Ministère des Armées

Master MVA

Neural network compression and inference acceleration : a literature review

September 2019

Author

Samuel Hurault

1 Introduction

In the past few years Convolutional Neural Netowrks (CNN) have been build to increase the detection / classificaton accuracy, without paying attention to model complexity and computational efficiency. However, such performances come at the cost of high computational, energy and memory requirements.

With floating-point precision networks, the computational complexity can be expressed with FLOP (number of floating point operations necessary for one evaluation of the network), memory requirement and number of parameters used by the model.

We display in Table 1 the comparison of typical CNN models. We also display in Table 2, the time of a the forward and a backward pass on GPU and CPU for one image of the ImageNet dataset (images $3 \times 224 \times 224$). The figures were obtained from [jcjohnson, 2017].

Model	FLOP	Nb parameters (million)	Depth
AlexNet [Krizhevsky et al., 2012]	7.25×10^8	58.3	7
VGG16 [Simonyan and Zisserman, 2014]	1.55×10^{10}	134.2	16
ResNet-50 [He et al., 2015]	3.80×10^9	23.5	50
GoogLeNet [Szegedy et al., 2014]	1.57×10^9	6.0	22

Table 1 – Comparison of CNN models trained on ImageNet. FLOP : number of Floating-Point OPerations. Depth : number of layers in the network.

Model	CPU (ms)	GPU (ms)
Forward	2477	35
Backward	4150	69
Total	6627	104

Table 2 – Comparison of the forward and backpropagation execution times of ResNet50. CPU : Dual Xeon E5-2630 v3. GPU : Pascal Tital X

As shown in Table 2, training these large networks requires large amount of computing power and has only been possible thanks to Graphics Processing Units (GPU). However inference on GPU is efficient in the cloud but often require too much energy to be deployed on energy-constraint mobile systems. In other words, the complexity of very deep networks make them impractical for real-time processing on mobile devices or embedded hardware. It is therefore crucial to develop techniques that reduce model size, lower power consumption, faster inference while maintaining the initial performances.

The motivations for reducing network sizes are fostered by the following observation : when going deeper, a network will be able to capture stronger levels of abstraction. However, it may become over-parametrized and create significant redundancy. In [Qin et al., 2018], the authors visualize layers of the VGG-16 model trained on the CIFAR-10 dataset and find similar patterns inside each layers. This results in a waste of both computation and memory usage. The main idea for accelerating training and inference is to remove or to exploit this redundancy. We will show that it is even possible to improve performances with a smaller network. This observation was already given in [LeCun et al., 1990] : "a 'simple' network is more likely to generalize correctly than a more complex network, because it presumably has extracted the essence of the data and remove redundancy from it".

In this work we study three different network reduction methods : network pruning, network quantization and knowlegde distillation. Network quantization is self-explanatory. Network pruning consists in deleting unnecessary connections in a neural network. Knowlegde distillation consists in training a small network using knowlegde transfer from a combersome network.

The rest of the report is organized as follow. In section 2 we introduce briefly concepts, networks and notations used in the following development. After that we respectively treat in sections 3, 4, 5 network pruning, network quantization and knowledge distillation.

2 Background

2.1 Neural Networks

We first need to introduce the different neural networks structures that we are going to treat in this document : feed-forward, convolutional and recurrent networks.

2.1.1 Feed-forward Neural Network

A feed-forward network is the simplest type of artificial neural networks. As represented Figure 1, it is composed of successive fully-connected (FC) layers : an input layer, different hidden layers and an output layer. Fully-connected layers (FC) , or dense layers, calculate a weighted sum of their input. We note N^l the number of neurons at layer l. The weights between the two layers l and $l-1$ form a $N^{l-1} \times b^l$ matrix W^l and the bias b^l a vector of size N^l . Given an input $x^{l-1} \in \mathbb{R}^{N^{l-1}}$, the output is

$$z^l = W^l.x^{l-1} + b^l \tag{1}$$

2.1.2 Convolutional Neural Network

In the past few years, Convolutional Neural Networks (CNN) have achieved state of the art results for different tasks in different areas, such as Computer Vision , Speech Recognition, or Natural Language Processing. We are going to introduce here 2D convolutional layers but it can be extended to other dimensions (typically 1D or 3D).

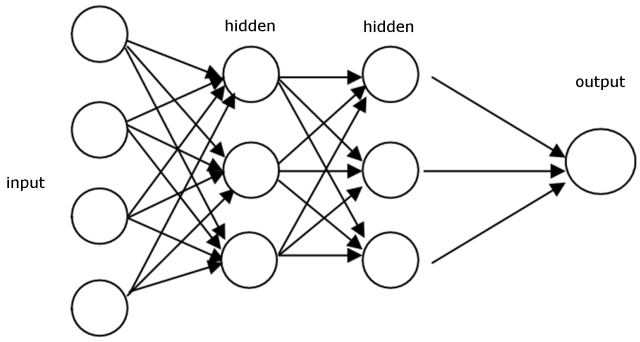

Figure 1 – Feed-forward neural network

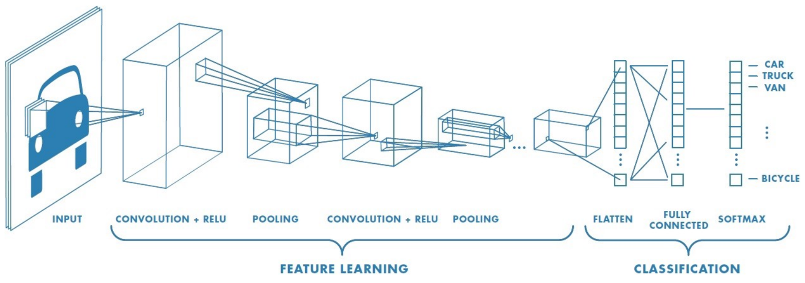

Figure 2 – 2D convolutional network

A CNN is represented Figure 2. It consists in convolutional layers and fully-connected layers. In typical state-of the art CNNs fully-connected layers contain over 90% of the parameters while convolution layers contain more than 90% of the required arithmetic operations.

One convolutional layer (CONV) is represented in detail Figure 3. Such a CONV layer l is composed of a set of N^l kernels $(W_k^l)_{k\in 0,..,N^l}$ of dimension $C^{l-1}\times h^l \times w^l$ regrouping weights, where C^{l-1} is the number of input channels. In short, contrary to FC layers, weights are shared across different neurons called filters. Each filter computes a convolution with a set of C^{l-1} input feature map $\hat{z}^{l-1}$. The input feature maps can either be the input of the network or the output from another convolutional layer. We note z_k^l the output of this convolutional layers. z_k^l feeds activation and pooling layer (see after) to obtain $\hat{z}_k^l$ which can be the input of a following convolutional layer. The output feature maps z_k^l are obtained by :

$$for\ k \in \{1, ..., N^l\}, \quad z_k^l = \hat{z}^{l-1} * W_k^l + b_k^l \tag{2}$$

We note $H^l \times W^l$ the dimensions of z_k^l and $\hat{H}^l \times \hat{W}^l$ the dimensions of $\hat{z}_k^l$.

A typical convolutional layer is followed by activation function and pooling. The ensemble CONV layer + Activation + Pooling is called a convolutional block.

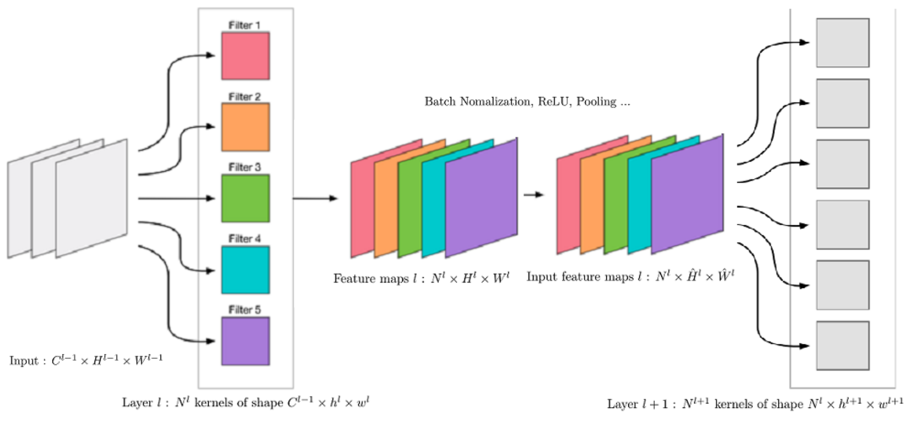
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Batch Nomalization, ReLU, Pooling ...
Feature maps l : $N^l \times H^l \times W^l$
Input feature maps l : $N^l \times \hat{H}^l \times \hat{W}^l$
Input : $C^{l-1} \times H^{l-1} \times W^{l-1}$
Layer l : N^l kernels of shape $C^{l-1} \times h^l \times w^l$
Layer $l+1$: N^{l+1} kernels of shape $N^l \times h^{l+1} \times w^{l+1}$

FIGURE 3 – 2D convolutional layer

To enable the network to capture non-linear relations, the output of FC and CONV layers can feed non-linear activation functions f. Nearly all state-of-the art CNNs use Rectified Linear Units (ReLU) $f(x) = max(0, x)$.

Pooling layers are used between successive convolutional layers in the network. They are sub-sampling functions used to reduce the spacial dimension of feature maps and encode translation invariance.

The last layer of CNNs is the output layer, it generally consists in a FC layer with a particular activation function. For classification tasks the softmax activation funtion is commonly used.

2.2 Recurrent Neural Network and LSTM

Recurrent Neural Network (RNN) is a class of neural network that maintain internal hidden states to model a dynamic behavior. They can take into consideration temporal context thanks to a recurrent connection within he hidden layer. A classic RNN unit is shown at top of Figure 4. At cell l, the output h_{t-1} of cell $l-1$ is multiplied by the input x_l to feed a tanh activation and form the output h_t.

A long short-term memory unit (LSTM) is a special kind of RNN unit represented at buttom of Figure 4. It adds of a memory, input and output gates that control the information flow. As the architecture of LSTM are very specific, we are often going to treat these neworks separately from CNN. LSTM is widely used in speech recognition.

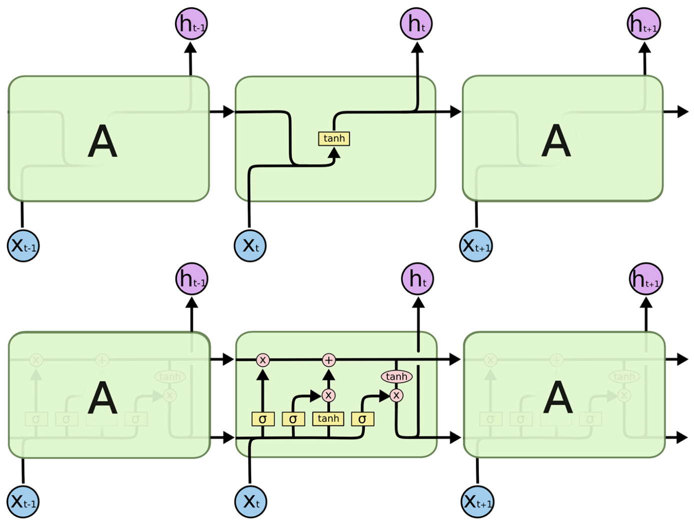

FIGURE 4 – (top) RNN unit and (buttom) LSTM unit

2.3 Training and inference

Finding the optimal parameters W_0 in the parameter space $\mathcal{W}$ is done by minimizing a loss function $\mathcal{L}(W) = \sum_{i=1}^{N} \mathcal{L}(g(X_i), Y_i|W)$ on a training dataset composed of N data points. g denotes the action of the network on a input X_i and Y_i a ground truth label.

$$W_0 = \underset{W \in \mathcal{W}}{\arg\min}\, \mathcal{L}(W) \tag{3}$$

The choice of the individual loss $\mathcal{L}$ depends on the task. For classification problems, cross-entropy is widely used.

A common optimization algorithm is Stochastic Gradient Descent (SGD). Iteratively, weights are updated with the following scheme :

$$W := W - \eta \nabla \mathcal{L}(W) \tag{4}$$

where η is the learning rate.

In practice a learning epoch is composed of successive forward and backward propagations.
— The forward pass in the network correspond simply to the evaluation $g(X_i)$. It is done on a batch of images randomly chosen to compute the loss on a batch of images.
— The backward pass computes the derivative of this loss w.r.t each parameter W $\mathcal{L}(W)$.

Training consists in done during several epochs. The inference corresponds to the evaluation of trained network on set of test dataset.

2.4 Networks and Datasets

ImageNet

The ImageNet ILSVRC [Russakovsky et al., 2015] competition is a large scale image classification and detection challenge which has been held annualy since 2015. It provides the ImageNet data set consisting of over 1 million images and 1000 categories. A strandart performance measure for deep CNNs is their classification accuracy on the ILSRVC 2012 data. Measures consists in the top-1 and top-5 accuracy on the validation dataset. This competition gave birth to numerous famous state-of-the art CNNs. In this report we focus on four of them : AlexNet [Krizhevsky et al., 2012] , VGG16 [Simonyan and Zisserman, 2014], ResNet [He et al., 2015] and GoogLeNet [Szegedy et al., 2014].

CIFAR

However, ImageNet uses large images ($224 \times 224 \times 3$) and is very complex. A single training can take several days. CIFAR ([Krizhevsky, 2009]) is a smaller dataset also commonly used to benchmark CNN performances. CIFAR-10 contains 10 classes of 6000 images 32×32. CIFAR-100 is like CIFAR-10 with 100 classes.

3 Network pruning

A straightforward approach to reduce a model size and complexity is to remove unnecessary connections. Such a pruning of network connections has to be done in a manner that preserves the original accuracy of the network. Pruning can me measured with different metrics like the pruning ratio : $r = \frac{Nb\ of\ parameters\ in\ pruned\ network}{/Nb\ of\ parameters\ in\ original\ network}$, number of FLOP or computational time of an operation. Computational time depends on the hardware used but it is important to keep this measure as the sparcity created by pruning can be structured to optimize hardware computations.

Pruning can be realized one-shot but current state-of-the-art pruning follows an iterative and greedy scheme ([Molchanov et al., 2016], [Han et al., 2015], [Anwar et al., 2015]) :
- Train the original large network.
- Evaluate and remove the least important elements.
- Fine-tune by retraining the remaining network.
- Repeat until stopping criterion.

The stopping criterion can be a target value in pruning ratio, FLOPs or network performance. Such an iterative process is said to be useful because the final network inherits knowledge from the bigger and cumbersome network.

3.1 CNN pruning

In deep CNNs, convolutional layers impact computational complexity while FC layers are responsible for memory overloads. Depending on the objective, shortcut memory or runtime, the pruning method can be different.

Three important choices have to be done :

— A pruning can be realised at different **levels** (or granularities). It is possible to prune individual scalar weights (intra-kernel pruning), filters (kernel pruning) or entire channels (i-e the sets of kernels reponsible for a feature map).

— The **metric** used to evatuate the importance of these structures.

— At which **scale** do we make this comparison : in a single layer or in the entire network.

[Molchanov et al., 2016] formulates the pruning problem as follows : Let say that W represent the ensemble of individual weights, the ensemble of kernels, or the ensemble of the ensemble of channels. Let us call W_0 the pretrained parameters and W_p the subset of parameters kept after pruning. Pruning can be viewed as a combinatorial optimization :

$$\min_{W_p} |\mathcal{L}(W_p) - \mathcal{L}(W)| \quad with \quad ||W_p||_0 \leq B \tag{5}$$

where the l_0 norm bounds the number of non-zero parameters by B in W_p. This combinatorial problem requires $2^{|W|}$ evaluations of the cost function to be solved exactly which is unefeasable in practice. It is thus preferable to use the previously introduced iterative greedy approach.

3.1.1 Weight level pruning

One major branch of network pruning is individual weight pruning. Here W only represents the ensemble of individual weights of the network. It was developed as soon as early development of neural networks with Optical Brain Damage (OBD) [LeCun et al., 1990] and Optical Brain Surgeon (OBS) [Hassibi et al., 1993]. These two methods use second-order derivative of the objective function to select parameters for deletion (see 3.1.2). More recently, in [Han et al., 2015], the authors propose to simply use weight magnitude. They follow the iterative process descibed before. At each iteration, all connections with absolute weights below a threshold are removed from the network. They treat equally FC and convolutional layers. The pruning threshold is chosen as a quality parameter multiplied by the standard deviation of a layer's weights. The process can be repeated until $||W_p||_0 \leq B$ or until a drop in accuracy performance is noticed.

An interesting result is that CONV layers are more sensitive to pruning than FC layers : without loss of accuracy on AlexNet, it is possible to prune almost 75% of a FC layer but only around 60% of a CONV layer.

This method is very effective for network compression but this intra-kernel sparcity may not translate to faster inference if not computed on a specialized hardware. The same author introduce in [Han et al., 2016b] a dedicated engine that accelates the resulting sparce matrix-vector multiplication.

3.1.2 CONV layers structured pruning

Instead of using specific engines, convolutional layer pruning can accelerate inference large if it is structured. Instead of considering individual weights, it is possible to treat other structures :

- **At kernel level** ([Anwar et al., 2015], [Molchanov et al., 2016]) : entire filters are considered $W = \{W_k^l\}_{k,l}$. We observe in figure 5 an example of such a pruning. However, in practice, this kind of sparcity does not accelerate inference because it only creates sparcity in matrix products.

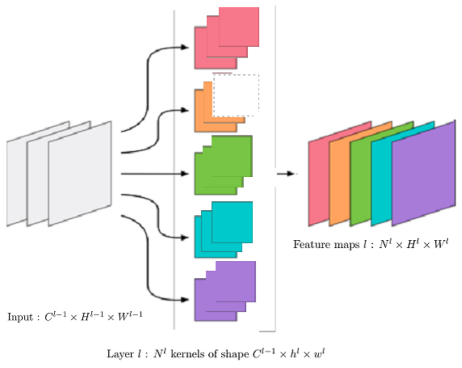
Feature maps l : $N^l \times H^l \times W^l$
Input : $C^{l-1} \times H^{l-1} \times W^{l-1}$
Layer l : N^l kernels of shape $C^{l-1} \times h^l \times w^l$

FIGURE 5 – kernel pruning

- **At channel level** ([Anwar et al., 2015], [Li et al., 2016], [Molchanov et al., 2016]) : the algorithm considers entire channels (sets of kernels responsible for a single feature maps). Removing one channel implies the suppression of its corresponding feature map. We observe in figure 6 an example of such a pruning. By removing whole channels, inference time can be reduced significantly.
- **At feature map level** ([Anwar et al., 2015], [Molchanov et al., 2016], [Hu et al., 2016]) : the algorithm considers individual feature maps instead of kernels. It is structurally equivalent to prune the corresponding channel, but the pruning metric now uses activation values and not weight values. Working with feature maps implies the knowledge of the stastistics of the activations. They can be estimated with one epoch forward pass on a set of input images.

As [Qin et al., 2018], we also distinguish :

- "Automatic architecture pruning" : the pruning algorithm globally compares all the

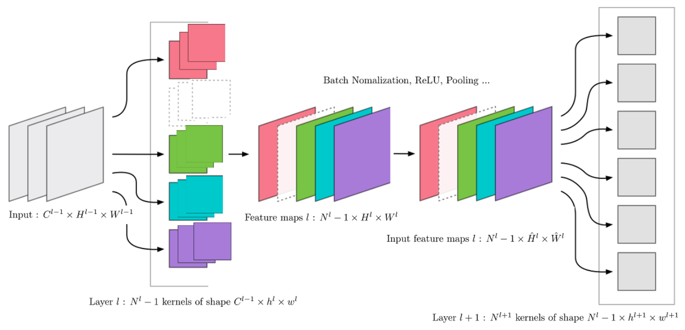
Batch Nomalization, ReLU, Pooling ...
Input : $C^{l-1} \times H^{l-1} \times W^{l-1}$
Feature maps l : $N^l - 1 \times H^l \times W^l$
Input feature maps l : $N^l - 1 \times \hat{H}^l \times \hat{W}^l$
Layer l : $N^l - 1$ kernels of shape $C^{l-1} \times h^l \times w^l$
Layer $l+1$: N^{l+1} kernels of shape $N^l - 1 \times h^{l+1} \times w^{l+1}$

FIGURE 6 – Feature map or channel pruning. The second feature map at layer l is pruned. The ongoing corresponding channel is pruned to. It also affect the width of the outgoing kernels.

elements, independantly of the layer they belong to, and finds an optimal architecture for each layer.

— "predefined architecture pruning" : a human predefines the pruned architecture with a layer-by-layer pruning ratio. For example, we may decide to prune 50% of each CONV layers and 70% of each FC layers. The algorithm then works locally, layer by layer.

Automatic architecture pruning

With automatic architecture pruning, we work globally on the network and compare elements independantly of the layer they belong to. As explained before, we have to choose a metric σ to sort elements according to their importance for the network. If we work at the kernel level we note σ_w and if we work at the feature map level σ_a. Once sorted, we remove a predefined percentage of the least important elements.

[Molchanov et al., 2016] compares different pruning metrics. :

— **Weight magnitude** : $\sigma_w(W_k^l) = ||W_k^l||_2$ The simplest criteria is to use the weights magnitude. Every kernel is represented by the $l2$ (or $l1$) norm of its weight tensor. [Li et al., 2016] does not observe noticeable difference between the $l2$ and $l1$ norms.

— **Activation magnitude** : $\sigma_a(z_k^l) = ||z_k^l||_2$ The same criteria can be used on the activations resulting from the kernel convolution (before activation function and pooling).

[Molchanov et al., 2016] uses these minimum weight and activation pruning methods with a global comparison between layers. Desappointly, they find that minimum weight pruning gives as good performances as random weight pruning. Indeed, for these methods, it makes more sense to make layerwise comparisons. We will develop this case in 3.1.2.

— **First order Taylor expension** : This criterion relies on the first order approximation of the change in loss caused by removing an element of the network. [Molchanov et al., 2016] applies the criterion at the feature map level, but it can be applied at the kernel level too. We denote h to be a kernel or a feature map and $\mathcal{L}(h = 0)$ the post-training cost function when h is set to 0. The choice of the metric is :

$$\sigma(h) = \frac{1}{M}|\Delta\mathcal{L}(h)|$$

where

$$\Delta\mathcal{L}(h) = \mathcal{L}(h = 0) - \mathcal{L}(h)$$

and M is the length of h.
With the first order Taylor expension, we have :

$$\Delta\mathcal{L}(h) \approx \nabla_h\mathcal{L}, h$$

Therefore, $\sigma(h) = \frac{1}{M}|\nabla_h\mathcal{L}, h|$.

— **Optimal Brain Damage (OBD)** : This criterion is the equivalent with second order approximation. The original article [LeCun et al., 1990] applies it for weight-level pruning but it can be applied, like in [Molchanov et al., 2016], at kernel/feature-map level. The second order derivative is used because, after convergence of the model, it can be assumed that the first order derivative of the cost function w.r.t the parameters is zero.

$$\Delta\mathcal{L}(h) \approx \nabla_h\mathcal{L}, h + \frac{1}{2}h^T Hh \approx \frac{1}{2}h^T Hh$$

with the Hessian $H = (\frac{\delta^2\mathcal{L}}{\delta h_i \delta h_j})_{i,j}$ [LeCun et al., 1990] proposes to approximate the Hessian by its diagonal. Therefore, $\sigma(h) = \frac{1}{2M}\nabla^2_h\mathcal{L}, h^2$ where M is the length of h. Note that OBD [LeCun et al., 1990] approximates the sign difference in loss while [Molchanov et al., 2016] with first order Taylor expension approximates the absolute difference in loss. Indeed, with $|y| = |\nabla_h\mathcal{L}, h|$ as pruning criterion, contrary to y, $|y|$ is non-zero in expectation.

— **Average percentage of zeros (APOZ)** ([Hu et al., 2016]) : ReLU activation function imposes sparcity during inference. It is proposed to use as a criteria the average percentage of zeros in a feature map after going through ReLU : $\sigma_a(\hat{z}^l_k) = |\{\hat{z}^l_k > 0\}|$

Pruning can be applied to reduce the number of parameters and/or the number of operations. However, different layers may require a different number of parameters/operations. [Molchanov et al., 2016] proposes to regularize the chosen metric σ by the number of parameters $param(l)$ or the number of FLOPs $FLOP(l)$ of the layer involved. For example :

$$\sigma(W^l_k) = \sigma(h) - \lambda_1 * param(l) - \lambda_2 * FLOP(l)$$

. This normalization enables a more efficent global comparison.

[Molchanov et al., 2016] claims that the optimal criteria is the "first order Taylor expension", closely followed by OBD. However, OBD is slower to compute because of the computation of the second order derivatives. On VGG16 trained on ImageNet, they succeed in reducing the number of GFLOPs from 30.96 to 11.5 with a loss of -2.3% in top-5 validation accuracy. They also find that this "optimal criteria" highly correlates with the ideal "oracle" ranking method which consists of pruning each kernel one-by-one and observing the changes in the cost function.

Predefined architecture pruning

We now do not compare all the filters in the network at once, but compare elements layer-by-layer. We note that the methods described before for automatic architecture target pruning are directly transcriptable to predefined architecture target pruning. We cite here some other works working specifically layer-by-layer.

— **Weigh magnitude** : [Li et al., 2016] uses the magnitude criteria for layer local comparision. Layer-by-layer, filters are sorted by their $l1$ norm and the m percent of filters with smallest norm are pruned. They empically set the target pruning ratio for each layer based on their pruning sensitivity. Setting different threhold for each layers based on pruning sensivity is time-consuming but enables more accurate results. In practice, with VGG16 on CIFAR 10, they prune 50% of layers 1 and 8 to 13, and they do not prune the others to obtain 34% FLOP reduction without loss of accuracy.

— **Activation magnitude** : [Li et al., 2016] also experiments feature map pruning. They compare the previous $l1$-norm filter pruning with feature-map pruning using different criteria to sort feature maps : the mean $\sigma_a(z_k^l) = mean(z_k^l)$, the standart deviation $\sigma_a(z_k^l) = std(z_k^l)$, the $l1$ and $l2$ norms. They find that the activation base pruning with the standart deviation criterion has similar performance to the $l1$-norm filter pruning. However, weight magnitude method is more covenient as it is data free.

— **Thinet** ([Luo et al., 2017]) : This method prunes the channel that has the smallest effect on the next layer's activation values. The idea is that if one channel of $\hat{z}_k^l$ at layer l does not affect the outputs $\{z_k^{l+1}\}$ at layer $l+1$, this channel can be safely removed. Removing $\hat{z}_k^l$ is equivalent to removing the corresponding filter W_k^l. The motivation for evaluating z_k^{l+1} instead of z_k^l is that when removing one filter at layer l, the size of z_k^{l+1} is not affected. Therefore, anayzing the effect on z_k^{l+1} is close to analysing the effect on the overall performance of the network. More precisely, the algorithm selects, at layer l, a subset C^l of channels in all the channels $\{1, ..., N^l\}$ that minimizes MSE between $\sum_k (z_k^{l+1})_{C^l}$ and $\sum_k z_k^{l+1}$ where $(z_k^{l+1})_{C^l}$ was orginally computed with only C^l input channels. In practice, for VGG16 on ImageNet, they choose to prune 50% of the 10 first (out of 13) CONV layers to obtain the same accuracy performances.

When pruning layer-by-layer comes the problem of retraining. Retraining directly after pruning on a layer-by-layer basis like [Han et al., 2015] can be extremely time-consuming. [Luo et al., 2017] chooses to fine-tune with only one or two epochs after each layer pruning. [Li et al., 2016] chooses to prune all layers and retrain the whole remaining network afterwards.

What's more when pruning at channel or feature map levels, pruning layer l affects the structure of layer $l+1$, as stated in [Li et al., 2016], it is possible to consider two strategies for layer-wise selection :

— Determine which channel to prune at layer l independantly of the pruning results of channel $l+1$.
— Account for the filters removed from from pruning the previous layer.

The first method is easier to implement but second method is more accurate and results in better performances especially when many filters are pruned.

3.1.3 Iterative pruning limits

[Qin et al., 2018] and [Liu et al., 2018] further analyses the previous iterative ranking pruning method.

[Qin et al., 2018] focuses on kernel magnitude pruning and visualize the network filters during the iterative process. They observe that, among the filters with high magnitude that are kept, it remains functionality repetitions. Therefore we do not completely get rid of redundancy. Moreover, among filters with small magnitude that are going to be removed by pruning, there exist very distinct features that contribute to feature diversity. Thus magnitude-based pruning method inevitably induces accuracy drops. Previous cited works use retraining to compensate for this accuracy drop. However, [Qin et al., 2018] shows that during this retraining phase, numerous filters' functionalities are changing : the retraining process reconstructs the network rather than fine-tuning the initial filter functionalities.

In order to bypass this problem, during the iterative pruning and retraining process, [Zhu and Gupta, 2017] proposes to prune the network rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights being pruned each time as there are fewer and fewer weights remaining in the network.

[Liu et al., 2018] adds the following observation : in some cases, given a pruned architecture, training the model from scratch with random weight initialization gives comparable or even better performance than fine-tuning. However, they show that the pruned architecture found with the previous method often brings benefits compared to the original architecture. Therefore network pruning can be seen as an optimal architecture search method. This observation is contrary to the ones given in [Li et al., 2016] or [Luo et al., 2017]. They argue on the contrary that training a pruned architecture from scratch performs worse than retraining.

[Qin et al., 2018] (and more recently by the same authors [?]) proposes instead a filter functionality-oriented kernel pruning with the following method :

— Visualize filters as 2D images.

- Group each filter applying K-means analysis on the Euclidean distance between visualized filter. They obtain groups of kernels with similar functionalities.
- In each cluster, sort the filters h_i according to a given criterion.
- For each cluster, determine a relative pruning rate and prune filters.

They show that with this method filters functionalities do not change during fine-tuning.

3.2 RNN and LSTM Pruning

Just like CNNs, in order to achieve higher prediction accuracy, RNN and LSTM recently got larger and larger. Structured pruning methods described before depend critically on the structure of the convolutional layers and are not extensible to recurrent architectures. It is however possible to apply individual scalar weight pruning. In [Han et al., 2016a], the same authors from [Han et al., 2015] use their weight magnitude pruning described before : at each iteration the weights with smaller absolute values are pruned. They experiment on the TIMIT dataset [Garofolo et al., 1993] and succeed in pruning 90% of the parameters.

4 Network Quantization

The previous section enables to select an optimal set of weights. To compress further we can represent these weights more efficiently thanks to quantization. Furthermore, we can also accelerate the calculations thanks to integer arithmetic. Neural network quantization is another abundant field of research. Quantized neural networks represent weights, activation or gradients with a smaller number of bits than the typical 32 bits floating points. This enables to shrink the memory requirements, and faster the inference and training processes. Recent research in this field has enabled hard quantized network (with even 1 or 2 bits) to reach the same accuracy as its full-precision counterpart.

4.1 Lower bitwidth representation

Floating point format

Floating points are commonly used to represent real values. They consist in a sign, an exponent, and a mantissa, all of them represented by integer values. The exponent gives the floating point formats wide ranges, and the mantissa a good precision. The real value is computed with this formula :

$$value = (-1)^{sign} \times (1 + \frac{mantissa}{2^{23}}) \times 2^{exponent-127} \tag{6}$$

Fixed point format

The fixed-point format consists in a global shared scaling factor and a precision parameter.

It can be represented by two integer parts $[IL, FL]$ where IL and FL denote the integer and fractional lengths of the number. The number of integer bits IL plus the number of fractional bits FL yields the total number of bits $n = IL + FL$ used to represent the number. This format limits the representable range to $[-2^{IL-1}, 2^{IL-1} - 2^{-FL}]$ (one bit is kept to represent the sign) and the precision to 2^{-FL}. IL controls the range and FL the precision. With such a representation, quantizing a group of values to fixed point format implies the choice of the parameters IL,FL.

In the strict fixed-point format IL is supposed to be shared by every quantized variable. For neural-networks, it is more convenient to use dynamic fixed point format. With this format, a few grouped variables (typically the weights in a same layer) share a common IL.

It is more common to find in the literature quantization with full precision scaling factors. Given a tensor x to quantize, a full-precision scale factor S_x is used to adapt the dynamic range of x. Then, the values are quantized as integers in the range range $[-2^{n-1}, 2^{n-1}-1]$. The global range is now controlled by the scaling factor S_x. The total bitwidth n controls the precision. This procedure can be seen as a full-precision scaling followed by a fixed point representation with $IL = n$ and $FL = 0$. The multiplication of two tensors x and y then implies multiplications between n-bit integers (which can be much faster) and only one full-precision multiplication.

4.2 Quantization methods

We are going to introduce here some of the main methods use in the literature for quantizing a set of weights or activations. Let's call x the floating point vector to quantize. In practice x can represent any weight or activation group. We want to convert the elements of x to n bits fixed-point integers.

We use the following general formulation :

$$x_q = sc^{-1}(Q(sc(clip(x) - Z)) + Z) \tag{7}$$

Where sc is a certain inversible scaling function, $clip$ a clipping function, Q the actual quantization function and Z the zero-point. The zero-point Z is the quantized value of the real value 0. We first suppose $Z = 0$.

$$x_q = sc^{-1}(Q(sc(clip(x)))) \tag{8}$$

The scaling and clipping functions control the range of the quantization process and Q controls its precision.

Rounding methods

Rounding is the simplest and most commom way to quantize a floating-point to a fixed-point integer. With this uniform quantization, the resolution s fixes the number of quantization levels equally spaces in a range R.

For each individual scalar x_i in the vector x :

$$Q(x_i) = \frac{\lfloor x_i s \rfloor + \epsilon_i}{s} \tag{9}$$

Given $k_i = x_i s - \lfloor x_i s \rfloor$ the distance between the original point x_i and its closest smaller quantization point, we can deduce a deterministic and a stochastic rounding :

— Deteministic : $\epsilon_i = \begin{cases} 1 & \text{if } k_i \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$

— Stochastic : $\epsilon_i = Bernouilli(k_i)$ so that the probability of rounding x_i to $\hat{x}_i$ is proportional to the proximity between x_i and $\hat{x}_i$. The stochastic uniform rounding is an unbiased estimator as $E[Q(x)] = x$.

Note that we will be interested later in the intermediate integer value $sQ(x_i)$ in order to compute operations only between integers.

To determine the scaling and clipping functions, we have to consider the two fixed-point representation cases introduced before :

— **strict fixed-point format** $[IL, FL]$:
Let's remind that with this formulation, a fixed-point number is represented with IL bits for its integer part and FL bits for its fractional part and $n = IL + FL$. The resolution is $s = 2^{-FL}$ and the range $R = [-2^{IL-1}, 2^{IL-1} - 2^{-FL}] = [R0, R1]$. As the range is already defined by the parameter IL, there is no need for scaling and $sc = id$. The clipping function $clip$ is simply :

$$clip(x) = \begin{cases} R0 & \text{if } x \leq R0 \\ R1 & \text{if } x \geq R1 \\ x & \text{otherwise} \end{cases} \tag{10}$$

— **floating-point scaling factor** :
With this formulation, the values in the vector x share a common scaling factor S_x that is an arbitrary floating point number : $sc(x) = \frac{x}{S_x}$. S_x and the clipping functions are then defined joinlty according to the distribution of the values in x. The resolution is $s = 2^{-(n-1)} - 1$ We first define the clipping function $clip$ to restrict the values to an interval $[a_x, b_x]$. Then we set $S_x = b_x - a_x$ so that $sc(clip(x_i)) \in [-1, 1]$. The rest of the work is to determine an optimal clipping method.

All in all, with linear rounding, we have :

$$x_q = S_x Q(clip(x)\frac{s}{S_x}) = \frac{S_x}{s}[clip(x)\frac{s}{S_x}] = \hat{S}_x[clip(x)\frac{1}{\hat{S}_x}] \tag{11}$$

where $[x] = sQ(x)$ is an integer and $\hat{S}_x = \frac{S_x}{s}$ can be a floating point.

Zero-point

It is possible tu use two different rounding modes : asymmetric and symmetric.

— The symmetric mode is the mode previously described with $Z = 0$. The quantization range is symmetric w.r.t zero.

— In asymmetric mode, we introduce a zero-point (also called quantization offset) to align the quantization range to the initial floating-points range. It enables to utilize fully the quantization range. For example, after a ReLU activation function, the values are positive. Quantizing in symmetric mode results in losing one bite.

Clipping methods

Clipping enables to make the quantization grid narrower to fit the distribution of the values. In falls back to threholding the outliers of the distribution. We assume that the distribution is symetric and introduce a threshold T such that :

$$clip(x) = \begin{cases} -T & \text{if } x \leq -T \\ T & \text{if } x \geq T \\ x & \text{otherwise} \end{cases} \tag{12}$$

The literature provides many techniques to determine the clipping threshold T ([Banner et al., 2019],[Zhao et
We will introduce them later.

Like pruning, depending on the objective, quantization can be realized after training, with or without fine-tuning, or alongside training.

4.3 Post-training quantization

Post-traning quantization is useful to compress a network and obtain a fast and energy-efficient inference. Suppose we have a pretrained CNN model in full floating point precision. We want to quantize the floating-points weights and/or activations into low-precision. As quantization often results in a loss of accuracy, the majority of the literature involves retraining/fine-tuning after quantization. However, there are world scenarios where retraining is not possible, for instance if we don't have the training data. For such scenarios, it can be useful to have access to a platform like [Migacz, 2017] that could quantize any pretrained neural network into low-precision for inference.

4.3.1 Weight quantization : network compression

The simplest trick is to quantize only the weights of the pretrained model. Quantizing the set of weights falls back to clustering them to shared values. Such a method enables efficient compression because it is possible to store in memory only the indexes and a codebook. Given a certain number of target bits n, the codebook stores the 2^n cluster values. The indexes assigns to each weight position an integer between 1 and 2^n. For an orignal network with N_C connections, orginally coded in b bits and with weights quantized to n bits, the compression ratio is given by :

$$r = \frac{bN_C}{nN_C + 2^n b} = \frac{b}{n + b\frac{2^n}{N_C}} \tag{13}$$

The term $2^n b$ is responsible for the codebook storage. If the number of clusters 2^n is negligeable compared to the total number of parameters in the network $2^n << N_C$, the codebook storage is negligeable and :

$$r \approx \frac{b}{n} \tag{14}$$

As shown in 7, if we choose to quantize only the weights, inference is kept in full precision and the arithmeric remains unchanged.

Vector quantization methods

All the rounding methods described in 4.2 apply to weight quantization. However with this methods, the objective was to assign fixed-point integers to floating point weight. With weight-only quantization, this is not necessary as all the operations are realised in full-precision. Given n bits, we want to find 2^n optimal floating-point cluster values. This can be done ([Gong et al., 2014], [Han et al., 2016c], [Choi et al., 2016]) by one-dimensional k-means.

[Gong et al., 2014] and [Han et al., 2016c] choose to not share weights across layers. At layer l, all the individual scalar weights $w \in W^l$ are partitioned into k^l clusters $C^l = \{C_1^l, ..., C_k^l\}$. It is done by k-means minimization :

$$\arg\min_{C^l} \sum_{k^l} \sum_{w \in C_i^l} |w - c_i^l|^2 \text{ where } c_i^l = \frac{1}{|C_i^l|} \sum_{w \in C_i^l} w \tag{15}$$

[Han et al., 2016c] studies 3 different cluster value initializations : random (initialization with k^l input points) , density-based or linear (uniform in the range of W^l values) and found that the uniform initialization is more efficient. Their explanation is that this initialization helps maintaining large cluster values as weights distribution is often bell-shaped around 0.

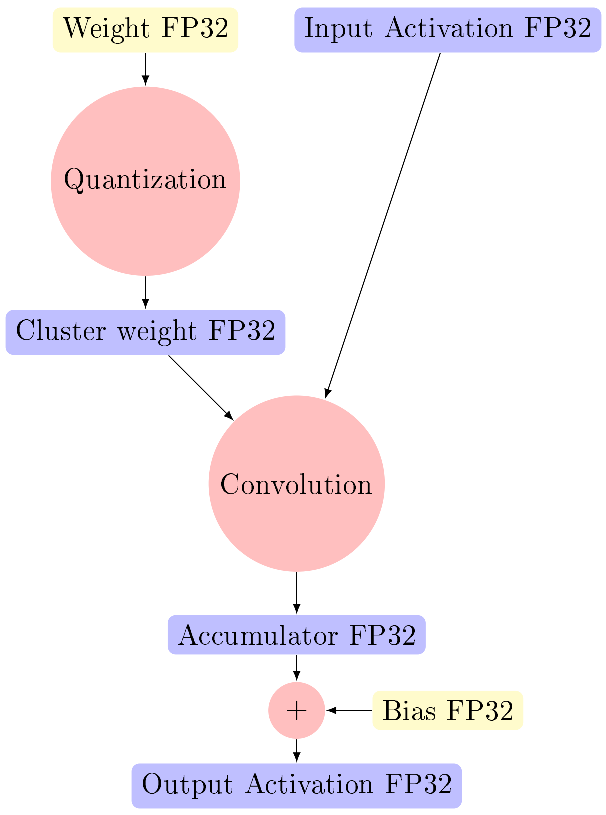

Figure 7 – Weight-only quantized inference. Full-precision is considered to be FP32.

[Choi et al., 2016] proposes to quantize all the network at once rather than layer-by-layer. Furthermore, in order to minimize the quantization impact on the performance loss, they develop a Hessian weighted k-means clustering. All the network scalar weights $W = \{w_j\}_j$ are partitioned into k clusters $C = \{C_1, ..., C_k\}$.

$$\arg\min_{C^l} \sum_k \sum_{w_j \in C_i} h_{jj}|w_j - c_i|^2 \text{ where } c_i = \frac{\sum_{w_j \in C_i} h_{jj} w_j}{\sum_{w_j \in C_i} h_{jj}} \tag{16}$$

Where $h_{jj} = \frac{\delta^2 C}{\delta w_j^2}$ is the j-th diagonal element of the Hessian $H = (\frac{\delta^2 C}{\delta w_i \delta w_j})_{i,j}$. This choice is motivated by the second order Taylor expension of the loss function with respect to $\delta W = W_q - W$. W_q being the ensemble of quantized weights. Assuming a local minima reached during training by W :

$$\delta C \approx \frac{1}{2} \delta W^T H \delta W \tag{17}$$

With the approximation of the Hessian by its diagonal :

$$\delta C \approx \frac{1}{2} \sum_j h_{jj} \delta w_j^2 \tag{18}$$

Therefore, they decide to give a larger clustering penalty for a parameter when its second-order derivative is larger as it will theoretically limit the impact on the loss function.

Hessian-weighted k-means handles the different impacts of quantization errors on the overall performance across and within layers. It can therefore by employed for quantization of all network at once. Despite the additional cost brought by the computation of the second order derivatives, this method avoids a costly optimization realized by [Han et al., 2016c] of the number of bits allocated for each layer. [Choi et al., 2016] shows that Hessian-weighted k-means outperforms global and layer-by-layer k-means clustering. Without fine-tuning, it achieves a lossless compression of ResNet-32 on ImageNet with 5 bits when k-means needs 6 bits.

Huffman coding

Huffman coding [Huffman, 1952] is an optimal variable lenght coding method commonly used for lossless data compression. With variable-length coding, instead of giving as input a fixed number of bits n, [Choi et al., 2016] adds a constraint on the compression ratio $r > r_{min}$ to the quantization problem. [Han et al., 2016c] uses Huffman coding without controlling the compression ratio nor the number of individual bits. The entropy $H = -\sum_i p_i log_2 p_i$ (in bits) is the weighted sum of the information content of each symbol. In our case the symbols are the previously obtained shared weights, H is the entropy of the network, p_i the proportion of the shared weight i $p_i = |C_i|/N_C$. Variable-lenght coding means that every shared weight i is going to be coded with a certain number of bits n_i. The average code word length is then :

$$\bar{n} = \frac{1}{N}\sum_{i=1}^{k}|C_i|n_i$$

The compression ratio is still :

$$r \approx \frac{b}{\bar{n}} \tag{19}$$

Entropy also represents the smallest average number of bits $\bar{n}$ to possibly represent the data. This bound is closely achieved by Huffman coding :

$$\bar{n} \approx H = -\sum_i p_i log_2 p_i \tag{20}$$

Finally :

$$r \approx \frac{b}{H} \tag{21}$$

Therefore, the final compression ratio constraint $r > r_{min}$ becomes an entropy constraint $H < \frac{b}{r_{min}}$. The overall quantization problem with this additional constraint is the classical Entropy Constraint Scalar Quantization (ESCQ). [Choi et al., 2016] uses an iterative algorithm similar to weighted k-means. With this algorithm they manage, without retraining, a lossless compression of ResNet-32 with an average code-word length of less than 3 bits.

Power-of-two quantization

[Zhou et al., 2017] proposes to quantize CNN weights to either power-of-two of zero values. The advantage is that the original floating-point multiplications are replaced with binary bit shift operations. The target quantization map M_l is defined layer-by-layer by $M_l = \{\pm 2^{n_1}, ..., \pm 2^{n_2}, 0\}$. Weighs are quantized as previously. First, weights with absolute value smaller than 2^{n_2} are set to zero. Then, weights are quantized to the closest power-of-two value available in the target map. In this case, the range is parameterizd by n_1 and the precision by n_2. They determine these parameters with the following rules :

$$n_1 = \lfloor \log_2(\frac{4}{3} \max_{w^l in W^l} |w^l|) \rfloor \tag{22}$$

$$n_1 - n_2 = 2^{n-1} - 1 \tag{23}$$

(22) enables to capture all the range of the weight distribution. (23) comes from the fact that the expected input bitwidth n determines the number of quantization values in the target map.

4.3.2 Weight and activation quantization

Quantizing activations alongside weights enable to take advantage of fixed-point arithmetic so as to accelerate inference. As shown in Figure 8 the operations are realized in limited precision. Therefore, weights must be represented as integers and floating point weight-clustering methods like in 4.3.3 are not applicable. While the weights and activations are typically converted to 8 or lower bit integers, the accumulator and biases are typically computed in higher precision like INT32 or FP32.

All the rounding methods described in 4.2 apply to weight and activation quantization.

With symmetric uniform rounding, we have :

$$x_f = \hat{S}[clip(x)\frac{1}{\hat{S}}] = \hat{S}x_q \tag{24}$$

where $x_q = [clip(x)\frac{1}{\hat{S}}]$ is an integer and $\hat{S}$ can be a floating point.

Let's write the convolution operation when both weights and activations are quantized. We note the input x, the bias b, the output z, the filter W and y the input of the next layer. We have :

$$x_f = \hat{S}_x[clip(x)\frac{1}{\hat{S}_x}] = \hat{S}_x x_q$$

$$W_f = \hat{S}_W[clip(W)\frac{1}{\hat{S}_W}] = \hat{S}_W W_q$$

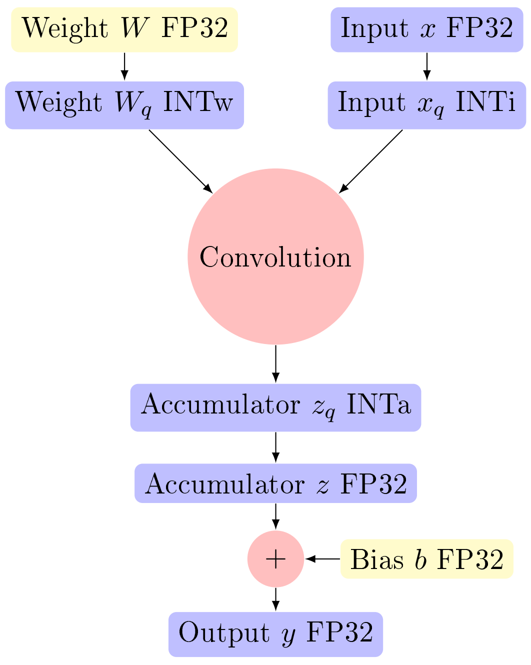

FIGURE 8 – Weight and Activations quantized inference.

W_q is computed offline before inference. However, x_q has to be computed online. However, we can calculate offline an approximation of the associated scaling factor S_x by gathering activation statistics running a few "calibration" batches on the trained model.

The convolution :

$$\begin{aligned} z_f &= x_f * W_f + b && (25a) \\ &= \hat{S}_x \hat{S}_W x_q * W_q + b && (25b) \end{aligned}$$

We can also write :

$$z_f = \hat{S}_x \hat{S}_W (x_q * W_q + \frac{b}{\hat{S}_x \hat{S}_W}) \qquad (26)$$

At this point, the accumulator $x_q * W_q$ is in limited precision while the bias $\frac{b}{\hat{S}_x \hat{S}_W}$ is in full-precision. We can round the bias to the precision of the accumulator if we want to use limited precision addition. It is equivalent to quantize linearly b with scaling factor $\hat{S}_b = \hat{S}_x \hat{S}_W$.

$$z_f = \hat{S}_x \hat{S}_W (x_q * W_q + b_q) \qquad (27)$$

At this point we have the FP32 output of the convolution. We also have a scaling factor S_z for the output z_f. It can be calculated online based on currrent z_f statistics or preset offline by 'calibration'.

$$z_f = \hat{S}_z z_q \qquad (28)$$

$$z_q = \frac{\hat{S}_x \hat{S}_W}{S_z}(x_q * W_q + b_q) \tag{29}$$

We obtain a FP32 scaled output.

z_q then feeds the activation function and is rounded to limited precision to obtain the quantized input of next layer.

$$z_q = ReLU(z_q) \tag{30a}$$
$$z_q = Round(clip(z_q)) \tag{30b}$$

It is possible to clip and round before ([Jacob et al., 2017]) or after ([Migacz, 2017],[Banner et al., 2019]) ReLU. The input of next layer is then directly quantized to limited precision.

There are several remarks to do here :

— With this implementation, we have only one floating point-operation per filter : $\frac{\hat{S}_x \hat{S}_W}{S_z}$. If the activation statistics are computed offline, it can be calculated offline. If we work in pure fixed-point format (see 4.2), scale factors are in power-of-two format : this multiplication is replaced with bit-shifts ([Gupta et al., 2015]). Else, as explained in [Zmora et al., 2018] and [Jacob et al., 2017], it is also possible to work with a fixed-point multiplication. Let call S the current scale factor. We write $S = 2^{-n}S_0$. S_0 is an integer coded in m bits, where m is a parameter. Given S and m, we determine S_0 and n as follow :

$$S_0 = \lfloor 2^n S \rfloor \tag{31}$$

$$n = \lfloor log_2 \frac{2^m - 1}{S_0} \rfloor \tag{32}$$

[Migacz, 2017] decides to realize it in full-precision. [Jacob et al., 2017] decides to implement it with fixed-point multiplication.

— Specific between layers calculus such as pooling or activation functions different than ReLU may require to work with full-precision activations. In that case, we can directly work with $z_f = \hat{S}_x \hat{S}_W (x_q * W_q + b_q)$, compute pooling and apply quantization. Such a process requires to rescale the input at each layer which cost numerous floating point multiplications. Note that with max-pooling we can work directly with quantized activations, which is not the case with average-pooling for example.

— Clipping : In this formulation, we can add clipping after scaling, contrary to what was formulated in 4.2. To do so, we just have to divide the clipping threshold by the scaling factor.

Clipping methods

[Park et al., 2018] observes that both weight and activation distributions are very concentrated around small values but are also very wide with a small number of large data. The wider the distribution the larger quantization error we obtain with linear quantization. This large data points are outliers of the distribution. Clipping methods proposes to find an optimal threshold for each distribution to remove these outliers. Like in [Zhao et al., 2019], we now give a brief overview of the different clipping methods used in the literature. Given the formulation in 4.2, we want to optimize the threshold value T. Thresholding can be applied at different levels : filters, layers, feature maps. Once again, when working with activation quantization, we need a step forward in the network in order to collect statistics of the activations.

— **Minimal Mean Square Error (MMSE)** :
The method chooses the threshold T that minimizes the $L2$ norm between the original vector x and its quantized version x_q. x_q is obtained through one of the quantization process defined before.

$$E = \frac{1}{N}\sum_{i=1}^{N}((x_q)_i - x_i) \tag{33}$$

[Zhao et al., 2019] generates a large value of possible T values evenly spaced between 0 and the maximum value of x, and choose the T value that minimize the MSE.
[Zhao et al., 2019] also claims that [Shin et al., 2015] and [Sung et al., 2015] use this method. However, their formulation of quantization is quite different. Their objective is to optimize the quantization step s when the size of the quantization grid M is fixed. On the opposite, in our formulation, we have s fixed by the number of bits n, and our objective is to determine the threshold T or equivently M with $M = \lfloor \frac{2T}{s} \rfloor + 1$
With their formulation, they get $x_q = sz(x, s)$ and then $E = \frac{1}{N}\sum_{i=1} N(sz_i - x_i)$. Therefore, they can use a two-step iterative computation of s otimizing first z given s and secondly s given z.

— **Minimal KL divergence** ([Migacz, 2017]) : This method is similar to the previous one exept that it uses a different energy to compare the full-precision x and quantized x_q distributions. Instead of the MMSE, the energy to mimize is the Kullback-Leibler (KL) divergence between both distributions.

$$E = -\sum_{i=1}^{N}(x_q)_i log(\frac{x_i}{(x_q)_i}) \tag{34}$$

— **Analytical Clipping for Integer Quantization (ACIQ)** ([Banner et al., 2019]) : The authors first observe that the activation tensors have bell-shaped distributions and can be well approximated with Gaussian or Laplacian distributions. Given an input tensor to quantize, the method first decide weither it's closer to a Laplacian or a Gaussian distribution. For each of these distributions, they determine a closed-form solution of the clipping threshold minimizing the MSE between original vector and its quantized version. This method is much faster to implement. Moreover the threshold value is not fixed offline but dynamically adapted to the curent activation values.

The authors of [Zhao et al., 2019] and [Migacz, 2017] find that there is no advantage in doing any kind of weight clipping for weight quantization on large bitwidth (larger than 7 or 8 bits). For smaller bitwidths, they don't find any clipping method being better than the other. It depends on the data and architecture used. However activation quantization is effective at all bitwidth from 5 to 8 bits, MMSE outperforms the other methods on the majority of activation quantization cases.

Alternatives of clipping

[Park et al., 2018] states that large weights/activations have a big impact on the quality of the output. Therefore clipping methods are likely to cause accuracy loss.

- [Zhao et al., 2019] proposes a different approach to remove outliers : Outlier Channel Splitting (OCS). First the algorithm spots outliers in weight and activation distributions :
  - For weights, kernels are sorted decreasingly according to their maximum weight absolute value and the first $r\%$ of the kernels are selected.
  - For activation, they use a small dataset to collect activation statistics. Kernels are sorted according the frequency of outliers in their corresponding output feature map (an outlier is an activation value greater than 99% of other activations). Again, the first $r\%$ of the kernels are selected.

  Then the idea is to reduce the magnitude of kernel outliers by 1) duplicating the kernel and 2) dividing its outgoing weights by 2. The output value is unchanged while the outliers are moved towards the center of the distribution. OCS introduces a new trade-off between reducing quantization error at cost of network size overhead. They find that their method outperforms clipping methods for weight quantization but performs worse for activation quantization.

Rather than modifying the distributions of weights and activations like clipping methods or [Zhao et al., 2019], [Park et al., 2018] prefer [Lin et al., 2015] to adapt quantization precision to the given distributions.

- With value-aware quantization (V-quant), [Park et al., 2018] apply reduced quantization precision to the majority of the data i-e small data while keeping high precision for large data. In order to classify data as large or small, they select the ratio that minimizes the bitwidth while maintaining high accuracy. On various networks trained on ImageNet, without retraining, without loss of accuracy, they manage to quantize 97% of data (small data) to 5 bits and the remaining 3% to 8 bits. Here data represents equally weights and activation.
- [Lin et al., 2015] goes even further and chooses to adapt the quantization precision layer-by-layer. This is done by minimizing the network SQNR (Signal to Quantization Noise Ratio). They first find that the SQNR γ_{h_l} associated with the quantization of weights or activations h_l in a layer l is in dB approximately proportional to the bitwidth used to quantize this layer n_{h_l} : $10\log(\gamma_{h_l}) \approx \kappa n_{h_l}$ Then they find that the

SQNR at the output of the network is the harmonic mean of all SQNR of preceding quantization steps :

$$\frac{1}{\gamma_{output}} = \sum_{l=1}^{L} \frac{1}{\gamma_{a^l}} + \frac{1}{\gamma_{W^l}} = \sum_i \frac{1}{\gamma_i} \tag{35}$$

where γ_{a^l} and γ_{W^l} are respectively the SQNRs associated with the quantization of activations and weights at layer l. The authors propose the following optimization problem :

$$\min_{\gamma_i} \sum_i S_i \frac{10\log(\gamma_i)}{\kappa} \quad s.t \quad \sum_i \frac{1}{\gamma_i} \leq \frac{1}{\gamma_{min}} \tag{36}$$

where S_i is the scaling factor uses at quantization step i, γ_{min} is the minimum output SQNR required to achive a certain level of accuracy.
The solution verifies for any quantization steps i and j :

$$n_i - n_j = \frac{10\log(\frac{S_j}{S_i})}{\kappa} \tag{37}$$

the constant κ can be determined with the hypothesis that weight and activations follow a gaussian deistribution. Therefore, once a per-layer scaling factor defined, it is possible to set the determine the bitwidth allocation of weights and activations layer per layer.

4.3.3 Quantization fine-tuning

If retraining is possible, it is useful to retrain the network once or iteratively in order to compensate for the possible accuracy loss induced by weight quantization. With retraining, it is possible to achieve lossless weight quantization to lower bitwidth : When one-shot post-training quantization often enables to work up to 8 bits, retraining enables to reach 3 or 4 bits weight quantization. According the the quantization method used, the retraining procedure differ. The main retraining method is said quantization-aware. This method will be explored in the following section 4.3.4. Retraining falls back to use a method described in 4.3.4 with the model initialized with pretrained full-precision weights.

We explore here other training methods that are specific to post-training quantization :

Fine-tuning after weight clustering quantization

Both articles [Han et al., 2016c] and [Choi et al., 2016] previously studied in further fine-tune quantized shared cluster weight values in order to recover the loss due to quantization. Let's remind that after clustering, the (full-precision) cluster weights are kept in memory along with the index of the cluster they belong to. For retraining, during back-propagation, one need to compute only the gradient for each centroid. The gradient w.r.t one centroid is the sum of all the gradients w.r.t weights belonging to that cluster :

$$\frac{\partial \mathcal{L}}{\partial C_k} = \sum_w \frac{\partial \mathcal{L}}{\partial w} \frac{\partial w}{\partial C_k} = \sum_w \frac{\partial \mathcal{L}}{\partial w} 1_{w \in C_k} \tag{38}$$

Then only the centroid values are updated. Therefore, clustering is realised only once. The retraining only affects the full-precision shared values.

Iterative quantization

Instead of quantizing/retraining an entire network at once [Zhou et al., 2017] employ an iterative method. Starting with the trained full-precision network, it iteratively quantizes only a portion of the model. At each iteration, they partition the weights into two disjoint groups : the weights in the first group are quantized, the weights in the second group should adapt themselves to compensate for the quantization of the first group : they are to be retrained. During retraining, the weights of the second group are updated while the weights of the first group remain fixed. Then procedure is applied iteratively until all weights of the initial network are converted to limited precision. The weight partition strategy is inspired by pruning methods : the weights with larger absolute values are considered more important than the smaller ones and form a low-precision base. Thus, weights are divided according to their absolute value with layer-wise thresholds. Their method gives impressive results on the classic CNN models trained on ImageNet : with 5 bits quantization, the reduced model outperforms the initial full precision network, and they reach an almost lossless quantization with 3 bits. They also outperform [Han et al., 2016c] compression ratio quantizing weights to powers of two when [Han et al., 2016c] produce floating-point weights.

4.3.4 Quantization-aware training

In order to minimize accuracy losses due to quantization, current research tries to train models in a way that consider quantization. These methods can succeed in quantizing models with binary or ternary weights. Note that such bitwidth for weights eliminates the need for multiplications. First works on the subject ([Soudry et al., 2014]) did not train their DNN with backpropagation but with a variant called Expected Backpropagation (EBP). This algorithm approximates the Bayes calculation of the posterior distribution of the weights given new data. The authors of [Soudry et al., 2014] show that it outperforms backpropagation when training fully-connected binary networks. Since [Courbariaux et al., 2015], it is prefered to train with classic backpropagation and with “simulated quantization”. The process in schematized Figure 9. Quantization effect are simulated during training and therefore taken into account for weight updates. All weights and biases are stored in floating points and weight update is done in full-precision. This is necessary because stochastic gradient descent performs small changes that may be smaller than the quantization step.

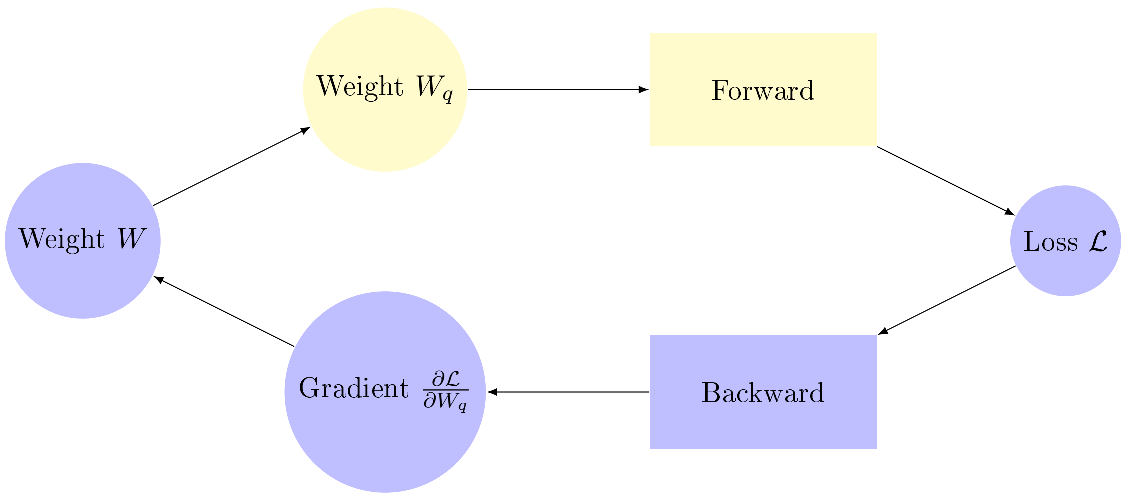

FIGURE 9 – Quantization-aware training. Labels are yellow when we work in limited precision and blue when we work with in full precision.

[Courbariaux et al., 2015] introduce **BinaryConnect**, a method to train binary weight networks. Weights are constrained to be ± 1 and activations remain in full-precision.

— During the forward pass, first binarize the weights and then compute activations.
— Backpropagation is done with binarized weights.
— The update is done on full-precision weights. Updated weights are then clipped to $[-1, 1]$.

They achieve state-of-the art results with small datasets like CIFAR-10. However, [Rastegari et al., 2016] shows that this method is not efficient on larger datasets like ImageNet.

In order to better understand the quantization-aware forward and backward processes, [Zhou et al., 2016] adopts the "straight-through-estimator" (STE) formulation. An STE is an operator that can have arbitrary forward and backward operations. This formulation is made necessary by the fact that quantized weights would have non-definite gradient w.r.t their inputs. For example, with this previous quantization method, the quantization operation realized on weights is replaced by the following STE :

$$\textbf{Forward :}\ x_q = sign(x) \tag{39}$$

$$\textbf{Backward :}\ \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial x_f} 1_{|x| \leq 1} \tag{40}$$

XNOR-Net [Rastegari et al., 2016] was the first work to binarize both weights and activations on large-scale datasets like ImageNet. It allows convolutions to be replaced by XNOR and bitcounting operations leading to $\approx 58\times$ speed-up. They introduce a different weight binarization method. They use scale factors for binarizing both weights and activation at $\pm\alpha$ values instead of ± 1. Filter W_k is binarized to $\pm\alpha_k$ with a scale $\alpha_k = \frac{1}{|W_k|}||W_k||_{l1}$:

the average of absolute values. However to compute the activation scale factor in the same way, we need activation statistics and thus to compute the full-precision convolution which is computationally inefficient.

The corresponding weight STE for filter x is :

$$\textbf{Forward :}\ x_q = sign(x) * \frac{1}{|x|}||x||_{l1} \tag{41}$$

$$\textbf{Backward :}\ \frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial x_f} \tag{42}$$

Activation quantization also induce its own activation STE.

Reducing the precision of activations maps hurts model accuracy much more than reducing precision of model parameters. However, [Mishra et al., 2018] shows that the memory footprint of activation maps is larger than the one of weights, especially for large batch-sizes. **WRPN** [Mishra et al., 2018] proposes to find a trade-off to quantize activations correctly alongside weights and find that 2-bits weights and 4-bits activations are just enough to match full-precision accuracy with training-aware quantization. The accuracy loss due to quantization is compensated by doubling the number of filter maps at each layer, which hurts considerably the final compression ratio.

ABC-Net [Lin et al., 2017] first study why the previous binary networks may fail and propose corresponding improvements :

— All the previous methods decide not to quantize the first and last layers of the network. It severely degrades the global compression rate. Indeed, as shown in [Zhou et al., 2016], quantizing the last layer makes the network more difficult to train and causes high performance drops. This performance drop is due to the large variation range of the quantized output that is not adapted to the softmax activation function : a large number of values fall into the saturation region of the softmax. [Lin et al., 2017] proposes instead to add a scale layer before the softmax function. This trick enables to multiply the global compression rate given in [Zhou et al., 2016] by 3 with almost no loss in accuracy.

— They analyse why BinaryConnect perform poorly on a large dataset like ImageNet and find that it is due to improper training strategy : the authors of [?] use a too large learning rate. At each training step, too many weights change their sign and thus their resulted binarized value. This causes high training instability. A diminution of the learning rate leads to great accuracy gains.

— Instead of introducing a scale-factor to quantize weights and activations like in [Rastegari et al., 2016],they decide to quantize weights and activations to ± 1 and then use PReLU instead of a classic RelU activation function. PReLU stands for Parametric ReLU :

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{otherwise} \end{cases} \tag{43}$$

With their method they manage to obtain a top1/top5 accuracy with binarized AlexNet (trained on ImageNet) of 75.6/51.4 compared to 80.2/56.6 for the full-precision model.

Gradient quantization

In the previous formulation, the back-propagation pass still requires convolution between floating point tensors. Most training time will therefore be spent during Back-propagation. In order to accelerate training, it is possible to quantize gradients as well. [Gupta et al., 2015] managed lossless training with with 16 bits fixed-point gradients. However, with a deterministic uniform rounding quantization method, the network could not learn with a further reduction in precision (below 14 bits). However, with stochastic rounding, they show that the network is able to learn with as few as 8 bits gradients.

[Zhou et al., 2016] even succeeds to quantize gradients to 6-bits during backpropagation, without performance degradation. They want to quantize the gradient $\Delta_a \mathcal{L}$ of the loss function $\mathcal{L}$ w.r.t some activation tensor a. In order to use a stochastic estimator, [Zhou et al., 2016] does not use Bernoulli random variables like explained in 4.2 but uses the following quantization method :

$$x_q = sc^{-1}(Q(sc(x))) = \hat{Q}(x) \tag{44}$$

$$Q(x_i) = \frac{\lfloor x_i s \rfloor + \epsilon_i}{s} \tag{45}$$

$$\epsilon_i \sim Uniform(-0.5, 0.5) \tag{46}$$

The random noise function ϵ_i is used to compensate the potential bias brought by gradient quantization. Gradient quantization is realized during backprop at each layer. It corresponds to the following STE :

$$\textbf{Forward :}\ x_q = x \tag{47}$$

$$\textbf{Backward :}\ \frac{\partial \mathcal{L}}{\partial x} = \hat{Q}(\frac{\partial \mathcal{L}}{\partial x_f}) \tag{48}$$

4.4 RNN quantization

An LSTM layer involves a collection of matrices, matrix products and element-wise multiplications. [Han et al., 2016a] quantizes weights and activations with linear quantization.

Weights are considered in the matrix they belong. Activation vectors correspond to the different cell input and output vectors. Without retraining, they manage to quantize a LSTM trained on the TIMIT corpus to 12 bits weights and activations.

4.5 Why does quantization work

As described in 3, neural networks, especially large CNN, have a very important number of redundant connections. Network pruning aims at keeping only the most important parameters. Every weight will have a strong impact in the inference process. On the opposite, network quantization exploits the redundancy to enable the reduction of the expressivity power of each weight. This observation is supported by the experiments conducted by [Mishra et al., 2018] and reported in Table 3. When widening the net, redundancy is created and it improves the performances.

Model	Accuracy
Resnet-34 (32b A, 32b W)	73.59
Resnet-34 (1b A, 1b W)	60.54
x2 filers/layer (1b A, 1b W)	69.85
x3 filers/layer (1b A, 1b W)	72.38

TABLE 3 – [Mishra et al., 2018] Binerization of ResNet-34 combined with an increase in the number of channels in each layers.

Quantization of a neural network can be seen as the introduction of noise in the network. A first observation can be that as neural network models are build to be robust to the noise present in the inputs, it makes sense that quantization does not strongly hurt the performances. To go further, with successive convolutions , redundant filters in a layer will be more or less averaged together. We can imagine that this averaging acts as denoising method to remove the quantization noise.

5 Knowlegde Distillation

A simple and common way to improve the performances of any machine learning model is to train multiple models on the same data and average their predictions. This process can be very long and require a large computational power.

5.1 Teacher-student knowledge distillation

[Buciluǎ et al., 2006] and [Ba and Caurana, 2013] first introduced knowledge distillation. It is the process of compressing a deep and wide network into a shallower one that mimick the function learned by the complex model. As stated by [Hinton et al., 2015] the challenge is to train the smaller network to generalize in the same way as the large model. Indeed, it is easy to make a large model generalize well because it can be for example the average of a ensemble of different models but it is much more complex for a smaller model.

In their works, model compression is done by passing unlabeled data called 'transfer set' through the large, accurate model. This new labeled data is then used to train the smaller mimic model. This smaller model is not trained on the original labels but on the function that was learned by the initial model. In practice the unlabeled data is labelled by a an ensemble of large neural network models.

As previously explained in 2.1.2, neural networks typically produce a class of probabilities passing the outputs through a "softmax" activation function σ that converts the output z_i for each class i into a probability p_i :

$$p_i = \sigma(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{49}$$

The authors in [Ba and Caurana, 2013] use the logits z_i rather than the output softmax probabilities p_i as targets for learning the small model. [Hinton et al., 2015] instead proposes to use a "soft" target probability distribution produced by the cumbersome model.

Let us call z_s and z_t the logits of respectively the student and teacher model, and $\mathcal{H}$ the cross-entropy loss function. In order to train the small network, they use a loss function $\mathcal{L}_\lambda$ which is a weighted average of two different loss functions :

- The cross-entropy loss between the ground-truth labels y_s and the output of the student network, passed through a softmax $\sigma(z_s)$.
- The Knowledge-Distillation loss i-e the cross-entropy loss between "soft targets" $\sigma(\frac{z_t}{T})$ and "soft outputs" $\sigma(\frac{z_s}{T})$. This soft distributions are created with a softmax functions weighted y a temperature T. It enables to put control on softening the signal arising from the output of the teacher network. It is also necessary to multiply this loss by T^2 in order to compensate for the $\frac{1}{T^2}$ factor after derivation. Note that it is here equivalent to use the cross-entropy loss or the KL divergence loss as the gradient is calculated w.r.t the student parameters.

These two losses are weighted by a trade-off parameter λ. All in all,

$$\mathcal{L}_\lambda = (1-\lambda)\mathcal{H}(\sigma(z_s), y_s) + \lambda T^2 \mathcal{H}(\sigma(\frac{z_s}{T}), \sigma(\frac{z_t}{T})) \tag{50}$$

The success of these works is to correlate with network pruning 3 : small networks often have the same representation capacity as large networks but they are just harder to train. Like network pruning, the distillation approach starts with a powerful cumbersome network and the trains a smaller student network. This smaller model can match or even outperform the target teacher. [Ba and Caurana, 2013] brings some reasons for this :

- In region where probability is difficult to learn, the teacher model provides simpler and soft labels to the student. The complexity in the dataset has been filtered by the teacher.

— It is easier to learn on soft targets than on 0/1 labels : the teacher model spreads the uncertainty over outputs thanks to probabilities that are much more informative.

5.2 Mutual learning

Recently, [Zhang et al., 2017] developed a related method called mutual learning. Instead of starting with a powerful large pretrained teacher network, they start directly with a collection of small untrained student networks who learn simultaneously together. Let us call K the number of student models $(\Theta_k)_{k\leq K}$.

Just like [Hinton et al., 2015], each Θ_k is trained with a loss $\mathcal{L}_k$ that a combination of two different losses :

— The classical cross-entropy with the ground truth labels and the output of the network $\mathcal{H}(\sigma(z_k), y_k)$

— The average of the KL-divergences between the considered network output probability distribution $\sigma(z_k)$ and the other student distributions $\sigma(z_l) l \neq k : KL(\sigma(z_k)|\sigma(z_l))$.

$$\mathcal{L}_k = \mathcal{H}(\sigma(z_k), y_k) + \frac{1}{K-1}\sum_{l=1, l\neq k}^{K} KL(\sigma(z_k)|\sigma(z_l)) \tag{51}$$

The authors experiment on CIRFAR100. For example, with only two student networks, ResNet32 and MobileNet, learning mutually, it is possible to improve their respective accuracy performance of 2.11% and 2.48%. As a comparison, they show that using the method of [Hinton et al., 2015] previously explained, to distill ResNet32 with a MobileNet teacher offers only a 0.5% accuracy gain.

5.3 Knowledge distillation to learn low-precision networks

Recent research focus on applying jointly knowledge distillation and quantization for better compression. The idea is to force the student model to be in quantized form and to train it from a full-precision teacher.

[Polino et al., 2018] develops a straightforward application of knowledge distillation to quantize weights. We consider a trained full-precision teacher model and a shallower and quantized student. The quantization-aware training of the student model is realised like in [Courbariaux et al., 2015] and explained in 4.3.4. They use the famous 'the distillation loss' from [Hinton et al., 2015] described above.

[Mishra and Marr, 2017] uses the same approach and compares a mutual learning approach and a pure teacher-student approach. They try three different schemes with a big ResNet teacher and a smaller ResNet student.

— The full-precision teacher network is jointly trained with the low-precision student.

— The full-precision teacher is trained first, and the student secondly with distillation, like in [Polino et al., 2018].
— Same as second but the student is first initialized with the full-precision training weights.

They find that with scenarios 1 and 2 and 3 give very similar performances, except that, as expected, training converges faster with 3.

6 Conclusion

In this work we studied the most common methods to compress and acelerate the inference of a neural network. We showed that pruning, quantification and distillation are three effective methods that can work jointly to obtain the best performances. In an other report, we are going to present the impementation of some of these algorithms.

Références

[Anwar et al., 2015] Anwar, S., Hwang, K., and Sung, W. (2015). Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571.

[Ba and Caurana, 2013] Ba, L. J. and Caurana, R. (2013). Do deep nets really need to be deep ? *CoRR*, abs/1312.6184.

[Banner et al., 2019] Banner, R., Nahshan, Y., Hoffer, E., and Soudry, D. (2019). ACIQ : Analytical clipping for integer quantization of neural networks.

[Buciluǎ et al., 2006] Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 535–541, New York, NY, USA. ACM.

[Choi et al., 2016] Choi, Y., El-Khamy, M., and Lee, J. (2016). Towards the limit of network quantization. *CoRR*, abs/1612.01543.

[Courbariaux et al., 2015] Courbariaux, M., Bengio, Y., and David, J. (2015). Binaryconnect : Training deep neural networks with binary weights during propagations. *CoRR*, abs/1511.00363.

[Garofolo et al., 1993] Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., Pallett, D. S., and Dahlgren, N. L. (1993). Darpa timit acoustic phonetic continuous speech corpus cdrom.

[Gong et al., 2014] Gong, Y., Liu, L., Yang, M., and Bourdev, L. D. (2014). Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115.

[Gupta et al., 2015] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. *CoRR*, abs/1502.02551.

[Han et al., 2016a] Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., Yang, H., and Dally, W. J. (2016a). ESE : efficient speech recognition engine with compressed LSTM on FPGA. *CoRR*, abs/1612.00694.

[Han et al., 2016b] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. (2016b). EIE : efficient inference engine on compressed deep neural network. *CoRR*, abs/1602.01528.

[Han et al., 2016c] Han, S., Mao, H., and Dally, W. J. (2016c). Deep compression : Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626.

[Hassibi et al., 1993] Hassibi, B., Stork, D. G., Wolff, G., and Watanabe, T. (1993). Optimal brain surgeon : Extensions and performance comparisons. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 263–270, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*.

[Hu et al., 2016] Hu, H., Peng, R., Tai, Y., and Tang, C. (2016). Network trimming : A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250.

[Huffman, 1952] Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9) :1098–1101.

[Jacob et al., 2017] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A. G., Adam, H., and Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877.

[jcjohnson, 2017] jcjohnson (2017). cnn-benchmark. `https://github.com/jcjohnson/cnn-benchmarks`.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[LeCun et al., 1990] LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann.

[Li et al., 2016] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

[Lin et al., 2015] Lin, D. D., Talathi, S. S., and Annapureddy, V. S. (2015). Fixed point quantization of deep convolutional networks. *CoRR*, abs/1511.06393.

[Lin et al., 2017] Lin, X., Zhao, C., and Pan, W. (2017). Towards accurate binary convolutional neural network. *CoRR*, abs/1711.11294.

[Liu et al., 2018] Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. (2018). Rethinking the value of network pruning. *CoRR*, abs/1810.05270.

[Luo et al., 2017] Luo, J., Wu, J., and Lin, W. (2017). Thinet : A filter level pruning method for deep neural network compression. *CoRR*, abs/1707.06342.

[Migacz, 2017] Migacz, S. (2017). 8-bit inference with tensorrt.

[Mishra et al., 2018] Mishra, A., Nurvitadhi, E., Cook, J. J., and Marr, D. (2018). WRPN : Wide reduced-precision networks. In *International Conference on Learning Representations*.

[Mishra and Marr, 2017] Mishra, A. K. and Marr, D. (2017). Apprentice : Using knowledge distillation techniques to improve low-precision network accuracy. *CoRR*, abs/1711.05852.

[Molchanov et al., 2016] Molchanov, P., Tyree, S., Karras, T., Aila, T., and Kautz, J. (2016). Pruning convolutional neural networks for resource efficient transfer learning. *CoRR*, abs/1611.06440.

[Park et al., 2018] Park, E., Yoo, S., and Vajda, P. (2018). Value-aware quantization for training and inference of neural networks. *CoRR*, abs/1804.07802.

[Polino et al., 2018] Polino, A., Pascanu, R., and Alistarh, D. (2018). Model compression via distillation and quantization. *CoRR*, abs/1802.05668.

[Qin et al., 2018] Qin, Z., Yu, F., Liu, C., and Chen, X. (2018). Demystifying neural network filter pruning. *CoRR*, abs/1811.02639.

[Rastegari et al., 2016] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. (2016). Xnor-net : Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3) :211–252.

[Shin et al., 2015] Shin, S., Hwang, K., and Sung, W. (2015). Fixed point performance analysis of recurrent neural networks. *CoRR*, abs/1512.01322.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

[Soudry et al., 2014] Soudry, D., Hubara, I., and Meir, R. (2014). Expectation backpropagation : Parameter-free training of multilayer neural networks with continuous or discrete weights. volume 2.

[Sung et al., 2015] Sung, W., Shin, S., and Hwang, K. (2015). Resiliency of deep neural networks under quantization. *CoRR*, abs/1511.06488.

[Szegedy et al., 2014] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

[Zhang et al., 2017] Zhang, Y., Xiang, T., Hospedales, T. M., and Lu, H. (2017). Deep mutual learning. *CoRR*, abs/1706.00384.

[Zhao et al., 2019] Zhao, R., Hu, Y., Dotzel, J., Sa, C. D., and Zhang, Z. (2019). Improving neural network quantization without retraining using outlier channel splitting. *CoRR*, abs/1901.09504.

[Zhou et al., 2017] Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y. (2017). Incremental network quantization : Towards lossless cnns with low-precision weights. *CoRR*, abs/1702.03044.

[Zhou et al., 2016] Zhou, S., Ni, Z., Zhou, X., Wen, H., Wu, Y., and Zou, Y. (2016). Dorefa-net : Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, abs/1606.06160.

[Zhu and Gupta, 2017] Zhu, M. and Gupta, S. (2017). To prune, or not to prune : exploring the efficacy of pruning for model compression.

[Zmora et al., 2018] Zmora, N., Jacob, G., and Novik, G. (2018). Neural network distiller.