3.12 for the case $n_{l-1} = 3$ and $n_l = 4$, this figure shows that points of the tesseract won't have a pre-image.

Note that if you consider an input space such as $\mathbb{R}^{n_0}$, simply transforming it into $\{-1, 1\}^{n_0}$ may be problematic, as we just saw. A simple and often used way to tackle this burdensome limitation consists in using a binarized encoding of the inputs and thus transforming the input space into $\{-1, 1\}^{32 n_0}$.

Let's note $W^{(b)}$ a binary kernel and $X^{(b)}$ all the possible concatenated inputs in $\{-1, 1\}^n$. All the possible outputs are noted $Y^{(b)}$ and as previously shown $Y^{(b)} \in \{-1, 1\}^{m \times 2^n}$ is composed of up to $\{-1, 1\}^n$ different vectors. In the case $n = 2$, we have,

$$\sigma(A^{(b)} \times X^{(b)}) = \sigma(Y^{(b)}) = \begin{pmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & 1 & -1 \\ -1 & 1 & 1 & 1 \end{pmatrix} \tag{3.38}$$

Let's consider the set $\mathcal{Y}_n \subset \mathbb{Z}$ of all possible scalar products of vectors in $\{-1, 1\}^n$. If we consider two vectors $x^{(1)}, x^{(2)} \in \{-1, 1\}^n$, then for any $k \in [\![1; n]\!]$ we have $x_k^{(1)} x_k^{(2)} \in \{-1, 1\}$ thus, the value of $\langle x^{(1)}, x^{(2)} \rangle$ is defined by the number $\tilde{n}$ of $k \in [\![1; n]\!]$ such that $x_k^{(1)} x_k^{(2)} = 1$. Then we

$$\langle x^{(1)}, x^{(2)} \rangle = 2\tilde{n} - n \tag{3.39}$$

From this equation, we can deduce that

$$\mathcal{Y}_n = \begin{cases} 2\mathbb{Z} \cap [\![-n; n]\!] & \text{if } n \in 2\mathbb{N} \\ (2\mathbb{Z} + 1) \cap [\![-n; n]\!] & \text{if } n \in (2\mathbb{N} + 1) \end{cases} \tag{3.40}$$

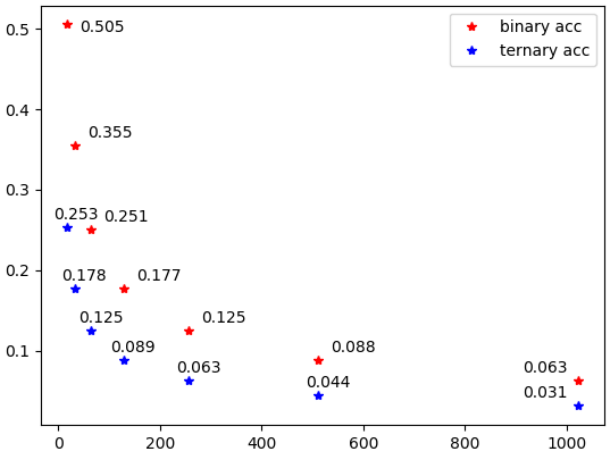


Figure 3.14: In this figure, we show some values of the angular distance θ for Binary and Ternary representations between a vertex of $\mathbb{K}^{n_l}$ and its immediate neighbors.

Either way, we have $|\mathcal{Y}_n| = n + 1$. The output Y lives in $\tilde{E}^{(b)} \subset \mathcal{Y}_n^{m \times 2^n}$, with $|\tilde{E}^{(b)}| \leq |\{-1, 1\}^n|$. Then $\sigma(Y) \in E^{(b)} \subset |\{-1, 1\}^m|$, with $|E^{(b)}| \leq |\{-1, 1\}^n|$. In fact, we can define a binary kernel such that $|E^{(b)}| = |\{-1, 1\}^n|$ simply by taking $m = 2^n$ and $A^{(b)} = (X^{(b)})^T$. Thus, the kernel doesn't limit the representative power of a BNN more than the inputs.

In a BNN, each neuron is given the same absolute importance. We can see $y = f_1^{(b)}(x)$ as a concatenation of functions learned (i.e. neurons). Now let's assume that we don't have duplicates, i.e. all the rows of $W^{(b)}$ are different. Then each different information computed will impact the prediction similarly. This may be wanted with an adequate number of parameters. However, in practice this is not predictable for a given architecture yet.

To tackle this problem, we can now allow for the presence of duplicates. Duplicates allow us to simulate a scaling parameter λ

$$\sigma(W^{(b)}x) = \lambda \odot \sigma(\tilde{W}^{(b)}x) \tag{3.41}$$

where $\tilde{W}^{(b)}$ doesn't contain duplicates and $\lambda \in \mathcal{Y}_{d_1} \times \cdots \times \mathcal{Y}_{d_m}$ is the scaling vector, with d_k the number of duplicates of neuron k of $\tilde{W}^{(b)}$. This is illustrated in fig 3.13. Note that this also allows the network to cancel a neuron's activation.

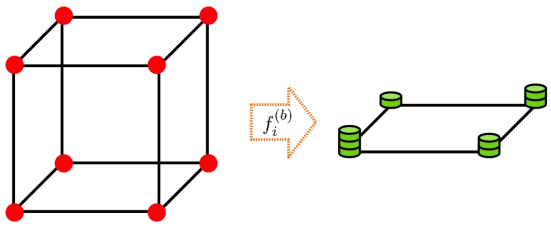


Figure 3.13: In this figure, we show how identical neurons allow stacking identical predictions in order to simulate an integer scaling.

Each layer $f_l^{(b)} : \{-1; 1\}^{n_{l-1}} \to \{-1; 1\}^{n_l}$ of a BNN is a function that selects positive and negative data for each coordinate to place it on a vertex of the hyper-cube $[-1; 1]^{n_l}$. Let's note $\mathbb{K}^{n_l}$ the set $\{-1; 1\}^{n_l}$ in $\mathbb{R}^{n_l}$. Then we have

$$\forall x, \quad \|f_l(x)\|_2^2 = \sqrt{n_l}, \text{ i.e. } \mathbb{K}^{n_l} \subset \sqrt{n_h}\mathbb{S}^{n_l} \tag{3.42}$$

where $\mathbb{S}^{n_l}$ is the unit sphere in $\mathbb{R}^{n_l}$. Also, the minimal geometrical angle θ between two elements of $\mathbb{K}^{n_l}$ can be computed as

$$\theta = \arccos\left(\frac{n_i - 2}{n_l}\right) \tag{3.43}$$

This result comes from the fact that, for any pair, $x \neq y \in \mathbb{K}^{n_l}$ the maximum of their scalar product is reached when x and y are equal on $n_l - 1$ coordinates and different on the last. In this case, the value of their scalar product is $n_l - 2$. Some values of θ, for different values of n_l, are shown in Fig 3.14. As we can observe, binary quantization induces a significantly higher angular error than ternary quantization. In order to get similar performance as a ternary layer with 128 neurons, a binary layer requires 512 neurons. This highlights the difficulty of disentangling features for BNNs.

Now let's compute the maximum angular error $\bar{\theta}$ between any element s of $\mathbb{S}^{n_l}$ and its closest element k_s in $\mathbb{K}^{n_l}$. Let's first note that the angle between k_s and s is equal to the angle between k_s and the

projection k^s of s on the cube $[-1;1]^{n_l}$. Then the solution for the maximum is reached when k^s is the projection of barycenter of a face of $[-1;1]^{n_l}$ on $\mathbb{S}^{n_{l-1}}$, whose faces are all identical, up to a rotation. Therefore, the maximum is reached for $(k^s)_i = (k_s)_i$ on $n_l - 2$ coordinates and $(k^s)_j = 0$ on the others: thus, we have $\bar{\theta} = \theta$. A simple calculus can show that in the case of Ternary Neural Networks (TNN) we have a maximum angular error θ_T of

$$\theta_T = \frac{1}{2} \arccos\left(\frac{n_l - 2}{n_l}\right) \tag{3.44}$$

Novel Approach to Binary Quantization

As suggested, it is of paramount importance to share higher precision data and, in particular, tensors with at least zero as an extra value. Consequently, our conclusions are two-fold:

- First, we would reduce the number of output neurons of each layer by a factor k and implement a multiplicative variable ω independently for each neuron which would be quantized to an unsigned two bits value to encode the number of duplications of the corresponding neuron (as in fig 3.13). Then, after training, I would duplicate the neurons accordingly in order to retrieve a proper BNN.
- Second, we would add zeros to the neural network computations in a constrained manner through the positional encoding in order to mitigate the angular error from binary quantization.
- Third, we would investigate an efficient way to perform the attention over binarized tensors: use binary values vectors to encode a rotation and use a scalar to encode the magnitude using a higher precision. This is equivalent to a decomposition of the attention mechanism in polar coordinates.

These are our first insights towards future work in deep acceleration and compression. However, there remains a wide range of such applications that can be further improved. In the following chapter, we propose a set of research leads which are, in our perspective, of particular interest of the compression community.

Chapter 4

Insights for Future Work

In this final chapter of this thesis, we propose a summary of the work accomplished during this thesis, as well as five leads on future deep compression related research. The outline for this chapter reads as follows:

1. We summarize the contributions of this thesis to the field of deep neural network compression.
2. We propose a new perspective for deep neural networks pruning, which changes the notion of granularity and may pave the way towards new techniques for low power devices.
3. We highlight the most significant pitfall of current gradient-based post training quantization techniques regarding the trending generative AI models.
4. We conclude on what we consider to be a future shortcoming in deep quantization: group-wise quantization.
5. We provide a theoretical discussion around the reasons why modular arithmetic is often overlooked in the quantization literature.
6. We discuss a new approach to efficient training based on the trending adapters.

4.1 Our Contributions to Compression

As previously detailed, we contributed to several aspects of deep neural network compression. In summary, we tackled both data-free and data-driven pruning, as well as post-training quantization. Through our study of pruning, we compared redundancy-based [252, 253] and importance-based [254] approaches, which led to several publications (see Appendix 5 for a list of the publications). On the flip side, regarding quantization, we first introduced an optimal solution to a crucial step: batch-normalization folding [247]. Then, we tackled three major challenges of data-free quantization: granularity [256], uniformity [255] and hardware support [246]. Following these findings, we investigated GPTQ methods and bridged the gap with non-uniform quantization [250, 249].

On top of our work on compression, we studied how to adapt our methods to other applications, such as robust inference [251] and fighting overfitting [208].

In summary, during this PhD thesis, we discovered, studied and contributed to the field of deep neural networks acceleration and compression. In this manuscript, we presented a general landscape of the domain based on our accumulated knowledge and also detailed our own contributions. We hope you had enjoyed reading this manuscript as much as we enjoyed learning research under the mentorship of Arnaud Dapogny and Kevin Bailly.

Still, there are many remaining challenges in deep compression. Some of which are of particular interest to us. Consequently, in the remainder of this manuscript, we share our insights on how to address them.

4.2 Pruning Matrix Multiplication Algorithms

This subject has been introduced at the end of chapter 2 (page 54). On larger devices, such as GPUs, it is almost systematically more efficient to perform multiple operations in parallel on contiguous memory segments rather than computing a sequence of fewer operations. On the other hand, on very low power, edge devices, fewer operations often leads to a lower power consumption. In particular, in the context of matrix multiplications, algorithm optimization consists in the search of a set of operations requiring fewer multiplications in order to achieve the exact same result. The prime example of such algorithm is the Strassen algorithm. We recall the naive and Strassen matrix multiplication algorithms:

$$\begin{aligned} \text{Strassen} \quad A \times B &= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix} \\ \text{naive} \quad A \times B &= \begin{pmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{pmatrix} \end{aligned}, \tag{4.1}$$

where the M_i are individual multiplications as previously defined. While the Strassen algorithm requires fewer multiplications (7 instead of 8) it requires more additions (18 instead of 4). This overhead of addition can be reduced by using the Winograd form [112] which reads

$$A \times B = \begin{pmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & N_3 + N_2 + (A_{1,1} + A_{1,2} - A_{2,1} - A_{2,2})B_{2,2} \\ N_3 + N_1 + (B_{1,2} + B_{2,1} - B_{1,1} - B_{2,2})A_{2,2} & N_1 + N_2 + N_3 \end{pmatrix} \tag{4.2}$$

where $N_1 = (A_{2,1} - A_{1,1})(B_{1,2} - B_{2,2})$, $N_2 = (A_{2,1} + A_{2,2})(B_{1,2} - B_{1,1})$ and $N_3 = A_{1,1}B_{1,1} + (A_{2,1} + A_{2,2} - A_{1,1})(B_{1,1} + B_{2,2} - B_{1,2})$. In this form, the Strassen algorithm requires 15 additions instead of 18 and still uses only 7 multiplications. Further iterations over the Strassen algorithm have been introduced. In short, these methods leverage the laser method [201] which is a refinement of the Schönhage's asymptotic sum inequality [184] in order to reduce the bound ω on the complexity. Schönhage's asymptotic sum inequality generalizes the runtime bounds to tensor multiplications

$$\sum_{i=1}^{L} \text{Vol}(\langle n, m, p \rangle)^{\omega/3} \leq \underline{\text{R}} \left(\bigoplus_{i=1}^{L} \langle n, m, p \rangle \right), \tag{4.3}$$

where $\text{Vol}(\langle n, m, p \rangle) = nmp$ with $\langle n, m, p \rangle$ a tensor (algebraic tensor) which encodes the multiplication of a $n \times m$ matrix by a $m \times p$ one, and $\underline{\text{R}}$, the border rank, is the smallest r such that there is a sequence of tensors of rank at most r converging to tensor T. An example of the application of this inequality is the case $\langle 4, 1, 4 \rangle \oplus \langle 1, 9, 1 \rangle$ which gives:

$$16^{\omega/3} + 9^{\omega/3} \leq 17. \tag{4.4}$$

From this, Schönhage derived the bound $\omega < 2.55$. This gives the existence of a matrix multiplication algorithm with a runtime $O(n^{2.55})$. The most known matrix multiplication algorithm derived from the laser method is the Coppersmith-Winograd algorithm [10] which has been the most theoretically efficient one for over 20 years, with a complexity of $O(n^{2.38})$. It has been proven [9] that such approach cannot get faster than $O(n^{2.3725})$ with the current state-of-the-art [8] begin $O(n^{2.37286})$. However, these algorithms are referred to as galactic algorithms [121], meaning that the size required for them to reach their asymptotic behavior is too high to be worth implementing [136].

Stemming on these efficient algorithms, we propose to implement them such that previous importance-based pruning criteria can be leveraged in order to derive new inference algorithms. For example, let's consider the Winograd form of the Strassen algorithm: in such a case, we can either try to further reduce the number of multiplications or the number of additions by drawing inspiration from the previously introduced pruning techniques.

4.2.1 Multiplications Removal

The core idea consists in measuring the magnitude of an importance indicator (e.g. the gradients) with respect to a component of the matrix multiplication algorithm. Formally, let's consider two distinct layers f_1 and f_2 with weights and inputs (W_1, X_1) and (W_2, X_2) respectively. Then, their respective inference

under the Winograd algorithm would read

$$\begin{aligned} W_1 \times X_1 &= \begin{pmatrix} W_{11,1}X_{11,1} + W_{11,2}X_{12,1} & N_{13} + N_{12} + (W_{11,1} + W_{11,2} - W_{12,1} - W_{12,2})X_{12,2} \\ N_{13} + N_{11} + (X_{11,2} + X_{12,1} - X_{11,1} - X_{12,2})W_{12,2} & N_{11} + N_{12} + N_{13} \end{pmatrix} \\ W_2 \times X_2 &= \begin{pmatrix} W_{21,1}X_{21,1} + W_{21,2}X_{22,1} & N_{23} + N_{22} + (W_{21,1} + W_{21,2} - W_{22,1} - W_{22,2})X_{22,2} \\ N_{23} + N_{21} + (X_{21,2} + X_{22,1} - X_{21,1} - X_{22,2})W_{22,2} & N_{21} + N_{22} + N_{23} \end{pmatrix} \end{aligned} . \tag{4.5}$$

In order to use one formulation for both pruning during training and pruning post-training, let's introduce a set of variables $(\lambda_k)_{k\in\{1,\dots,7\}} \in [0;1]$ and transformations $(t_k)_{k\in\{1,\dots,7\}}$ such that we infer using

$$\begin{cases} \begin{pmatrix} \mathbf{t_1}(W_{i1,1})X_{i1,1} + \mathbf{t_2}(W_{i1,2})X_{i2,1} & N_{i3} + N_{i2} + \mathbf{t_3}(W_{i1,1} + W_{i1,2} - W_{i2,1} - W_{i2,2})X_{i2,2} \\ N_{i3} + N_{i1} + (X_{i1,2} + X_{i2,1} - X_{i1,1} - X_{i2,2})\mathbf{t_4}(W_{i2,2}) & N_{i1} + N_{i2} + N_{i3} \end{pmatrix} \\ \\ N_{i1} = \mathbf{t_5}(W_{i2,1} - W_{i1,1})(X_{i1,2} - X_{i2,2}) \\ N_{i2} = \mathbf{t_6}(W_{i2,1} + W_{i2,2})(X_{i1,2} - X_{i1,1}) \\ N_{i3} = W_{i1,1}X_{i1,1} + \mathbf{t_7}(W_{i2,1} + W_{i2,2} - W_{i1,1})(X_{i1,1} + X_{i2,2} - X_{i1,2}) \\ \\ t_k(W) = \lambda_k W + (1 - \lambda_k) \end{cases} . \tag{4.6}$$

Intuitively, the idea is that for any λ_k equal to 0 means that we do not use the weight values and thus do not perform the multiplication. On the other hand, for λ_k equal to 1, we keep the original inference computations. Thus, we solved two problems at once. First, with this formulation, we can optimize the inference algorithm during training using learnable variables $(\lambda_k)_{k\in\{1,\dots,7\}}$. Second, the variables are shared across layers by design, *i.e.* there are only 7 extra scalar variables for the entire network. This enables the use of a single inference algorithm, which reduces the implementation complexity. Further study could be conducted on the cost of using different inference algorithms for different part of the network (different computational blocks) or even different stages, like in the case of diffusion models. A similar process can be implemented for the additions.

4.2.2 Additions Removal

In the case of additions, we have to be wary of the fact that, to avoid memory overhead, weight additions are not computed in advance. Consequently, one cannot overlook terms such as the N_{i1} in equation 4.5. As a result, Consequently, we will introduce 32 extra variables $(\gamma_k)_{k\in\{1,\dots,32\}} \in [0;1]$ such that

$$\begin{cases} \begin{pmatrix} \boldsymbol{\gamma_1}W_{i1,1}X_{i1,1} + \boldsymbol{\gamma_2}W_{i1,2}X_{i2,1} & \boldsymbol{\gamma_3}N_{i3} + \boldsymbol{\gamma_4}N_{i2} + (\boldsymbol{\gamma_5}W_{i1,1} + \boldsymbol{\gamma_6}W_{i1,2} - \boldsymbol{\gamma_7}W_{i2,1} - \boldsymbol{\gamma_8}W_{i2,2})X_{i2,2} \\ \boldsymbol{\gamma_9}N_{i3} + \boldsymbol{\gamma_{10}}N_{i1} + (\boldsymbol{\gamma_{11}}X_{i1,2} + \boldsymbol{\gamma_{12}}X_{i2,1} - \boldsymbol{\gamma_{13}}X_{i1,1} - \boldsymbol{\gamma_{14}}X_{i2,2})W_{i2,2} & \boldsymbol{\gamma_{15}}N_{i1} + \boldsymbol{\gamma_{16}}N_{i2} + \boldsymbol{\gamma_{17}}N_{i3} \end{pmatrix} \\ \\ N_{i1} = (\boldsymbol{\gamma_{18}}W_{i2,1} - \boldsymbol{\gamma_{19}}W_{i1,1})(\boldsymbol{\gamma_{20}}X_{i1,2} - \boldsymbol{\gamma_{21}}X_{i2,2}) \\ N_{i2} = (\boldsymbol{\gamma_{22}}W_{i2,1} + \boldsymbol{\gamma_{23}}W_{i2,2})(\boldsymbol{\gamma_{24}}X_{i1,2} - \boldsymbol{\gamma_{25}}X_{i1,1}) \\ N_{i3} = \boldsymbol{\gamma_{26}}W_{i1,1}X_{i1,1} + (\boldsymbol{\gamma_{27}}W_{i2,1} + \boldsymbol{\gamma_{28}}W_{i2,2} - \boldsymbol{\gamma_{29}}W_{i1,1})(\boldsymbol{\gamma_{30}}X_{i1,1} + \boldsymbol{\gamma_{31}}X_{i2,2} - \boldsymbol{\gamma_{32}}X_{i1,2}) \end{cases} . \tag{4.7}$$

As a result, this formulation can learn to remove additions. For example, having γ_{18} and γ_{19} both set to zero, would remove the need for computing N_{i1}. On the other hand, in the previous implementation, having $\lambda_5 = 0$ would lead to removing the multiplication in N_{i1} by only computing $X_{i1,2} - X_{i2,2}$. Consequently, we think it is worth considering the option of adding extra variables to the addition removal formulation to enable a similar behavior as in the multiplication formulation.

While we believe this to be a very relevant path forward with respect to deep neural network pruning, we observed that quantization generally outperforms pruning, with binary quantization being the peak compression rate that has been achieved. However, we still struggle to efficiently quantize autoregressive models. In the following section, we propose a path towards solving this issue.

4.3 GPTQ and Auto-Regressive Models

As discussed in the previous chapter, gradient-based post-training quantization performs a weight optimization, block per block, sequentially from the first layer to the last layer. Intuitively, as the first layers are optimized, the current layer is processed with input features that correspond to the behavior that would actually be produced at inference. In other words, the current GPTQ methods rely on the following assumptions: the deep neural network to quantize F is a directed acyclic graph [125] (DAGs).

For example, all the standard convolutional neural networks and transformers for image classification and object detection are DAGs. However, most of the trending generative AI models do not behave like DAGs at inference. For instance, a diffusion model will iterate several times over the same input, progressively denoising it. Similarly, large language models are decoding tokens one by one and thus cycle multiple times over the same input as they complete a sentence. As a result, these models infringe the basic assumptions of GPTQ methods.

In practice, this does not prevent GPTQ methods from outperforming most data-free techniques, which implies that this does not lead to a catastrophic performance degradation. Still, we think that significant improvements could be achieved for GPTQ methods on generative AI. In order to design a non-DAG friendly GPTQ method, let's consider the use-case of a diffusion model F which takes as inputs a noisy image I_t and a time-step t. The current approach to perform a GPTQ method such as AdaRound would be to randomly sample time-steps for the current layer inputs. However, as diffusion models are not DAGs, the optimized weight values would be optimized for inputs that will be modified by the next layers optimization. In order to alleviate this problem, we would propose to optimize one variable ϵ_t per time-step independently. Formally, for a unique, fixed time-step, a diffusion model is indeed a DAG, which implies that GPTQ methods can be performed straightforwardly. This would result in a set of $(\epsilon_t)_{t\in[\![1;T]\!]}$. A first solution would be to simply change these binary values given the current time-step. However, this approach would introduce an inference overhead. To avoid this shortcoming, an interesting solution would be to use the $(\epsilon_t)_{t\in[\![1;T]\!]}$ and average them over the time-steps in order to get a new initialization before performing the final pass of the GPTQ method using all time-steps. Intuitively, we use each time-step to vote for a starting point.

While this subject tackles an explicit problem of current quantization methods on trending neural architectures, it does not address all challenges. In particular, the aforementioned outliers are currently handled with group-wise quantization for large language models. In the following section, we will detail the pitfall of this approach and suggestions for future workarounds.

4.4 Working past Group-wise Quantization

As discussed in Chapter 3, large language models and more generally, generative AI models learn weight distributions which include outliers. Formally, an outlier is defined as any scalar value within a tensor that is at least 6 standard deviations away from the mean. In the specific context of quantization, the presence of outliers stretches out the support of the tensor to quantize, which leads to lower values being quantized to zero. The phenomenon is illustrated in Figure 4.1 in the context of uniform quantization using 5 values in the quantized space. In particular, in the provided example, we can see that the almost all values are quantized over two bins: zero and the outlier. In other words, while our representation format can encode 5 distinct values only 2 are leveraged which is a waste and often leads to a catastrophic performance drop. In order to alleviate this limitation, many solutions have introduced, including some discussed in the previous sections, such as PowerQuant [255] and REx [246]. However, one appears to me as a future limitation: group-wise quantization [167].

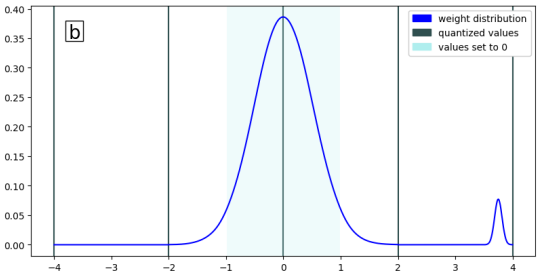


Figure 4.1: Outliers stretch the distribution out and lead to zero quantization of most of the weight values.

Group-wise quantization consists in a new granularity for deep quantization. This matter has been detailed in section 3.1.2 where we introduced the dimensionality constraint before addressing it with SPIQ [256]. In short, the dimensionality constraint states that only the weight values or the inputs can be quantized per-channel at once in order to preserve the inference speed-ups. In the case of group-wise quantization, the weights are quantized in a finer-grain than per-channel:

$$\begin{cases} \text{tensor-wise} & (WX)_j = \lambda \sum_i W_{i,j} X_i \\ \text{channel-wise} & (WX)_j = \lambda_j \sum_i W_{i,j} X_i \\ \text{group-wise} & (WX)_j = \sum_i \lambda_{j,k_i} W_{i,j} X_i \end{cases} \tag{4.8}$$

where the k_i cluster subsequent input features (in practice groups of size 128 are the standard) W are quantized weights and inputs X are not. The reason why inputs are not quantized is the fact that the

summation can not be performed at once as we have a different scaling term for different terms of the sum, *i.e.* scalings must be performed before reduction. As a result, every method that leverages group-wise quantization is bound to be limited to the compression of the memory footprint of the weights and not their latency. While memory footprint is a significant challenge on its own, latency should not be overlooked.

In order to address this issue, we would investigate the possibility to carry both weight and input outliers in binary sparse tensors, similarly as in REx. In a nutshell, in REX for LLMs, we implemented a binary sparse residue to specifically encode weight tensors. From these weight tensors, outlier features may arise, and we propose to catch these specific candidate outliers and also store them in residues in order to leverage efficient residual expansions of the features rather than weight only group-wise quantization. This may unlock higher compression rates. However, it would not solve the general major limitation of quantization: the inability of the compression community to tackle proper modular arithmetic. In the following section, we share our insights regarding this aspect of deep quantization.

4.5 Modular Arithmetic

To the best of our knowledge, the first and only significant contribution to the goal of proper modular arithmetic in deep quantization is WrapNet [161]. Formally, given a layer f with weight tensor W and input features X, its goal is to achieve a satisfactory accuracy for the whole network F quantized in $\mathfrak{b}$ bits with

$$f : X \mapsto \text{mod}_{2^{\mathfrak{b}}} \left[(W \times X) + (2^{\mathfrak{b}-1} - 1)\right] - (2^{\mathfrak{b}-1} - 1) \tag{4.9}$$

assuming that the weights and inputs are already quantized. The core challenge at hand is the bit-width $\mathfrak{b}$. In WrapNet, the authors do not actually achieve modular arithmetic as intended here: the use of a specific quantization bit-width for the weights and activations and another bit-width for the accumulation (corresponding to the modulo of the last equation). Ideally, one would like to use one bit-width $\mathfrak{b}$ which would lead to simpler arithmetic from a hardware perspective and no need for multiple format support. However, in practice, the reason why this was not done in WrapNet was to limit the occurrences of overflows, *i.e.* the proposed method can support few overflows but not systematic overflows which would occur if we had one unique bit-width. Formally, using the previous notations for quantization, a binary neural network would be noted W1/A1. If we add the accumulator bit-width, it would read W1/A1/A1. The WrapNet method supports W2/A2/A8. However, our goal would be W4/A4/A4 or any W$\mathfrak{b}$/A$\mathfrak{b}$/A$\mathfrak{b}$.

During, this thesis, we reproduced this method and tested with the desired W$\mathfrak{b}$/A$\mathfrak{b}$/A$\mathfrak{b}$ to no result. Our insight is the following: the difficulty arises from the fact that the work space is no longer ordered[1]. Intuitively, all current training techniques, that we are aware of, leverage an ordered space. For instance, classification requires a definition of the likeliest class, *i.e.* we need to rank logits, and thus we need an order over the logit space. However, modular spaces are not ordered.

A naive and straightforward solution to this issue would be to simply use a clipping operation rather than the modulo in order to always remain on the same format. However, this is even worse, as the output of a layer would depend on the order of the inputs. Formally, let's consider the following quantized scalar product on 4 bits (quantized space is $[\![-8; 7]\!]$): $4 \times 4 - 1 \times 2 + 1 \times 1 = 6 \neq 7 = -1 \times 2 + 1 \times 1 + 4 \times 4$ in the case of clipping. Furthermore, this loss of the invariance to neuron order breaks the symmetry, which is a core property that is often considered to be central to the good performance of deep neural network.

In order to train a modular quantized model with a bit-width $\mathfrak{b}$, We would introduce a pseudo ordering and investigate the adoption of the spiking mechanism to the optimization process. Regarding, the pseudo ordering, We would use the distance to $2^{\mathfrak{b}-1} - 0.5$ (the edge of the support). The modular space can be seen as a discreet circle over which we rotate. The ranking is thus defined as the distance to a reference point selected such that by default each prediction are even spread around the reference point. Intuitively, this is similar to the standard initialization of any classification neural network, as the initialization method leads to an average prediction around the center of the support. Second, regarding the optimization, let's consider a shallow network F such that

$$F : X \mapsto W_2 \text{ReLU}(W_1 \times X) \tag{4.10}$$

For the sake of simplicity, we will assume that $X \in \mathbb{R}^2$ and $F(X) \in \mathbb{R}$ for a binary classification. We propose to run all the derivations of the back-propagation of F, for a label $Y^* = (0, 1)$. By definition, the

[1] We need to add a reference to the order matters paper [207] from a colleague, although it is completely unrelated, as a joke

output $Y = F(X)$ lies in $[\![-8; 7]\!]$. In order to have the maximum value at 7.5 (or -7.5), a naive approach would be to have the loss

$$\mathcal{L} = L\left(\frac{|Y + 0.5|}{7.5}, Y^*\right), \tag{4.11}$$

where L is any regular loss function such as the l_1 or the cross-entropy [74]. As a result, this loss is computed in $\mathbb{R}$ (using high precision floating point values), this is not an issue as our goal here is not to tackle efficient training with low bit optimization [82, 178, 235]. Assuming, a binary cross entropy as our loss, the gradient term becomes

$$\frac{\partial \mathcal{L}}{\partial \hat{Y}} = \frac{\partial - Y^* \log(\hat{Y}) - (1 - Y^*) \log(1 - \hat{Y})}{\partial \hat{Y}} = \frac{-Y^*}{\hat{Y}} - \frac{1 - Y^*}{1 - \hat{Y}}, \tag{4.12}$$

with $\hat{Y} = \frac{|Y+0.5|}{7.5}$. We derive the gradient term with respect to the output Y,

$$\frac{\partial \mathcal{L}}{\partial Y} = -\text{sign}(Y + 0.5) \times \left(\frac{Y^*}{|Y + 0.5|} + \frac{1 - Y^*}{7.5 - |Y + 0.5|}\right). \tag{4.13}$$

Then the standard gradient update of W_2 is proportional to $\text{sign}(Y+0.5)\times\left(\frac{Y^*}{|Y+0.5|} + \frac{1-Y^*}{7.5-|Y+0.5|}\right) \text{ReLU}(W_1 \times X)$. However, the dynamic of the input features $\text{ReLU}(W_1 \times X)$ may drastically change after the update of W_1 because of a possible overflow. In order to alleviate this shortcoming, we would investigate the spike mechanism for the optimizer. Our inspiration is as follows: spike neural networks propagate the information if and only if enough energy is given. In the context of modulo quantized networks optimization, we would define the energy as inversely proportional to the quotient from the modulo operation. Intuitively, if a feature was not derived from overflows, its optimization is straightforward (high energy) and to the contrary, if a feature was obtained from numerous overflows (low energy) its optimization is sensitive. Multiple questions remain open: should we carry the energy over the layer? while the energy is defined, how should we derive the actual weight update?

Regarding the first question, the intuition is that carrying the energy from earlier layers in the forward pass would decrease the energy of the last layers and thus put more emphasis on the optimization of the first layers. This property is the opposite of the current optimizer methods, which tend to over-train the last layers (vanishing gradients [19]). Regarding the second question, we would perform an integer update in order to keep weight values as integers throughout the process and reduce the memory footprint. Consequently, the update would be the sign function of the accumulated average gradients at the moment of the spike in the optimizer.

Our intuition is that the modulo quantized model training will have difficulty disentangling the intermediate features. The implementation of regularization terms might be necessary. This research topic is exploratory and significantly differs from the current literature. A second subject that is not well studied is efficient training, which we propose to discuss in the following section.

4.6 Efficient Training with Adapters

To the best of our knowledge, the only work on efficient training is Net2Net [32]. In order to reduce the training duration and cost, the authors propose to use a narrower initial network that they progressively grow during the training process. Consequently, the first epochs are less costly in terms of memory and duration as compared to a standard training of the fully grown model.

Efficient training was initial a part of this PhD thesis, but we did not have the time to contribute to it. Still, we propose a lead towards lower training costs. To do so, we draw inspiration from adapters [175]: set the model architecture and train it from scratch only using adapters. The great benefit of adapters for fine-tuning is their relative low memory footprint as compared to the full model. However, it is unlikely that the same performance could be reached using a small adapter as compared to the full network. Consequently, we suggest drawing inspiration from Net2Net and use different adapters sizes as the training goes. As a result, the LoRa [95] method is best suited for this task as it can be folded, which means that there is no overhead from changing the adapter size during training.

All these leads may be of significant impact in the future, and we hope they spark enough interest in the reader to investigate them.

Bibliography

[1] H. Abdi. Singular value decomposition (svd) and generalized singular value decomposition. *Encyclopedia of measurement and statistics*, 907:912, 2007.

[2] J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. Sanity checks for saliency maps. *NeurIPS*, 31, 2018.

[3] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[4] K. Aggarwal and H. K. Verma. Hash_rc6—variable length hash algorithm using rc6. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 450–456. IEEE, 2015.

[5] A. Ahmad, S. A. R. Kashif, M. A. Saqib, A. Ashraf, and U. T. Shami. Tariff for reactive energy consumption in household appliances. *Energy*, 186:115818, 2019.

[6] S. Ahmed, M. Nawaz, A. Bakar, N. A. Bhatti, M. H. Alizai, J. H. Siddiqui, and L. Mottola. Demystifying energy consumption dynamics in transiently powered computers. *ACM Transactions on Embedded Computing Systems (TECS)*, 19(6):1–25, 2020.

[7] M. M. Al-Kofahi, M. Y. Al-Shorman, and O. M. Al-Kofahi. Toward energy efficient microcontrollers and internet-of-things systems. *Computers & Electrical Engineering*, 79:106457, 2019.

[8] J. Alman and V. V. Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 522–539. SIAM, 2021.

[9] A. Ambainis, Y. Filmus, and F. Le Gall. Fast matrix multiplication: limitations of the coppersmith-winograd method. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 585–593, 2015.

[10] M. Anderson and S. Barman. The coppersmith-winograd matrix multiplication algorithm. Technical report, Tech. Rep., 2009.[Online]. Available: https://lists. cs. wisc. edu/archive . . . , 2009.

[11] A. S. Andrae. Projecting the chiaroscuro of the electricity use of communication and computing from 2018 to 2030. *Preprint*, pages 1–23, 2019.

[12] S. Anwar, K. Hwang, and W. Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[13] M. Astrid and S.-I. Lee. Cp-decomposition with tensor power method for convolutional neural networks compression. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 115–118. IEEE, 2017.

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[15] D. Bacciu and D. P. Mandic. Tensor decompositions in deep learning. *arXiv preprint arXiv:2002.11835*, 2020.

[16] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[17] K. Banerjee, R. R. Gupta, K. Vyas, B. Mishra, et al. Exploring alternatives to softmax function. *arXiv preprint arXiv:2011.11538*, 2020.

[18] R. Banner, Y. Nahshan, and D. Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems*, 32, 2019.

[19] S. Basodi, C. Ji, H. Zhang, and Y. Pan. Gradient amplification: An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3):196–207, 2020.

[20] S. Basu and A. DasGupta. The mean, median, and mode of unimodal distributions: a characterization. *Theory of Probability & Its Applications*, 41(2):210–223, 1997.

[21] D. Baymurzina, E. Golikov, and M. Burtsev. A review of neural architecture search. *Neurocomputing*, 474:82–93, 2022.

[22] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[23] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[24] Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *AAAI*, volume 34, pages 7432–7439, 2020.

[25] D. Blalock, J. J. Gonzalez Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[26] J. Bonnard, A. Dapogny, F. Dhombres, and K. Bailly. Privileged attribution constrained deep networks for facial expression recognition. In *2022 26th International Conference on Pattern Recognition (ICPR)*, pages 1055–1061. IEEE, 2022.

[27] T. Bouguera, J.-F. Diouris, J.-J. Chaillout, and G. Andrieux. Energy consumption modeling for communicating sensors using lora technology. In *2018 IEEE Conference on Antenna Measurements & Applications (CAMA)*, pages 1–4. IEEE, 2018.

[28] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer. Zeroq: A novel zero shot quantization framework. In *CVPR*, pages 13169–13178, 2020.

[29] H. Cao, C. Tan, Z. Gao, G. Chen, P.-A. Heng, and S. Z. Li. A survey on generative diffusion model. *arXiv preprint arXiv:2209.02646*, 2022.

[30] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *WACV*, pages 839–847. IEEE, 2018.

[31] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *ArXiv*, abs/1706.05587, 2017.

[32] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[33] T. Chen, B. Ji, T. Ding, B. Fang, G. Wang, Z. Zhu, L. Liang, Y. Shi, S. Yi, and X. Tu. Only train once: A one-shot neural network training and pruning framework. *NeurIPS*, 34, 2021.

[34] Y. Chen, A. Saporta, A. Dapogny, and M. Cord. Delving deep into interpreting neural nets with piece-wise affine representation. In *ICIP*, 2019.

[35] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6(3):264–274, 2020.

[36] Y. Cheng, D. Wang, et al. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[37] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1002–1012, 2010.

[38] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[39] A. Choromanska, Y. LeCun, and G. B. Arous. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pages 1756–1760. PMLR, 2015.

[40] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019.

[41] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[42] T. community. Mlperf tiny inference benchmark. *TinyML*, 2021.

[43] G. Cong et al. Squant: On-the-fly data-free quantization via diagonal hessian approximation. *ICLR*, 2022.

[44] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3213–3223, 2016.

[45] M. Courbariaux, I. Hubara, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *NeurIPS*, 2016.

[46] Data-Bricks. Databricks' dolly-v2, an instruction-following large language model trained on the databricks machine learning platform. huggingface reference (`https://huggingface.co/databricks`), 2023.

[47] J. Deng, W. Dong, et al. ImageNet: A Large-Scale Hierarchical Image Database. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2009.

[48] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[49] L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[50] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer. Llm. int8 (): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

[51] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[52] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[53] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[54] T. DeVries and G. W. Taylor. Learning confidence for out-of-distribution detection in neural networks. *stat*, 1050:13, 2018.

[55] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[56] S. Elfwing, E. Uchibe, and K. Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks : the official journal of the International Neural Network Society*, 107:3–11, 2017.

[57] A. C. Elster and T. A. Haugdahl. Nvidia hopper gpu and grace cpu highlights. *Computing in Science & Engineering*, 24(2):95–100, 2022.

[58] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys (CSUR)*, 24(4):441–476, 1992.

[59] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.

[60] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[61] A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320*, 2020.

[62] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[63] B. Federico and P. Jeff. Exploiting nvidia ampere structured sparsity with cusparselt. https://developer.nvidia.com/blog/exploiting-ampere-structured-sparsity-with-cusparselt/, 2020.

[64] Z. Feng, J. Lai, and X. Xie. Resolution-aware knowledge distillation for efficient inference. *IEEE Transactions on Image Processing*, 30:6985–6996, 2021.

[65] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

[66] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. Optq: Accurate quantization for generative pre-trained transformers. In *ICLR*, 2023.

[67] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

[68] L. Gao, J. Tow, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, K. McDonell, N. Muennighoff, J. Phang, L. Reynolds, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. A framework for few-shot language model evaluation, Sept. 2021.

[69] A. Géron. Hands-on machine learning with scikit-learn, keras & tensorflow farnham. *Canada: O'Reilly*, 2019.

[70] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

[71] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[72] A. Gomez, C. Pinto, A. Bartolini, D. Rossi, L. Benini, H. Fatemi, and J. P. de Gyvez. Reducing energy consumption in microcontroller-based platforms with low design margin co-processors. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 269–272. IEEE, 2015.

[73] R. Gong et al. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *ICCV*, pages 4852–4861, 2019.

[74] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 14(1):107–114, 1952.

[75] Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[76] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[77] A. Gudibande, E. Wallace, C. Snell, X. Geng, H. Liu, P. Abbeel, S. Levine, and D. Song. The false promise of imitating proprietary llms. *arXiv preprint arXiv:2305.15717*, 2023.

[78] J. Guo, W. Ouyang, and D. Xu. Multi-dimensional pruning: A unified framework for model compression. *CVPR*, pages 1508–1517, 2020.

[79] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang. Fbna: A fully binarized neural network accelerator. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 51–513. IEEE, 2018.

[80] Y. Guo, H. Yuan, J. Tan, Z. Wang, S. Yang, and J. Liu. Gdp: Stabilized neural network pruning via gates with differentiable polarization. *ICCV*, pages 5239–5250, 2021.

[81] O. Hamdi and H. Imen. Performance evaluation for dense sparse matrix product algorithms. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 33–40. IEEE, 2017.

[82] Q. Han, Y. Hu, F. Yu, H. Yang, B. Liu, P. Hu, R. Gong, Y. Wang, R. Wang, Z. Luan, et al. Extremely low-bit convolution optimization for quantized neural network on modern computer architectures. In *Proceedings of the 49th International Conference on Parallel Processing*, pages 1–12, 2020.

[83] B. Hanin and M. Sellke. Approximating continuous functions by relu nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[84] E. B. Hansen and S. Bøgh. Artificial intelligence and internet of things in small and medium-sized enterprises: A survey. *Journal of Manufacturing Systems*, 58:362–372, 2021.

[85] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[86] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[87] D. Hendrycks, K. Zhao, S. Basart, J. Steinhardt, and D. Song. Natural adversarial examples. In *CVPR*, pages 15262–15271, 2021.

[88] H. Herrlich. *Axiom of choice*, volume 1876. Springer, 2006.

[89] J. Hessel, A. Holtzman, M. Forbes, R. L. Bras, and Y. Choi. Clipscore: A reference-free evaluation metric for image captioning. *arXiv preprint arXiv:2104.08718*, 2021.

[90] D. L. N. Hettiarachchi, V. S. P. Davuluru, and E. J. Balster. Integer vs. floating-point processing on modern fpga technology. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pages 0606–0612. IEEE, 2020.

[91] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *Advances in neural information processing systems*, 2014.

[92] C. Holmes, M. Zhang, Y. He, and B. Wu. Nxmtransformer: Semi-structured sparsification for natural language understanding via admm. *Advances in neural information processing systems*, 34:1818–1830, 2021.

[93] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[94] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.

[95] E. J. Hu, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2021.

[96] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[97] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[98] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

[99] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[100] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[101] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[102] M.-E. Ionascu and M. Marcu. Energy profiling for different bluetooth low energy designs. In *2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 2, pages 1032–1036. IEEE, 2017.

[103] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[104] Y. Jeon, B. Park, S. J. Kwon, B. Kim, J. Yun, and D. Lee. Biqgemm: matrix multiplication with lookup table for binary-coding-based quantized dnns. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, 2020.

[105] C. Jin, X. Bai, C. Yang, W. Mao, and X. Xu. A review of power consumption models of servers in data centers. *applied energy*, 265:114806, 2020.

[106] A. Kapishnikov, S. Venugopalan, B. Avci, B. Wedin, M. Terry, and T. Bolukbasi. Guided integrated gradients: An adaptive path method for removing noise. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5050–5058, 2021.

[107] R. M. Kaplan. A method for tokenizing text. *Inquiries into words, constraints and contexts*, 55, 2005.

[108] P. Kidger and T. Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

[109] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer. I-bert: Integer-only bert quantization. In *International conference on machine learning*, pages 5506–5518. PMLR, 2021.

[110] W. Kim, S. Kim, M. Park, and G. Jeon. Neuron merging: Compensating for pruned neurons. *Advances in Neural Information Processing Systems*, 33:585–595, 2020.

[111] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[112] D. Knuth. The art of computer programming, 2 (seminumerical algorithms). *(No Title)*, 1981.

[113] R. Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[114] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images, 2009.

[115] S. Kullback and R. A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[116] M. Kurtz, J. Kopinsky, R. Gelashvili, A. Matveev, J. Carr, M. Goin, W. Leiserson, S. Moore, B. Nell, N. Shavit, and D. Alistarh. Inducing and exploiting activation sparsity for fast inference on deep neural networks. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5533–5543, Virtual, 13–18 Jul 2020. PMLR.

[117] A. Kusupati, V. Ramanujan, R. Somani, M. Wortsman, P. Jain, S. Kakade, and A. Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *ICML*, pages 5544–5555. PMLR, 2020.

[118] A. Kuzmin, M. Van Baalen, Y. Ren, M. Nagel, J. Peters, and T. Blankevoort. Fp8 quantization: The power of the exponent. *Advances in Neural Information Processing Systems*, 35:14651–14662, 2022.

[119] D. H. Le and B.-S. Hua. Network pruning that matters: A case study on retraining variants. In *International Conference on Learning Representations*, 2020.

[120] Q. V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

[121] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *2012 IEEE 53rd annual symposium on foundations of computer science*, pages 514–523. IEEE, 2012.

[122] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[123] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[124] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

[125] C.-Y. Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

[126] J. Lee, S. Park, S. Mo, S. Ahn, and J. Shin. Layer-adaptive sparsity for the magnitude-based pruning. In *International Conference on Learning Representations*, 2020.

[127] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. In *International Conference on Learning Representations*, 2020.

[128] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *NeurIPS*, 2021.

[129] Y. Li, F. Zhu, R. Gong, M. Shen, X. Dong, F. Yu, S. Lu, and S. Gu. Mixmix: All you need for data-free compression are feature and data mixing. In *ICCV*, pages 4410–4419, 2021.

[130] Z. Li, L. Ma, M. Chen, J. Xiao, and Q. Gu. Patch similarity aware data-free quantization for vision transformers. *arXiv preprint arXiv:2203.02250*, 2022.

[131] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[132] H. Lin and S. Jegelka. Resnet with one-neuron hidden layers is a universal approximator. *Advances in neural information processing systems*, 31, 2018.

[133] M. Lin, R. Ji, et al. Hrank: Filter pruning using high-rank feature map. *CVPR*, pages 1529–1538, 2020.

[134] T. Lin et al. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[135] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice, and D. Gruss. Nethammer: Inducing rowhammer faults through network requests. In *EuroS&PW*, pages 710–719. IEEE, 2020.

[136] R. J. Lipton, K. W. Regan, R. J. Lipton, and K. W. Regan. David johnson: Galactic algorithms. *People, Problems, and Proofs: Essays from Gödel's Lost Letter: 2010*, pages 109–112, 2013.

[137] J. Liss. A software technique for diagnosing and correcting memory errors. *IEEE transactions on reliability*, 35(1):12–18, 1986.

[138] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[139] J. Liu, L. Niu, Z. Yuan, D. Yang, X. Wang, and W. Liu. Pd-quant: Post-training quantization based on prediction difference metric. In *CVPR*, pages 24427–24437, 2023.

[140] J. Liu, B. Zhuang, Z. Zhuang, Y. Guo, J. Huang, J. Zhu, and M. Tan. Discrimination-aware network pruning for deep model compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[141] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J.-H. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, and W. Zhang. Group fisher pruning for practical network compression. In *ICML*, pages 7021–7032. PMLR, 2021.

[142] Z. Liu, Z. Shen, M. Savvides, and K.-T. Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 143–159. Springer, 2020.

[143] W. Ma and J. Lu. An equivalence of fully connected layer and convolutional layer. *arXiv preprint arXiv:1712.01252*, 2017.

[144] B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *arXiv preprint arXiv:2003.11535*, 2020.

[145] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey. Recalibrating global data center energy-use estimates. *Science*, 367(6481):984–986, 2020.

[146] A. N. Mazumder, J. Meng, H.-A. Rashid, U. Kallakuri, X. Zhang, J.-S. Seo, and T. Mohsenin. A survey on the optimization of neural network accelerators for micro-ai on-device inference. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):532–547, 2021.

[147] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Image-based localization using hourglass networks. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 879–886, 2017.

[148] A. Menon, K. Mehrotra, C. K. Mohan, and S. Ranka. Characterization of a class of sigmoid functions with applications to neural networks. *Neural networks*, 9(5):819–835, 1996.

[149] Microsoft. Deepspeed. *GitHub*, 2023.

[150] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

[151] M. P. Mills. The cloud begins with coal. *Digital Power Group*, 1, 2013.

[152] A. Mishra, J. A. Latorre, J. Pool, D. Stosic, D. Stosic, G. Venkatesh, C. Yu, and P. Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[153] D. Miyashita, E. H. Lee, and B. Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[154] 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.

[155] R. T. Mullapudi, S. Chen, K. Zhang, D. Ramanan, and K. Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 3573–3582, 2019.

[156] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

[157] M. Nagel, M. v. Baalen, T. Blankevoort, and M. Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1325–1334, 2019.

[158] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[159] A. Nguyen, K. Pham, D. Ngo, T. Ngo, and L. Pham. An analysis of state-of-the-art activation functions for supervised deep neural network. In *2021 International Conference on System Science and Engineering (ICSSE)*, pages 215–220. IEEE, 2021.

[160] Q.-H. Nguyen and F. Dressler. A smartphone perspective on computation offloading—a survey. *Computer Communications*, 159:133–154, 2020.

[161] R. Ni, H.-m. Chu, O. Castaneda, P.-y. Chiang, C. Studer, and T. Goldstein. Wrapnet: Neural net inference with ultra-low-precision arithmetic. In *International Conference on Learning Representations*, 2020.

[162] M. Nonnenmacher, T. Pfeil, I. Steinwart, and D. Reeb. Sosp: Efficiently capturing global correlations by second-order structured pruning. *ICLR*, 2021.

[163] P. Novello, T. Fel, and D. Vigouroux. Making sense of dependence: Efficient black-box explanations using dependence measure. *NeurIPS*, 2022.

[164] Nvidia. Nvidia a100 tensor core gpu architecture. web tech report (`https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf`), 2021.

[165] C. C. Paige and M. A. Saunders. Towards a generalized singular value decomposition. *SIAM Journal on Numerical Analysis*, 18(3):398–405, 1981.

[166] K. PARESH. Tensorfloat-32 in the a100 gpu accelerates ai training, hpc up to 20x. https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/, 2020.

[167] G. Park, B. Park, S. J. Kwon, B. Kim, Y. Lee, and D. Lee. nuqmm: Quantized matmul for efficient inference of large-scale generative language models. *arXiv preprint arXiv:2206.09557*, 2022.

[168] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[169] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[170] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola. Federated deep learning for zero-day botnet attack detection in iot-edge devices. *IEEE Internet of Things Journal*, 9(5):3930–3944, 2021.

[171] K. H. Randall. *Cilk: Efficient multithreaded computing*. PhD thesis, Massachusetts Institute of Technology, 1998.

[172] S. Randhawa and L. Chopra. Pestilential impacts of battery industry discharged metal waste on human health. *Materials Today: Proceedings*, 52:434–438, 2022.

[173] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pages 525–542. Springer, 2016.

[174] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *International conference on machine learning*, pages 2902–2911. PMLR, 2017.

[175] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8119–8127, 2018.

[176] I. Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie: Workshop, Hannover, 29. Sept.–1. Okt. 1977*, pages 83–114. Springer, 1978.

[177] L. Renjie, Z. Xunkai, L. Tian, L. Yuqi, T. Mingxing, L. Khanh, M. Chao, J. Amy, M. Luiz, GUStavo, L. Yunlu, S. Suharsh, A. Raziel, C. Lawrence, K. Jess, L. Mike, L. Shuangfeng, and S. Sarah. Higher accuracy on vision models with efficientnet-lite, 2020.

[178] A. Rodriguez, E. Segal, E. Meiri, E. Fomenko, Y. J. Kim, H. Shen, and B. Ziv. Lower numerical precision deep learning inference and training. *Intel White Paper*, 3(1):19, 2018.

[179] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, June 2022.

[180] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[181] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[182] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[183] M. Sandler, A. Howard, et al. Mobilenetv2: Inverted residuals and linear bottlenecks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[184] A. Schönhage. Partial and total matrix multiplication. *SIAM Journal on Computing*, 10(3):434–455, 1981.

[185] T. J. Scott. Mathematics and computer science at odds over real numbers. *ACM SIGCSE Bulletin*, 23(1):130–139, 1991.

[186] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? *arXiv preprint arXiv:1611.07450*, 2016.

[187] M. Shen, H. Yin, P. Molchanov, L. Mao, J. Liu, and J. M. Alvarez. Halp: hardware-aware latency pruning. *arXiv preprint arXiv:2110.10811*, 2021.

[188] S. Shi, Q. Wang, and X. Chu. Efficient sparse-dense matrix-matrix multiplication on gpus using the customized sparse storage format. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 19–26. IEEE, 2020.

[189] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[190] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[191] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[192] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain. Machine translation using deep learning: An overview. In *2017 international conference on computer, communications and electronics (comptelix)*, pages 162–167. IEEE, 2017.

[193] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[194] Z. Song, X. Zhang, and C. Eriksson. Data center energy and cost saving evaluation. *Energy Procedia*, 75:1255–1260, 2015.

[195] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *BMVC*, 2015.

[196] A. Srivastava, S. Jain, and M. Thigle. Out of distribution detection on imagenet-o. *arXiv preprint arXiv:2201.09352*, 2022.

[197] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[198] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang. Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 766–780. IEEE, 2020.

[199] K. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8:131–162, 2007.

[200] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[201] V. Strassen. Relative bilinear complexity and matrix multiplication. *Journal für die reine und angewandte Mathematik*, 1987.

[202] V. Strassen et al. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[203] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[204] W. Sun, A. Zhou, S. Stuijk, R. Wijnhoven, A. O. Nelson, H. Corporaal, et al. Dominosearch: Find layer-wise fine-grained n: M sparse schemes from dense neural networks. *NeurIPS*, 34, 2021.

[205] M. Sundararajan, A. Taly, and Q. Yan. Axiomatic attribution for deep networks. *ICML*, pages 3319–3328, 2017.

[206] K. Suzuki. Overview of deep learning in medical imaging. *Radiological physics and technology*, 10(3):257–273, 2017.

[207] G. Tallec, A. Dapogny, and K. Bailly. Multi-order networks for action unit detection. *IEEE Transactions on Affective Computing*, 2022.

[208] G. Tallec, E. Yvinec, A. Dapogny, and K. Bailly. Fighting over-fitting with quantization for learning deep neural networks on noisy labels. *arXiv preprint arXiv:2303.11803*, 2023.

[209] M. Tan and Q. Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[210] M. Tan and Q. V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International conference on machine learning*, pages 6105–6114, 2019.

[211] Y. Tang, Y. Wang, Y. Xu, Y. Deng, C. Xu, D. Tao, and C. Xu. Manifold regularized dynamic network pruning. *arXiv preprint arXiv:2103.05861*, 2021.

[212] P. Teich. "tearing apart google's tpu 3.0 ai coprocessor". the next platform. retrieved 2020-08-11. google invented its own internal floating point format called "bfloat" for "brain floating point" (after google brain). https://www.nextplatform.com/2018/05/10/tearing-apart-googles-tpu-3-0-ai-coprocessor/, 2018.

[213] J. Terven and D. Cordova-Esparza. A comprehensive review of yolo: From yolov1 to yolov8 and beyond. *arXiv preprint arXiv:2304.00501*, 2023.

[214] A. Tjandra, S. Sakti, and S. Nakamura. Tensor decomposition for compressing recurrent neural network. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[215] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34:24261–24272, 2021.

[216] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. *International conference on machine learning*, 2020.

[217] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image transformers. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

[218] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[219] Y. Tyler. Torchsummary. *GitHub*, 2023.

[220] J. Valente, S. Wu, A. Gelfand, and C. Sirmans. Apartment rent prediction using spatial modeling. *Journal of Real Estate Research*, 27(1):105–136, 2005.

[221] H. Vanholder. Efficient inference with tensorrt, 2016.

[222] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[223] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio. Manifold mixup: Better representations by interpolating hidden states. In *ICML*, pages 6438–6447. PMLR, 2019.

[224] S. Wan, L. Qi, X. Xu, C. Tong, and Z. Gu. Deep learning models for real-time human activity recognition with smartphones. *Mobile Networks and Applications*, 25:743–755, 2020.

[225] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. Association for Computational Linguistics, 2018.

[226] H. Wang, C. Qin, Y. Bai, Y. Zhang, and Y. Fu. Recent advances on neural network pruning at initialization. *arXiv preprint arXiv:2103.06460*, 2021.

[227] Z. Wang, C. Li, and X. Wang. Convolutional neural network pruning with structural redundancy reduction. *CVPR*, pages 14913–14922, 2021.

[228] X. Wei, R. Gong, Y. Li, X. Liu, and F. Yu. Qdrop: randomly dropping quantization for extremely low-bit post-training quantization. *ICLR*, 2022.

[229] R. Wightman, H. Touvron, and H. Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.

[230] J. H. Wilkinson, F. L. Bauer, and C. Reinsch. *Linear algebra*, volume 2. Springer, 2013.

[231] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics.

[232] M. Woodroofe. A one-armed bandit problem with a concomitant variable. *Journal of the American Statistical Association*, 74(368):799–806, 1979.

[233] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu. Collaborate edge and cloud computing with distributed deep learning for smart city internet of things. *IEEE Internet of Things Journal*, 7(9):8099–8110, 2020.

[234] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. *CVPR*, pages 4820–4828, 2016.

[235] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[236] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[237] J. Xiao, M. Yin, Y. Gong, X. Zang, J. Ren, and B. Yuan. Comcat: Towards efficient compression and customization of attention-based vision models. *arXiv preprint arXiv:2305.17235*, 2023.

[238] Q. Xie, M.-T. Luong, E. Hovy, and Q. V. Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10687–10698, 2020.

[239] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8*, pages 563–574. Springer, 2019.

[240] S. Xu, H. Li, B. Zhuang, J. Liu, J. Cao, C. Liang, and M. Tan. Generative low-bitwidth data free quantization. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 1–17. Springer, 2020.

[241] M. Yan, C. A. Chan, A. F. Gygax, J. Yan, L. Campbell, A. Nirmalathas, and C. Leckie. Modeling the total energy consumption of mobile network services and applications. *Energies*, 12(1):184, 2019.

[242] R. Yang, B. Wang, and M. Bilgic. Idgi: A framework to eliminate explanation noise from integrated gradients. *CVPR*, 2023.

[243] H. Yin, P. Molchanov, J. M. Alvarez, Z. Li, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[244] S. Yu, Z. Yao, A. Gholami, Z. Dong, S. Kim, M. W. Mahoney, and K. Keutzer. Hessian-aware pruning and optimal neural implant. *WACV*, pages 3880–3891, 2022.

[245] X. Yu, T. Serra, S. Zhe, and S. Ramalingam. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. *arXiv preprint arXiv:2203.04466*, 2022.

[246] E. Yvinec, A. Dapgony, M. Cord, and K. Bailly. Rex: Data-free residual quantization error expansion. *Advances in Neural Information Processing Systems*, 2023.

[247] E. Yvinec, A. Dapogny, and K. Bailly. To fold or not to fold: a necessary and sufficient condition on batch-normalization layers folding. In *Thirty-First International Joint Conference on Artificial Intelligence (IJCAI 22)*, pages 1601–1607. International Joint Conferences on Artificial Intelligence Organization, 2022.

[248] E. Yvinec, A. Dapogny, and K. Bailly. Designing strong baselines for ternary neural network quantization through support and mass equalization. *arXiv preprint arXiv:2306.17442*, 2023.

[249] E. Yvinec, A. Dapogny, and K. Bailly. Gradient-based post-training quantization: Challenging the status quo. *arXiv preprint arXiv:2308.07662*, 2023.

[250] E. Yvinec, A. Dapogny, and K. Bailly. Nupes: Non-uniform post-training quantization via power exponent search. *arXiv preprint arXiv:2308.05600*, 2023.

[251] E. Yvinec, A. Dapogny, and K. Bailly. Safer: Layer-level sensitivity assessment for efficient and robust neural network inference. *arXiv preprint arXiv:2308.04753*, 2023.

[252] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly. Red: Looking for redundancies for data-freestructured compression of deep neural networks. *Advances in Neural Information Processing Systems*, 34:20863–20873, 2021.

[253] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly. Red++: Data-free pruning of deep neural networks via input splitting and output merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3664–3676, 2022.

[254] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly. Singe: Sparsity via integrated gradients estimation of neuron relevance. *Advances in Neural Information Processing Systems*, 35:35392–35403, 2022.

[255] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly. Powerquant: Automorphism search for non-uniform quantization. *International Conference on Learning Representations*, 2023.

[256] E. Yvinec, A. Dapogny, M. Cord, and K. Bailly. Spiq: Data-free per-channel static input quantization. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3869–3878, 2023.

[257] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[258] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan. Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition. *IEEE Access*, 7:133509–133520, 2019.

[259] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[260] D. Zhang, J. Yang, D. Ye, and G. Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. *ECCV*, pages 365–382, 2018.

[261] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[262] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[263] S. Q. Zhang, B. McDanel, H. Kung, and X. Dong. Training for multi-resolution inference using reusable quantization terms. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 845–860, 2021.

[264] X. Zhang, H. Qin, Y. Ding, R. Gong, Q. Yan, R. Tao, Y. Li, F. Yu, and X. Liu. Diversifying sample generation for accurate data-free quantization. In *CVPR*, pages 15658–15667, 2021.

[265] Y. Zhang, Z. Zhang, and L. Lew. Pokebnn: A binary pursuit of lightweight accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12475–12485, 2022.

[266] Q. Zhao, M. Sugiyama, L. Yuan, and A. Cichocki. Learning efficient tensor representations with ring-structured networks. In *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 8608–8612. IEEE, 2019.

[267] R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang. Improving neural network quantization without retraining using outlier channel splitting. In *ICML*, pages 7543–7552, 2019.

[268] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *ICLR*, 2017.

[269] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 633–641, 2017.

[270] G. Zhou, J. Li, and Z. Jia. Power-saving exploration for high-end ultra-slim laptop computers with miniature loop heat pipe cooling module. *Applied Energy*, 239:859–875, 2019.

[271] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[272] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[273] D. Zou and Q. Gu. An improved analysis of training over-parameterized deep neural networks. *Advances in neural information processing systems*, 32, 2019.

Chapter 5

Publications

In this chapter, we provide a snapshot of each article that we published during this thesis. The articles are sorted by publication type.

5.1 International Conferences

RED: Looking for redundancies for data-free structured compression of deep neural networks [252] RED was the first research work that we conducted during this PhD thesis. The core idea is a data-free compression technique relying on three steps: weight hashing, neuron merging and depthwise separation decomposition. we were published at NeurIPS 2021 which unfortunately was fully virtual. We presented it at NeurIPS in Paris instead. We would like to thank Pierre Marion and the Sorbonne center for artificial intelligence (SCAI) for organizing this event.

To fold or not to fold: a necessary and sufficient condition on batch-normalization layers folding [247] This article was published at IJCAI 2022 and simply solves batch-normalization folding optimally with a necessary and sufficient condition for folding. IJCAI was the first conference that we attended.

Singe: Sparsity via integrated gradients estimation of neuron relevance [254] SInGE was motivated by two elements: first, we observed that gradient-based pruning was not very effective and figured an intuition as to why. Second, Jules Bonnard (a colleague PhD student from Sorbonne) was working with attribution techniques which alleviated the shortcomings of gradient-based importance estimation for the inputs. As a result, SInGE bridges the gap between attribution and pruning. It was published at NeurIPS 2022 which was by far my favorite conference to attend.

SPIQ: Data-free per-channel static input quantization [256] SPIQ was specifically designed after PowerQuant as we discovered an implementation mistake which was related to the dimensionality constraint that we study in SPIQ. The paper was published at WACV 2023.

PowerQuant: Automorphism search for non-uniform quantization [255] The PowerQuant method started from the disappointment of the difficulty from leveraging non-uniform quantization. Consequently, we wanted to use a method that maps multiplications to multiplications. The resulting method happened to be my best quantization technique in terms of accuracy with respect to the bit-width, but did not happen to be easier to leverage. PowerQuant was published at ICLR 2023

Designing strong baselines for ternary neural network quantization through support and mass equalization [248] This work was a part of the original REx article and specifically improves residual expansions for ternary quantization. It was published at ICIP 2023.

Fighting over-fitting with quantization for learning deep neural networks on noisy labels [208] This research was conducted with the help of Gauthier Tallec. The question that is studied was: are compression techniques good regularizers to fight overfitting and noisy labels? Gauthier helped me

with benchmarking this on affective computing tasks, which are known to be prone to overfitting. In short, yes. The full study was published at ICIP 2023

REx: Data-Free Residual Quantization Error Expansion [246] REx was the second work a conducted during this PhD thesis and has been quite challenging to publish. After several rejections, we decided to simplify the method and only keep the residual expansion and its sparse counterpart in this paper. This work was published at NeurIPS 2023.

5.2 International Journals

Red++: Data-free pruning of deep neural networks via input splitting and output merging [253] In RED++, we provided a thorough theoretical evaluation of the method introduced in RED and proposed a method to leverage memory access rather than compute. This extension was published in TPAMI.

5.3 National Conferences

Red++: Data-free pruning of deep neural networks via input splitting and output merging [253] re-publication at RFIAP.

5.4 Under Review

SAfER: Layer-Level Sensitivity Assessment for Efficient and Robust Neural Network Inference [251] In SAfER, we wanted to address two problems at once: robust inference and layer ranking. The core idea was proposed by Valeo in the Confiance AI group: identify layers that are least important for efficient and robust inference in the context of critical systems.

NUPES: Non-Uniform Post-Training Quantization via Power Exponent Search [250] In NUPES, we extended our work from PowerQuant to GPTQ methods and LLMs.

Gradient-Based Post-Training Quantization: Challenging the Status Quo [249] In this work, we wanted to find for future leads on post-training quantization. The core idea was to test as many implicit assumptions from AdaRound [156] as possible.

NETWORK MEMORY FOOTPRINT COMPRESSION THROUGH JOINTLY LEARNABLE CODEBOOKS AND MAPPINGS In this work, we propose a novel approach to weight hashing.

5.5 Patents

WO/2023/083808QUANTIFICATION DE RÉSEAU NEURONAL

WO/2023/012316 - SUPPRESSION D'OPÉRATION POUR LA RÉDUCTION DE CALCUL INFORMATIQUE D'UNE INFÉRENCE DE RÉSEAU NEURONAL

Table A.1: We report the accuracy obtained from switching the original GELU activation for a ReLU activation function. The naive switch consists in simply replacing the GELU for a ReLU without further work.

baseline accuracy	naive relu switch	optimized relu switch
81.820	68.753	80.950

Appendix A

Partial-Transformer Self-Distillation

In order to replace non-ReLU activation functions in a pre-trained model, we propose to perform block-wise self-distillation. Let's consider a transformer F comprising B transformer blocks $(T_b)_{b\in[\![1;B]\!]}$. We recall that a transformer block computes a layer normalization of the inputs, followed by a multi-head self-attention and a feed forward network. The corresponding graph is illustrated in Figure 1.1, where the second red block corresponds to the GELU or SiLU activation function[1]. Then, our goal is to replace this layer by a ReLU in every block. However doing so in a naive fashion leads to a non-negligible accuracy drop. Consequently, our method aims at recovering from this drop.

We draw inspiration from AdaRound[2] [156]. Let's consider a subset from the training set of 1024 data points (images in the case of ImageNet). If we note $\tilde{T}$ the architecture that uses ReLU, then our goal is to find new weight values $\tilde{W}$ such that

$$\mathbb{E}\left[\|\tilde{T}(X) - T(X)\|\right] \approx 0. \tag{A.1}$$

In other words, we want to minimize the distance between the two models. We perform the optimization block per block in sequence from the first block to the last one. Let's consider Y and $\tilde{Y}$ the intermediate features of T and $\tilde{T}$ respectively, obtained from the already optimized first b blocks. As we focus on the $b+1$ block, we want to minimize the euclidean norm of the error on the intermediate features

$$\left\|T_{b+1}(Y) - \tilde{T}_{b_1}(\tilde{Y})\right\|_2. \tag{A.2}$$

The intuition behind this sequential optimization is that we fine-tune the current block based on the intermediate features that would actually be used at inference. In practice, the optimization is performed using AdaMax [111] with default parameters and an initial learning of 10^{-4} for 1500 optimization steps and a batch size of 32 examples. In order to operate a smooth transition from the GELU to the ReLU activation, we performed a convex combination of the two activations during the optimization. The scheduling of this combination was linear from full GELU (initially) to a full ReLU (end of the optimization), with a warm up of a 100 steps. Our results are reported in Table A.1

We observe that a simple change of activation function (naive relu switch) without any form of fine-tuning leads to a catastrophic accuracy loss of 12.227 points. This result is reassuring as it entails that the GELU plays a significant role in the definition of the learned predictive function. On the other hand, we observe that the proposed optimization process manages to efficiently switch the GELU for a ReLU at a marginal accuracy cost of 0.446 points.

Replacing the activation function can lead to significant accuracy benefits when performed in combination with other compression techniques, especially quantization. to complete

[1] The first red block corresponds to a softmax.

[2] We do not detail the specificities of AdaRound here, as it will be our main focus for section 3.2 in the Quantization chapter.

Appendix B

Quantization Implementations

Appendix C

Redundancy-based pruning theory: RED and RED++

The content of these appendix was extracted from the RED [252] and RED++ [253] articles.

C.1 Detailed proofs

For simplicity, we propose to first study the case of a simple perceptron f with weights $W \in \mathbb{R}^{n^0 \times n^1}$ before extending the result to the whole network. We introduce the pseudo distance between the original weight values and their hashed version $|w - \tilde{w}|$ (note that this is a difference between scalars). We have the following upper bound:

$$|w - \tilde{w}| \leq \min_{m \in M^+} \{m > w\} - \max_{m \in M^+} \{m < w\} \tag{C.1}$$

This follows from the fact that hashing is based on a partition of the support of W, *i.e.* for all consecutive pairs $m_i, m_{i+1} \in M^+$, $\forall w \in [m_i; m_{i+1}]$, $\tilde{w} \in \{m_i, m_{i+1}\}$. We can deduce an upper bound on the expected value of $|w - \tilde{w}|$

$$\mathbb{E}[|w - \tilde{w}|] \leq \sum_{m_i, m_{i+1} \in M^+ \cup \{m_1^-, m_{|M^-|}^-\}} (m_{i+1} - m_i) \int_{m_i}^{m_{i+1}} \mathbb{P}_w dw \tag{C.2}$$

where dw is the density of W and $m_1^-, m_{|M^-|}^-$ are the minimum and maximum of M^- respectively where $\mathbb{P}_w$ is the density of the weights.

We derive another upper bound using two properties of the density estimation. First, on each $[m_i^-, m_{i+1}^-]$ d^l is monomodal. Second, its variance can be estimated. We have:

$$\mathbb{E}\left[|w - \tilde{w}|\right] = \int_{W^{\min}}^{W^{\max}} |w - \tilde{w}| \mathbb{P}_w dw \tag{C.3}$$

where $W^{\min}$ and $W^{\max}$ are the minimum and maximum of W. This expression can be split in a sum over the partition:

$$\mathbb{E}\left[|w - \tilde{w}|\right] = \sum_{i=1}^{|M^+|} \int_{m_i^-}^{m_{i+1}^-} \left|w - m_i^+\right| \mathbb{P}_w dw \tag{C.4}$$

where the m_i^+ and m_i^- are the ordered elements of M^+ and M^-. We know from [20], that the mean and mode of a unimodal distribution lie within $\sqrt{3}$ standard deviations of each other. Because of the kernel used in KDE, we have a sum of Gaussians, thus we can use the fact that the average absolute distance of a random sample to the distribution mean is $\sigma\sqrt{2/\pi}$. The triangular inequality thus brings:

$$\mathbb{E}\left[|w - \tilde{w}|\right] \approx \int |w - \tilde{w}| d(w) dw \leq \max_{i \in [\![1; |M^+|]\!]} \sigma_i \left(\sqrt{\frac{2}{\pi}} + \sqrt{3}\right) \tag{C.5}$$

where σ_i is the standard deviation of d restricted to $[(M^-)_i; (M^-)_{i+1}]$. We compute σ_i using equation 2.3.

$$\sigma_i^2 = \int_{(M^-)_i}^{(M^-)_{i+1}} \frac{1}{n^i \Delta} \sum_{n=1}^{n^i} K\left(\frac{w - w_n}{\Delta}\right) w^2 dw - \mathbb{E}_d[X]^2 \tag{C.6}$$

Because K is a Gaussian kernel we can deduce the value of σ_i and update the formula for $\mathbb{E}\left[|w - \tilde{w}|\right]$ the upper bound.

$$\mathbb{E}\left[|w - \tilde{w}|\right] \leq \frac{\Delta}{\sqrt{2\pi}} \left(\sqrt{\frac{2}{\pi}} + \sqrt{3}\right) \tag{C.7}$$

Now we have two upper bounds for $\mathbb{E}$. First A, from equation C.2, based on the pseudo distance and the hashing properties. Second B, from equation C.7, based on KDE properties. We can combine these bounds to obtain:

$$\mathbb{E}\left[|w - \tilde{w}|\right] \leq \min\{A, B\} = u \tag{C.8}$$

u is our per-weight upper bound on the error. In practice both A and B are relevant and used situationally. This result can be extended to DNN with multiple layers.

Multi-layer Preservation Through Hashing

In order to generalize the previous upper bound to a feed forward CNN with L layers, we first compute the upper bound for a layer f^l. The upper bound u_l measures the error on each weight values. The weights are used in scalar products with $n^{l-1} \times w^l \times h^l \times n^l$ elements. The average error behaves following the Central Limit Theorem. We detail the computation of equations C.15 and C.22. A convolution is defined by the following operation

$$\text{Output}_{i,j,l} = \sum_{\delta i, \delta j, k} \text{Input}_{i+\delta i, j+\delta j, k} W_{i+\delta i, j+\delta j, k, l} \tag{C.9}$$

This gives us $n^{l-1} w^l h^l$ multiplications per output. For each of these operations we have the upper bound u_l. Therefore, the errors are sampled in $[-u_l; u_l]$. The Central limit theorem gives us that the average error converges to a standard Gaussian distribution $\mathcal{N}(0, 1)$ and we get that

$$\mathbb{E}[\|\tilde{f}^l - f^l\|] \leq \frac{u_l}{\sqrt{n^{l-1} w^l h^l}} \tag{C.10}$$

assuming no activation function on layer l. Now let's assume we have $L = 2$, i.e. the DNN $f : x \mapsto f^2(f^1(x))$ follows

$$\mathbb{E}_X[\|\tilde{f} - f\|] = \mathbb{E}_X[\|\tilde{f}^2(\tilde{f}^1(X)) - f^2(f^1(X))\|] \tag{C.11}$$

where X is the random variable defined by the inputs. Because f is piece-wise affine, we can assert that

$$\begin{aligned} \mathbb{E}_X[\|\tilde{f} - f\|] \simeq & \mathbb{E}_X[\|\tilde{f}^2(f^1(X)) + \tilde{f}^2(\|\tilde{f}^1(X) - f^1(x)\|) \\ & - f^2(f^1(X))\|] \end{aligned} \tag{C.12}$$

Now using the triangular inequality, we get

$$\mathbb{E}_X[\|\tilde{f} - f\|] \simeq \mathbb{E}_{f^1(X)}[\|\tilde{f}^2 - f^2\| + \mathbb{E}_{\|\tilde{f}^1(X) - f^1(x)\|}[\tilde{f}^2] \tag{C.13}$$

We deduce

$$\mathbb{E}_X[\|\tilde{f} - f\|] \leq \mu^1 \mathbb{E}[\|\tilde{f}^2 - f^2\|] + \mu^2 \mathbb{E}[\|\tilde{f}^1 - f^1\|] \tag{C.14}$$

and it follows equation C.22.

This adds a multiplicative term $\frac{1}{\sqrt{n^{l-1} w^l h^l}}$ to the expected error per layer. Furthermore, we need to take into account the activation function. Assuming a ReLU activation function, statistically, the average proportion of negative inputs is given by the CDF of a Gaussian distribution of parameters μ^l and σ^l. These statistics are obtained from the batch normalization layers. This adds a multiplicative term $\frac{1}{2}\left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l \sqrt{2}}\right)\right)$ where erf is the Gauss error function, $\text{erf} : z \mapsto \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt$. Therefore we get

$$\mathbb{E}[\|\tilde{f}^l - f^l\|] \leq \frac{u_l}{\sqrt{n^{l-1} w^l h^l}} \frac{1}{2} \left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l \sqrt{2}}\right)\right) \tag{C.15}$$

where u_l is the layer-wise upper bound described in equation C.8 . We extend this result recursively across all layers. We detail the computation of equations C.15 and C.22. A convolution is defined by the following operation

$$\text{Output}_{i,j,l} = \sum_{\delta i,\delta j,k} \text{Input}_{i+\delta i,j+\delta j,k} W_{i+\delta i,j+\delta j,k,l} \tag{C.16}$$

This gives us $n^{l-1}w^l h^l$ multiplications per output. For each of these operations we have the upper bound u_l. Therefore, the errors are sampled in $[-u_l; u_l]$. The Central limit theorem gives us that the average error converges to a standard Gaussian distribution $\mathcal{N}(0,1)$ and we get that

$$\mathbb{E}[\|\tilde{f}^l - f^l\|] \leq \frac{u_l}{\sqrt{n^{l-1}w^l h^l}} \tag{C.17}$$

assuming no activation function on layer l. Now let's assume we have $L = 2$, i.e. the DNN $f : x \mapsto f^2(f^1(x))$ follows

$$\mathbb{E}_X[\|\tilde{f} - f\|] = \mathbb{E}_X[\|\tilde{f}^2(\tilde{f}^1(X)) - f^2(f^1(X))\|] \tag{C.18}$$

where X is the random variable defined by the inputs. Because f is piece-wise affine, we can assert that

$$\begin{aligned}\mathbb{E}_X[\|\tilde{f} - f\|] \simeq &\mathbb{E}_X[\|\tilde{f}^2(f^1(X)) + \tilde{f}^2(\|\tilde{f}^1(X) - f^1(x)\|) \\ &- f^2(f^1(X))\|]\end{aligned} \tag{C.19}$$

Now using the triangular inequality, we get

$$\mathbb{E}_X[\|\tilde{f} - f\|] \simeq \mathbb{E}_{f^1(X)}[\|\tilde{f}^2 - f^2\| + \mathbb{E}_{\|\tilde{f}^1(X) - f^1(x)\|}[\tilde{f}^2] \tag{C.20}$$

We deduce

$$\mathbb{E}_X[\|\tilde{f} - f\|] \leq \mu^1 \mathbb{E}[\|\tilde{f}^2 - f^2\|] + \mu^2 \mathbb{E}[\|\tilde{f}^1 - f^1\|] \tag{C.21}$$

and it follows equation C.22.

$$U = \prod_{l=1}^{L} \left(\frac{u_l}{\sqrt{n^{l-1}w^l h^l}} \frac{1}{2} \left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l \sqrt{2}} \right) \right) \right) + \mu^l - \prod_{l=1}^{L} \mu^l \tag{C.22}$$

The value of each u_l is a linear function of the bandwidth Δ_l. Therefore U is also a linear function of Δ_l which is very low in practice (see section 2.1): thus the reason why, in practice, the hashing error is very low. In Section C.1, we assumed ReLU activation functions. In this appendix, we extend the theoretical upper bound to SiLU activation function (EfficientNets) and GeLU activation function (image transformers) defined as:

$$\begin{cases} \text{SiLU} : x \mapsto x\sigma(x) \\ \text{GeLU} : x \mapsto \frac{x}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \end{cases} \tag{C.23}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function and erf is the Gauss error function defined as

$$\text{erf} : z \mapsto \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{C.24}$$

In the definition of the proposed upper-bound U on the error introduced by hashing, the impact of the activation function ReLU corresponds to the term $\frac{1}{2}\left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l\sqrt{2}}\right)\right)$ in the equation

$$U_{\text{ReLU}} = \prod_{l=1}^{L} \left(\frac{u_l}{\sqrt{n^{l-1}w^l h^l}} \frac{1}{2} \left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l \sqrt{2}} \right) \right) \right) + \mu^l - \prod_{l=1}^{L} \mu^l \tag{C.25}$$

Intuitively, this term accounts only for positive outputs as the negative outputs are zeroed-out and don't introduce an error. However, this is slightly different in the case of GeLU or SiLU activation functions. We note Pos_l the term $\frac{1}{2}\left(1 - \text{erf}\left(\frac{-\mu^l}{\sigma^l\sqrt{2}}\right)\right)$ which accounts for positive outputs of layer l. For both GeLU and SiLU, the error on such outputs remain bounded by $\frac{u_l}{\sqrt{n^{l-1}w^l h^l}}$. Consequently, we only focus on the negative terms in both cases. For negative terms, in both cases (GeLU and SiLU), the error is bounded by a constant strictly lower than 1. We note these constants $C_{\text{SiLU}} = \min \text{SiLU} \approx 0.27846$ and

$C_{\text{GeLU}} \min \text{GeLU} \approx 0.169971$. This is a direct consequence of their respective definitions. Consequently, the upper bound becomes:

$$\begin{cases} U_{\text{SiLU}} & = \prod_{l=1}^{L} \left(\frac{u_l}{\sqrt{n^{l-1} w^l h^l}} (\text{Pos}_l + (1 - \text{Pos}_l) C_{\text{SiLU}}) \right) + \mu^l \\ & - \prod_{l=1}^{L} \mu^l \\ U_{\text{GeLU}} & = \prod_{l=1}^{L} \left(\frac{u_l}{\sqrt{n^{l-1} w^l h^l}} (\text{Pos}_l + (1 - \text{Pos}_l) C_{\text{GeLU}}) \right) + \mu^l \\ & - \prod_{l=1}^{L} \mu^l \end{cases} \tag{C.26}$$

In practice, on EfficientNets, the value of Pos_l ranges from 0.730 to 0.979. Consequently, the term $(1 - \text{Pos}_l) C_{\text{SiLU}}$ is bounded by 0.075 and is at least one order of magnitude below the original term Pos_l. We deduce that the upper bound U_{SiLU} observe similar behavior as U_{ReLU}.

With regards to transformers, we observe equivalent values for Pos_l and a lower constant C_{GeLU} which leads to the same conclusion: the upper bound U_{GeLU} observe similar behavior as U_{ReLU}. To assess this theoretical study, we still need to empirically validate the hashing protocol. In the following section, we show that the upper bound U provides practical data-free guarantees on the accuracy preservation.

Data-Free Criterion on the Hashing Error

We propose a simple criterion based on the upper bound U and a data-free estimation of the norm of the logits. Assuming that we have the value of the expected logits norm $\mathbb{E}_X[\|f\|]$, we derive the following criterion to decipher whether the hashing is detrimental to the network accuracy:

1. if $\frac{U}{\mathbb{E}_X[\|f\|]} << 1$ then the modifications from hashing won't have a significant impact on the logits. Therefore their order is likely unchanged.

2. if $\frac{U}{\mathbb{E}_X[\|f\|]} \simeq 1$ then the modifications from hashing may have a significant impact on the predictions. As U is an upper bound we can't conclude yet.

To make this evaluation data-free, we propose a data-free estimator of the expected norm of the logits, referred to as $E[\text{norm}]$. To do this, we use the values of the weights of the batch normalization layers [101] as an estimate of the expected value of the shallowest layer and then use the last kernel with the linearity of the expectation to compute $E[\text{norm}]$. Explicitly, assume the shallowest layer f_L with parameters W_L and b_L has a batch-normalization layer for input with parameters $\mu, \sigma, \beta, \gamma$ such that $\text{BN}(x) = \gamma \frac{x - \mu}{\sigma} + \beta$, then our estimate $E[\text{norm}]$ is defined as

$$E[\text{norm}] = W_L \text{act}(\beta) + b_L \tag{C.27}$$

where act is the activation function. Because we use $\mathbb{E}_X[\|f\|]$ as the denominator, we need the estimate to be as close as possible while satisfying $E[\text{norm}] \leq \mathbb{E}_X[\|f\|]$ (which is validated empirically) in order not to have an over-confident criterion. In practice, we also use the variance estimate to obtain a confidence interval.

C.2 Extra empirical validations

parameter α strategy

C.3 Similarity pruning as a birthday problem

In this section, we conduct a probabilistic study of the relationship between reducing the number of unique weight values and the resulting pruning ratio in DNNs. This can be cast as a case of the generalized Birthday Problem. We assume that each weight is sampled independently from a unique distribution. First, assuming the least favorable prior (uniform) on this distribution, we want to compute the expected pruning factor from the merging step as well as the splitting step noted $\mathbb{E}_m$ and $\mathbb{E}_s$ respectively.

Lemma C.3.1. *Under the uniform prior, for any layer with weights $W^l \in \mathbb{R}^{w \times h \times n^{l-1} \times n^l}$ and hashed weights $\tilde{W}^l$ we have*

$$\mathbb{E}_s > \mathbb{E}_m \tag{C.28}$$

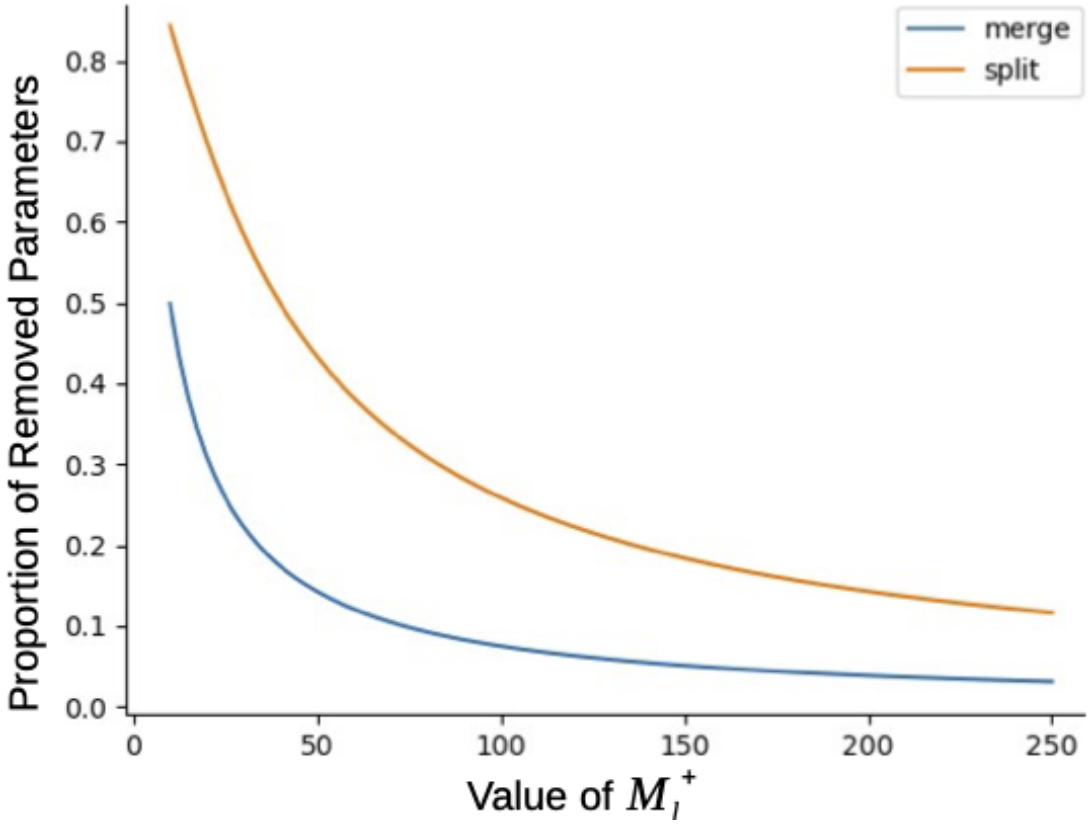


Figure C.1: Given a layer l with weights $W^l \in \mathbb{R}^{3\times3\times32\times128}$ sampled uniformly, we apply hashing such that we keep M_l^+ values. We plot the evolution of $\mathbb{E}_m$ (blue) and $\mathbb{E}_s$ (orange) for different values of M_l^+ using equations C.31 and C.32.

Proof. First we recall the probability $\mathbb{P}_n^K(k)$ to get k distinct values from K possible ones with a sampling size n. The probability that each of n samples belongs to those k values is $(k/K)^n$. However, this also includes cases where some of the k numbers were not chosen. The inclusion-exclusion rule says that the probability of drawing all of those k values is $(-1)^i \binom{k}{i} \left(\frac{k-i}{K}\right)^n$. As a consequence, we have:

$$\mathbb{P}_n^K(k) = \binom{K}{k} \sum_{i=0}^{k} (-1)^i \binom{k}{i} \left(\frac{k-i}{K}\right)^n \tag{C.29}$$

from this expression we can compute $\mathbb{E}_m$ and replace $K = whn^{l-1}M_l^+$ and $n = n^l$. This result is obtained by developing the standard definition of the expected value of a discreet variable. For $\mathbb{E} = \mathbb{E}_m$ or $\mathbb{E}_s$ we have:

$$\mathbb{E} = \sum_{k=0}^{K} k\mathbb{P}_n^K(k) = K\left(1 - \left(1 - \frac{1}{K}\right)^n\right) \tag{C.30}$$

it follows that the expected pruning ratio of the merging step is:

$$\mathbb{E}_m = 1 - \frac{whn^{l-1}M_l^+\left(1 - \left(1 - \frac{1}{whn^{l-1}M_l^+}\right)^{n^l}\right)}{n^l} \tag{C.31}$$

We illustrate in Fig C.1 the variation of $\mathbb{E}_m$ as a function of the proportion of the unique values among hashed weights over the original number of distinct weights, *i.e.* $M_l^+/(whn^{l-1}n^l)$ for different values of n^{l-1}. These results suggest that the larger the input dimension n^{l-1} the lower the number of redundancies. The value of $\mathbb{E}_s$ can be computed similarly with different values of K and n. We obtain

$$\mathbb{E}_s = 1 - \frac{whM_l^+\left(1 - \left(1 - \frac{1}{whM_l^+}\right)^{n^l}\right)}{n^l} \tag{C.32}$$

Similarly to $\mathbb{E}_m$, we illustrate $\mathbb{E}_s$ in Fig C.1. □

To extend the previous result we need a preliminary result which extends the birthday problem to non-uniform sampling. In this section we note $E = [\![1; m]\!]$ the sampling space, $(X_j)_{i[\![1;n]\!]}$ the i.i.d. variables sampled from law $\mathcal{L}$ which satisfy

$$\forall i \in E, \quad p_i = \mathbb{P}_{\mathcal{L}}(X_j = i) > 0 \tag{C.33}$$

We introduce $(Y_i)_{i\in E}$ the number of samples $X_j = i$, *i.e.* $Y_i = \sum_{j=1}^{n} \mathbb{1}_{X_j=i}$. In our case, we compute the expected value of the number V of distinct values in the sample, *i.e.*

$$V = \sum_{i=1}^{m} \mathbb{1}_{Y_i>0} = m - \sum_{i=1}^{m} \mathbb{1}_{Y_i=0} \tag{C.34}$$

We consider i a value belonging to a subset $K \subset E$. We note B_i the event where the value i is not sampled:

$$\begin{cases} \mathbb{P}_{\mathcal{L}}\left(\bigcap\limits_{i\in K} B_i\right) = \left(1 - \sum_{i\in K} p_i\right)^n \\ \mathbb{E}_{\mathcal{L}}[V] = \sum_{v=1}^{m} v \sum_{K\in A^m_{m-v}} \left(1 - \sum_{i\in K} p_i\right)^n \end{cases} \tag{C.35}$$

where A^m_{m-v} is the set of arrangements of E.

Lemma C.3.2. *Under the priors $\mathcal{L}^m$ on $x = (x_1, \dots, x_{n^{l-1}}) \in {M^l_+}^{w\times h\times n^{l-1}}$ and $\mathcal{L}^s$ on $x_j \in {M^l_+}^{w\times h}$, for any layer with hashed weights $\tilde{W}^l \in \mathbb{R}^{w\times h\times n^{l-1}\times n^l}$, such that $n^l = n^{l-1}$:*

$$\mathbb{E}_s > \mathbb{E}_m \tag{C.36}$$

Proof. The expected pruning factor for the splitting step is higher than for the merging step if and only if $\mathbb{E}_{\mathcal{L}^s}[V] < \mathbb{E}_{\mathcal{L}^m}[V]$, with V defined in equation C.34, that is to say that the expected number of remaining neurons is lower under the prior $\mathcal{L}^s$. Let's develop $\mathbb{E}_{\mathcal{L}^m}[V]$, following eq C.35 we simply replace the notations with $m = n^l$ and the $p_i = \mathbb{P}_{\mathcal{L}^m}$. Thus we get

$$\mathbb{E}_{\mathcal{L}^m}[V] = \sum_{v=1}^{m_m} v \sum_{K\in A^{m_m}_{m_m-v}} \left(1 - \sum_{i\in K} \mathbb{P}_{\mathcal{L}^m}(X = (x_1, ..., x_{n^{l-1}})\right)^{n^l} \tag{C.37}$$

where $m_m = |M^+_l| whn^{l-1}$. In the case of $\mathcal{L}^s$, like in the case of $\mathcal{L}^m$ we have n^l samples, thus we replace $n = n^l$ and $p_i = \mathbb{P}_{\mathcal{L}^s}$ to obtain

$$\mathbb{E}_{\mathcal{L}^s}[V] = \sum_{v=1}^{m_s} v \sum_{K\in A^{m_s}_{m_s-v}} \left(1 - \sum_{i\in K} \mathbb{P}_{\mathcal{L}^s}(X = x_i)\right)^{n^l} \tag{C.38}$$

where $m_s = |M^+_l| wh$. However by equation C.33, it follows that

$$\mathbb{P}_{\mathcal{L}^m}(X = x) = \prod_{j=1}^{n^{l-1}} \mathbb{P}_{\mathcal{L}^s}(X_j = x_j) > 0 \tag{C.39}$$

and $\mathbb{P}_{\mathcal{L}^m}(x) < \mathbb{P}_{\mathcal{L}^s}(x')$ when x' is a coordinate of x. In consequence, if we compare equation C.37 and C.38, we have a larger sum of larger terms. Therefore, we get the desired result. □

We propose in Fig C.2 an extension of this result for different configurations of n^{l-1} and n^l. We tested the Gaussian, exponential and uniform priors, in the case of 3×3 convolutional layer with weights $W \in \mathbb{R}^{3\times3\times n^{l-1}\times n^l}$ and $M^+ = 100$. We vary the input dimension n^{in} linearly from 2 to 128 with $n^l n^{l-1} = 64^2$ and plot E_m and E_s for each prior. In particular, as stated in lemma C.3.2, with $n^l = n^{l-1} = 64$, we have $\mathbb{E}_s > \mathbb{E}_m$ for all priors. We also observe that splitting performs better for larger input dimensions while merging is better for smaller ones, showing the complementarity between the two. The uniform prior is the least favorable while the exponential is the most favorable. However a limit to the theory is the i.i.d. hypothesis which is probably not satisfied in practice. We however show through the experiments that, despite discrepancies between theoretical and empirical data, the trends introduced in this sections remains true.

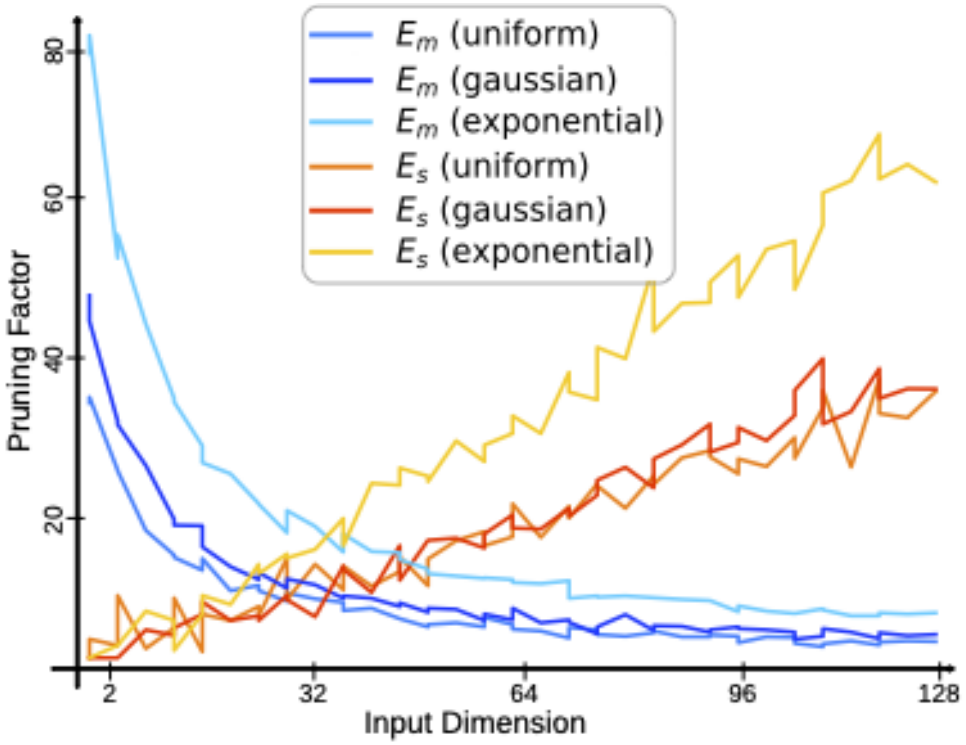


Figure C.2: Given a layer l with weights $W^l \in \mathbb{R}^{3\times3\times n^{l-1}\times n^l}$ we fix the value of the product $n^{l-1}\times n^l = 64^2$ and we plot the empirical values of the pruning ratios $\mathbb{E}_m$ (nuances of blue) and $\mathbb{E}_s$ (nuances of orange) for different values of n^{l-1} the input dimension. The considered priors are the discreet Gaussian, the exponential and uniform distribution. We observe the complementarity of merging and splitting.

C.4 New Pruning Paradigm

For the sake of clarity, let's consider a specific example: the Strassen matrix multiplication algorithm [202]. In short, let's consider two matrices A and B then the algorithm reads

$$A \times B = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \times \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}, \tag{C.40}$$

can be computed using intermediate results defined as

$$\begin{aligned} M_1 &= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\ M_2 &= (A_{2,1} + A_{2,2})B_{1,1} \\ M_3 &= A_{1,1}(B_{1,2} - B_{2,2}) \\ M_4 &= A_{2,2}(B_{2,1} - B_{1,1}) \\ M_5 &= (A_{1,1} + A_{1,2})B_{2,2} \\ M_6 &= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\ M_7 &= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2}) \end{aligned} \quad . \tag{C.41}$$

The resulting intermediate operations only use 7 instead of 8 multiplications as compared to the naive algorithm. Stemming on these intermediate computations, we derive the multiplied matrix with

$$A \times B = \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{pmatrix}. \tag{C.42}$$

Then, the new pruning granularity that we would advise to investigate is the matrix inference algorithm pruning. In other words, do not compute some of the intermediate M_k multiplications. This kind of pruning could be specific to a layer, block of layers or even an entire neural network. While such pruning would not address the memory footprint challenges, it would offer an explicit inference-specific improvement.

Appendix D

PowerQuant Proofs

The content of this appendix was extracted from the REx [246] and PowerQuant [255] articles.

D.1 Proof of Lemma 3.1.1

In this section, we provide a simple proof for lemma 3.1.1 as well as a discussion on the continuity hypothesis.

Proof. We have that $\forall x \in \mathbb{R}_+, Q(x) \times Q(0) = Q(0)$ and $\forall x \in \mathbb{R}_+, Q(x) \times Q(1) = Q(x)$ which induces that Q is either the constant 1 or $Q(0) = 0$ and $Q(1) = 1$. Because Q is an automorphism, we can eliminate the first option. Now, we will demonstrate that Q is necessarily a power function. Let n be an integer, then

$$Q(x^n) = Q(x) \times Q(x^{n-1}) = Q(x)^2 \times Q(x^{n-2}) = \cdots = Q(x)^n. \tag{D.1}$$

Similarly, for fractions, we get $Q(x^{\frac{1}{n}}) \times \cdots \times Q(x^{\frac{1}{n}}) = Q(x) \Leftrightarrow Q(x^{\frac{1}{n}}) = Q(x)^{\frac{1}{n}}$. Assuming Q is continuous, we deduce that for any rational $a \in \mathbb{R}$, we have

$$Q(x^a) = Q(x)^a \tag{D.2}$$

In order to verify that the solution is limited to power functions, we use a *reductio ad absurdum*. Assume Q is not a power function. Therefore, there exists $(x, y) \in \mathbb{R}_+^2$ and $a \in \mathbb{R}$ such that $Q(x) \neq x^a$ and $Q(y) = y^a$. By definition of the logarithm, there exists b such that $x^b = y$. We get the following contradiction, from (D.2),

$$\begin{cases} Q(x^{b^a}) = Q(y^a) = y^a \\ Q(x^{b^a}) = Q(x^{a^b}) = Q(x^a)^b \neq \left(x^{a^b} = y^a\right) \end{cases} \tag{D.3}$$

Consequently, the suited functions Q are limited to power functions *i.e.* $\mathcal{Q} = \{Q : x \mapsto x^a | a \in \mathbb{R}\}$. □

We would also like to put the emphasis on the fact that there are other Automorphisms of $(\mathbb{R}, \times)$. However, the construction of such automorphisms require the axiom of choice [88]. Such automorphisms are not applicable in our case which is why the key constraint is being an automorphism rather than the continuous property.

D.2 Local Convexity

We prove that the minimization problem defined in equation 3.14 is locally convex around the solution a^*. Formally we prove that

$$x \mapsto \left\| x - Q_a^{-1}\left(Q_a(x)\right) \right\|_p \tag{D.4}$$

is locally convex around a^* defined as $\arg\min_a \left\| x - Q_a^{-1}\left(Q_a(x)\right) \right\|_p$.

Lemma D.2.1. *The minimization problem defined as*

$$\arg\min_a \left\{ \left\| x - Q_a^{-1}\left(Q_a(x)\right) \right\|_p \right\} \tag{D.5}$$

is locally convex around any solution a^*.

Proof. We recall that $\frac{\partial x^a}{\partial a} = x^a \log(x)$. The function $\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|$ is differentiable. We assume $x \in \mathbb{R}$, then we can simplify the sign functions (assume x positive without loss of generality) and note $y = \max|x|$, then

$$\frac{\partial Q_a^{-1}\left(Q_a(x)\right)}{\partial a} = \frac{\partial \left|\left\lfloor (2^{b-1}-1)\frac{x^a}{y^a}\right\rfloor \frac{y^a}{2^{b-1}-1}\right|^{\frac{1}{a}}}{\partial a}. \tag{D.6}$$

This simplifies to

$$\frac{\partial Q_a^{-1}\left(Q_a(x)\right)}{\partial a} = y\frac{\partial \left(\frac{\left\lfloor B\left(\frac{x}{y}\right)^a\right\rfloor}{B}\right)^{\frac{1}{a}}}{\partial a}, \tag{D.7}$$

with $B = 2^{b-1}-1$. By using the standard differentiation rules, we know that the rounding operator has a zero derivative a.e.. Consequently we get,

$$\frac{\partial Q_a^{-1}\left(Q_a(x)\right)}{\partial a} = -a^2 y\left(\frac{\left\lfloor B\left(\frac{x}{y}\right)^a\right\rfloor}{B}\right)^{\frac{1}{a}} \log\left(\frac{\left\lfloor B\left(\frac{x}{y}\right)^a\right\rfloor}{B}\right). \tag{D.8}$$

Now we can compute the second derivative of $Q_a^{-1}\left(Q_a(x)\right)$,

$$\frac{\partial^2 Q_a^{-1}\left(Q_a(x)\right)}{\partial a^2} = a^4 y\left(\frac{\left\lfloor B\left(\frac{x}{y}\right)^a\right\rfloor}{B}\right)^{\frac{1}{a}} \log^2\left(\frac{\left\lfloor B\left(\frac{x}{y}\right)^a\right\rfloor}{B}\right). \tag{D.9}$$

From this expression, we derive the second derivative, using the property $(f\circ g)'' = f''\circ g \times g'^2 + f'\circ g \times g''$ and the derivatives $|\cdot|^{\frac{1}{p}\prime} = \frac{x|x|^{\frac{1}{p}-2}}{p}$ and $|\cdot|^{\frac{1}{p}\prime\prime} = \frac{1-p}{p^2}\frac{|x|^{\frac{1}{p}}}{x^2}$, then for any $x_i \in x$

$$\begin{aligned}\frac{\partial^2\left|x_i - Q_a^{-1}\left(Q_a(x_i)\right)\right|}{\partial a^2} &= \frac{1-p}{p^2}\frac{|x_i - Q_a^{-1}(Q_a(x_i)|^{\frac{1}{p}}}{(x_i - Q_a^{-1}(Q_a(x_i))^2}\left(\frac{\partial Q_a^{-1}\left(Q_a(x)\right)}{\partial a}\right)^2 \\ &+ \frac{(x_i - Q_a^{-1}(Q_a(x_i))|x_i - Q_a^{-1}(Q_a(x_i)|^{\frac{1}{p}-2}}{p}\frac{\partial^2 Q_a^{-1}\left(Q_a(x)\right)}{\partial a^2}\end{aligned} \tag{D.10}$$

We now note the first term in the previous addition $T_1 = \frac{1-p}{p^2}\frac{|x_i - Q_a^{-1}(Q_a(x_i)|^{\frac{1}{p}}}{(x_i - Q_a^{-1}(Q_a(x_i))^2}\left(\frac{\partial Q_a^{-1}(Q_a(x))}{\partial a}\right)^2$ and the second term as a product of $T_2 = \frac{(x_i - Q_a^{-1}(Q_a(x_i))|x_i - Q_a^{-1}(Q_a(x_i)|^{\frac{1}{p}-2}}{p}$ times $T_3 = \frac{\partial^2 Q_a^{-1}(Q_a(x))}{\partial a^2}$. We know that $T_1 > 0$ and $T_3 > 0$, consequently, and T_2 is continuous in a. At a^* the terms with $|x_i - Q_a^{-1}\left(Q_a(x_i)\right)|$ are negligible in comparison with $\frac{\partial^2 Q_a^{-1}(Q_a(x))}{\partial a^2}$ and $\left(\frac{\partial Q_a^{-1}(Q_a(x))}{\partial a}\right)^2$. Consequently, there exists an open set around a^* where $T_1 > |T_2|T_3$, and $\frac{\partial^2\left|x_i - Q_a^{-1}(Q_a(x_i))\right|}{\partial a^2} > 0$. This concludes the proof. □

D.3 Uniqueness of the Solution

In this section we provide the elements of proof on the uniqueness of the solution of the minimization of the quantization reconstruction error.

Lemma D.3.1. *The minimization problem over $x \in \mathbb{R}^N$ defined as*

$$\arg\min_a \left\{\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|_p\right\} \tag{D.11}$$

has almost surely a unique global minimum a^*.

Proof. We assume that x can not be exactly quantized, *i.e.* $\min_a \left\{\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|_p\right\} > 0$ which is true almost everywhere. We use a *reductio ad absurdum* and assume that there exist two optimal solutions a_1 and a_2 to the optimization problem. We expand the expression $\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|_p$ and get

$$\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|_p = \left\|x - \left|\left\lfloor (2^{b-1}-1)\frac{\operatorname{sign}(x)\times|x|^a}{\max|x|^a}\right\rfloor \frac{\max|x|^a}{2^{b-1}-1}\right|^{\frac{1}{a}}\operatorname{sign}(x)\right\|_{\cdot p} \tag{D.12}$$

We note the rounding term R_a and get

$$\left\|x - Q_a^{-1}\left(Q_a(x)\right)\right\|_p = \left\|x - \left|R_a \frac{\max|x|^a}{2^{b-1}-1}\right|^{\frac{1}{a}} \operatorname{sign}(x)\right\|_p . \tag{D.13}$$

Assume $R_{a_1} = R_{a_2} = R$, the minimization problem $\arg\min_a \left\|x - \left|R\frac{\max|x|^a}{2^{b-1}-1}\right|^{\frac{1}{a}} \operatorname{sign}(x)\right\|_p$ is convex and has a unique solution, thus $a_1 = a_2$. Now assume $R_{a_1} \neq R_{a_2}$.

Let's denote $D(R)$ the domain of power values a over which we have $\left\lfloor (2^{b-1}-1)\frac{\operatorname{sign}(x)\times|x|^a}{\max|x|^a}\right\rfloor = R$. If there is a value a outside of $D(R_{a_1}) \cup D(R_{a_2})$ such that R' has each of its coordinate strictly between the coordinates of R_{a_1} and R_{a_2}, then, without loss of generality, assume that at least half of the coordinates of R_{a_1} are further away from the corresponding coordinates of x than one quantization step. This implies that there exists a value a' in $D(R')$ such that $\left\|x - Q_{a'}^{-1}\left(Q_{a'}(x)\right)\right\|_p < \left\|x - Q_{a_1}^{-1}\left(Q_{a_1}(x)\right)\right\|_p$. which goes against our hypothesis. Thus, there are up to N possible values for R that minimize the problem which happens iff x satisfies at least one coordinate can be either ceiled or floored by the rounding. The set defined by this condition has a zero measure. □

Appendix E

REx Proofs

E.1 Exponential Convergence

The exponential convergence can be proved for the two methods: expansion and sparse expansion. We first prove it for the expansion on sequential models, then generalize the result to more diverse architectures. Before detailing the proof of lemma 3.1.3, we empirically motivate the assumption of symmetry over the weight values distribution. In Figure E.1, we plot the distributions of the weights of several layers of a ResNet 50 trained on ImageNet. The assumption is often satisfied in practice. Furthermore, in any instances where it would not be satisfied, it can be enforced using asymmetric quantization.

Lemma E.1.1. *Let f be a layer with weights $W \in \mathbb{R}^n$ with a symmetric distribution. We denote $R^{(k)}$ the k^{th} quantized weight from the corresponding residual error. Then the error between the rescaled $W^{(K)} = Q^{-1}(R^{(K)})$ and original weights W decreases exponentially, i.e.:*

$$\left| w - \sum_{k=1}^{K} w^{(k)} \right| \leq \left(\frac{1}{2^{b-1}-1} \right)^{K-1} \frac{\left(s_{R^{(K)}}\right)_i}{2} \tag{E.1}$$

where w and $w^{(k)}$ denote the elements of W and $W^{(k)}$ and $\left(s_{R^{(k)}}\right)_i$ denotes the row-wise rescaling factor at order k corresponding to w, as defined in equation 3.1.

We work on expanded layers which compute

$$f^{(K)} : x \mapsto \sigma \left(\sum_{k=1}^{K} R^{(k)} Q(x) s_{R^{(k)}} s_x + b \right) \tag{E.2}$$

Proof. Assume $K = 1$, then $W^{(1)}$ is the result of the composition of inverse quantization operator and quantization operator, i.e. $W^{(1)} = s_W \left\lfloor \frac{W}{s_W} \right\rceil$. By definition of the rounding operator we know that $|\lfloor a \rceil - a| \leq 0.5$. Thus we have $|w - w^{(1)}| \leq s_W/2$. Now in the case $k = 2$, we have by definition of the quantization of the residual error and the property of the rounding operator

$$\left| \left\lfloor \frac{w - w^{(1)}}{s_{R^{(2)}}} \right\rceil - \frac{w - w^{(1)}}{s_{R^{(2)}}} \right| \leq \frac{1}{2} \tag{E.3}$$

where $s_{R^{(2)}}$ is the rescaling factor in the second order residual R^2 computed from $w - w^{(1)}$. The quantized weights are thus given by:

$$\left| w - \sum_{i=1}^{2} w^{(i)} \right| \leq \frac{s_{R^{(2)}}}{2} \tag{E.4}$$

Because the weight distribution is symmetric we know that for any k, $s_{R^{(K)}} = \frac{\max\{w - \sum_{k=1}^{K-1} w^{(k)}\}}{2^{b-1}-1}$ or any other definition of the delta in the full-precision space. Also, by definition we have $\max\{w - \sum_{k=1}^{K-1} w^{(k)}\} \leq s_{R^{(K)}}$. Thus:

$$\left| w - \sum_{k=1}^{K} w^{(k)} \right| \leq \left(\frac{1}{2^{b-1}-1} \right) \frac{s_{R^{(K)}}}{2} \tag{E.5}$$

We conclude by using a trivial induction proof. □

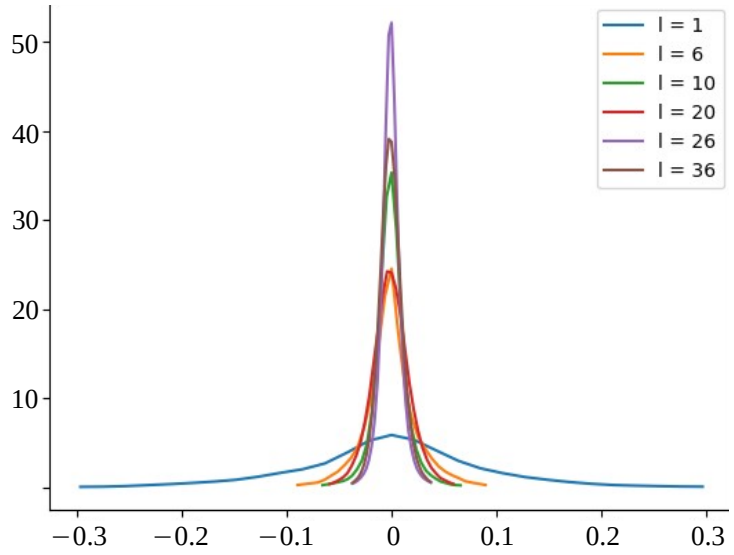


Figure E.1: Distribution of the scalar weight values of different layers of a ResNet 50 trained on ImageNet. We observe that every distribution is symmetric around 0.

As an immediate consequence we have the following corollary which justifies the expansion appellation:

Corollary E.1.2. *Let f be a layer of real-valued weights W with a symmetric distribution and $R^{(k)}$ the k^{th} quantized weight from the corresponding residual error. Then,*

$$\mathbb{E}\left[\left\|f - \sum_{k=1}^{K} f^{(k)}\right\|\right] \geq \mathbb{E}\left[\left\|f - \sum_{k=1}^{K+1} f^{(k)}\right\|\right] \tag{E.6}$$

and $f = \sum_{k=1}^{\infty} f^{(k)}$.

The first inequality results from detailing the induction in the previous proof. Instead of an upper bound on the error over all the scalar values we consider each error and show using the same properties that they go down after each step. $f = \sum_{k=1}^{\infty} f^{(k)}$ is a direct consequence of equation 3.18.

Sparse Expansion Let $N_i^{(k)}$ denotes the L_1 norm of an output channel i of the k-th order residue $R^{(k)}$. The sparse residue is defined as:

$$\left(R_\gamma^{(k)}\right)_i = (R^{(k)})_i \cdot \mathbb{1}_\gamma^{(k)} \tag{E.7}$$

where $\cdot$ is the element-wise multiplication, $\mathbb{1}_\gamma^{(k)} = \mathbb{1}_{\{N_i^{(k)} \geq \tau_\gamma^{(k)}\}}$ and $\tau_\gamma^{(k)}$ is a threshold defined as the γ percentile of $N^{(k)}$. In other words, we remove a proportion γ of channels from residue $R^{(k)}$ that are the least important, as indicated by their norm $N^{(k)}$. Note however that these pruned channels can be encoded in subsequent residuals, *i.e.* $R^{(k')}$, with $k' > k$. The result from Lemma 3.1.3 becomes:

Lemma E.1.3. *Let f be a layer of real-valued weights W with a symmetric distribution. Then we have*

$$\begin{aligned} &\left| w - \left(\sum_{k=1}^{K-1} w^{(k)} + Q^{-1}\left(R_\gamma^{(K)}\right) \right) \right| \\ &\qquad \leq \frac{\left\| N^{(K)} \cdot \mathbb{1}_\gamma^{(K)} \right\|_\infty (s_{R^{(k)}})_i}{(2^{b-1}-1)^K 2} \end{aligned} \tag{E.8}$$

where $\|\|_\infty$ is the infinite norm operator with the convention that $\|0\|_\infty = 1$ and $(s_{R^{(k)}})_i$ denotes the row-wise rescaling factor at order K corresponding to w.

Proof. From equation 3.18, we have:

$$\left| w - \left(\sum_{k=1}^{K-1} w^{(k)} + Q^{-1}\left(R_1^{(K)}\right) \right) \right| \leq \frac{(s_{R^{(K)}})_i}{2} \left(\frac{1}{2^{b-1}-1} \right)^K \tag{E.9}$$

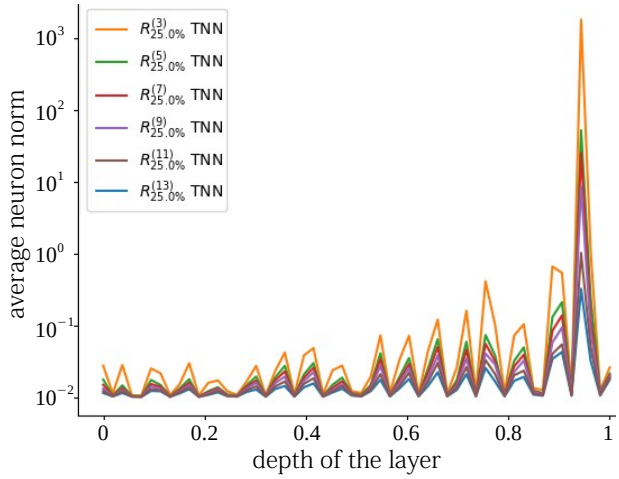


Figure E.2: Comparison of the average norm of the quantization error for each layers of a ResNet 50 trained on ImageNet. We observe the exponential convergence stated in lemma 3.1.3 and E.1.3.

which corresponds to the case where $\gamma^l = 1$. If $\gamma^l < 1$, we have two possibilities for w. First, the coordinate in $N^{(K)}$ associated to is greater than $\tau^{(K)}_{\gamma^l}$ then we fall in the case where $R^{(K)}_{\gamma} = R^{(K)}$ and as such we have the result from equation 3.18 which is stronger than equation E.8. Second, the coordinate in $N^{(K)}$ associated to is lower than $\tau^{(K)}_{\gamma^l}$. Then we have that the difference between the baseline weight w and the slim expansion is bounded by the expansion of lower order and the maximum of the norm $N^{(K)}$ which leads to the result in equation E.8. □

Empirical validation: In lemma 3.1.3 and E.1.3 we stated the exponential convergence to 0 of the approximation error on the weight values. In order to empirically confirm this theoretical result, we quantize a ResNet 50 trained on ImageNet in ternary values for different orders K. As can be seen in Figure E.2, the average error per layer, exponentially converges to 0 which matches our expectations. The figure also confirms the empirical result on the strategies for γ. The higher errors are located on the last layers, thus these layers require more attention.

E.2 Upper Bound Error

Theorem E.2.1. *Let F be a trained L layers sequential DNN. We note σ_l the largest singular value of $W_l - \sum_k R^{(k)}$, i.e. the spectral norm of $W_l - \sum_k R^{(k)}$. Then we have*

$$\begin{aligned} \max_{\|X\|=1} \|F(X) - F(X)^{(K)}\|_\infty &\leq U_{res} \\ U_{res} &= \prod_{l=1}^{L} \left(\sum_{i=1}^{l} \sigma_i u_i^{(K)} + 1 \right) - 1 \end{aligned} \tag{E.10}$$

where $u_l^{(K)} = \left(\frac{1}{2^{b-1}-1}\right)^{K-1} \frac{\left(s_{R^{(K)}}\right)_i}{2}$ from equation 3.18.

Proof. Let's consider $L = 2$, and $F : X \mapsto B\sigma(Ax)$. For any X in the domain of F such that $\|X\| = 1$, we have

$$\|F(X)\|_2 \leq \sigma_B + \sigma_A + \sigma_B \sigma_A \tag{E.11}$$

where σ_B is the largest singular value of B and σ_A is the largest singular value of A. Following the definition of the 2-norm and ∞-norm, we get that

$$\sigma_{A-A^{(K)}} \leq \sigma_A u_A^{(K)} \tag{E.12}$$

where $\sigma_{A-A^{(K)}}$ is the largest singular value of the residual error of order K, $A - A^{(K)}$ and $u_A^{(K)}$ is derived from equation 3.18. Consequently, we get

$$\|F(X) - F^{(K)}(X)\|_2 \leq \sigma_B u_B^{(K)} + \sigma_A u_A^{(K)} + \sigma_B u_B^{(K)} \sigma_A u_A^{(K)} \tag{E.13}$$

□

Sparse Expansion

Theorem E.2.2. *Let F be a trained L layers sequential DNN. We note σ_l the largest singular value of $W_l - \sum_k R^{(k)}$, i.e. the spectral norm of $W_l - \sum_k R^{(k)}$. Then we have*

$$\begin{aligned} \max_{\|X\|=1} \|F(X) - F(X)^{(K)}\|_\infty &\leq U_{sparse} \\ U_{sparse} &= \prod_{l=1}^{L} \left(\sum_{i=1}^{l} \sigma_i u_i^{(K)} + 1 \right) - 1 \end{aligned} \tag{E.14}$$

where $u_l^{(K)} = \frac{\left\|N^{(K)} \cdot \mathbb{1}_\gamma^{(K)}\right\|_\infty \left(s_{R^{(k)}}\right)_i}{(2^{b-1}-1)^K 2}$ *from equation E.8.*

This results is directly derived from Theorem E.2.1. This result can be extended to more sophisticated architectures. To do so we simply need to address specific attributes such as skip connections, concatenations and other activation functions.

Skip Connections and Concatenations In the case of skip connections, the graph is split from a starting layer l_1 and split in at least two branches that are added after layer l_2 and l_3. Assuming we can compute the upper bound for each branch (sub-networks) we simply add these sub-errors. In the case of U-nets, where skip connections contain skip connections, we simply perform this process recursively.

A similar approach can be applied to address concatenations. However in this case we keep the largest value instead of adding them.

Self-Attention and Cross-Attention blocks In order to generalize to attention modules, we need to generalize our formula to a product of layers. Let's consider the weight tensors of the keys W_{keys} and queries W_{queries}. Then the attention scores are computed as follows

$$\text{Att}(X) = (W_{\text{keys}} \times X)^T \times (W_{\text{queries}} \times X) \tag{E.15}$$

We want to bound the quantization error on the attention mechanism. However, the process involves the magnitude of the inputs X as we highlight

$$\text{Error}_{\text{Att}}(X) = \left\| (W_{\text{keys}} \times X)^T \times (W_{\text{queries}} \times X) - \left(\left(\sum_k R^{(k)}_{\text{keys}} \right) \times X \right)^T \times \left(\left(\sum_k R^{(k)}_{\text{queries}} \right) \times X \right) \right\| \tag{E.16}$$

If we note σ_{k} and σ_{q} the spectral norms of the residual errors of the keys and queries respectively, then we can simplify the previous formulation

$$\text{Error}_{\text{Att}}(X) = \left\| (\sigma_{\text{k}} \times X)^T (W_{\text{queries}} \times X) + (W_{\text{keys}} \times X)^T (\sigma_{\text{q}} \times X) + (\sigma_{\text{k}} \times X)^T (\sigma_{\text{q}} \times X) \right\| \tag{E.17}$$

In order to measure this influence on the softmax in the worst case scenario, we can simply compare the σ_{k} and σ_{q} to the smallest singular values of W_{queries} and W_{keys}. If we note α_k and α_q the largest singular values of W_{keys} and W_{queries} respectively, then we get

$$\text{Error}_{\text{Att}}(X)\Big|_{\|X\| \leq 1} \leq \sigma_{\text{k}} \alpha_q + \sigma_{\text{q}} \alpha_k + \sigma_{\text{k}} \sigma_{\text{q}} \tag{E.18}$$

If we note $\epsilon = \sigma_{\text{k}} \alpha_q + \sigma_{\text{q}} \alpha_k + \sigma_{\text{k}} \sigma_{\text{q}}$ this upper bound, then the error on the softmax scores becomes

$$\text{Error}_{\text{Softmax}}(X)\Big|_{\|X\| \leq 1} \leq 1 - e^{-2\epsilon} \tag{E.19}$$

Other Activation Functions Although ReLU activations are predominant in modern DNNs, there are still many other widely used activation functions such as SiLU, GeLU or even sigmoid. SiLU and GeLU are bounded by the ReLU on the positive side which is where the highest errors occur. Consequently, the upperbound is invariant to GeLU and SiLU activation functions (although under more assumptions on the support, the upper bound could be tightened for ReLU and should be modified for GeLU and SiLU). On the other hand, for sigmoid activations or similar activations (e.g. tanh), the upper bound becomes an upper bound on X in the domain of F instead of X on the unit circle.

E.3 Sparse Expansion Outperforms Standard Expansion

Lemma E.3.1. *Let f be a layer of real-valued weights W with a symmetric distribution. Then, for $K' < K$ two integers, we have:*

$$Err\left(R^{(1)} + \sum_{k=2}^{K'} R_{\gamma_1}^{(k)}\right) \geq Err\left(R^{(1)} + \sum_{k=2}^{K} R_{\gamma_2}^{(k)}\right) \tag{E.20}$$

where Err is the quantization error (i.e. the absolute difference between the quantized and original weights, as in Equation 3.18) and $K' \times \gamma_1 = K \times \gamma_2 = \beta$.

Proof. Let's assume the layers outputs two channels. Then, we have $\gamma_1 = 1$ and $\gamma_2 = 0.5$. We simply need to prove the result for $k_1 = 2$ and $k_2 = 1$ as the result will extend naturally from this case. The idea of the proof consists in showing that using lower β values enables more possibilities of expansions which may lead to better performance. Let's note $(W)_1$ and $(W)_2$ the weights corresponding to the computation of the first and second output channels respectively. Using $\gamma_1 = 1$, the second order expansion correspond to either quantizing $(W)_1$ or $(W)_2$. Assume $(W)_1$ is chosen for $R_{\gamma_1}^{(2)}$. Then, $R_{\gamma_1}^{(3)}$ will either quantize the error from $(W)_2$ or further quantizes the error from $R_{\gamma_1}^{(2)}$. In the first case we end up with $R^{(1)} + \sum_{i=2}^{k_1} R_{\gamma_1}^{(i)} = R^{(1)} + \sum_{n=2}^{k_2} R_{\gamma_2}^{(i)}$. Otherwise, $\text{Err}\left(R^{(1)} + \sum_{i=2}^{k_1} R_{\gamma_1}^{(i)}\right) > \text{Err}\left(R^{(1)} + \sum_{i=2}^{k_2} R_{\gamma_2}^{(i)}\right)$. □