Deep Model Compression for Mobile Devices : A Survey

Anthony Berthelier[1,2] Priyanka Phutane[2] Stefan Duffner[3] Christophe Garcia[3]
Christophe Blanc[1] Thierry Chateau[1]

[1] Université Clermont Auvergne, France
[2] Wisimage, France
[3] Université de Lyon, CNRS, INSA Lyon, LIRIS, UMR5205, Lyon, France

anthony.berthelier@etu.uca.fr

Résumé

Ces dernières années, les réseaux de neurones profond sont devenus indispensables pour le développement d'applications intelligentes. Ils ont atteint des performances remarquables, devenant plus complexes et cumulant des millions de paramètres. Ainsi, faire fonctionner ce type de modèle sur des appareils disposant de ressources limitées comme des téléphones mobiles n'est pas une tâche triviale. Cet article a pour but de présenter des méthodes permettant le portage de ces modèles sur plateformes mobiles. Nous nous focalisons sur des techniques de compression permettant de diminuer la consommation des modèles de manière globale (en taille, en mémoire, ou en temps de calcul par exemple).

Mots Clef

Apprentissage profond, compression, plateformes mobiles

Abstract

Over the past, deep neural networks have proved to be an essential element for developing intelligent solutions. They have achieved remarkable performances at a cost of deeper layers and millions of parameters. Therefore utilizing these networks on limited resource platforms such as mobile phones is a challenging task. This paper presents a survey of methods suitable for porting these models to mobile devices. We are mainly focusing on compression techniques which decrease the overall resource requirements (e.g. size, memory, computation time).

Keywords

Deep learning, compression, mobile devices

1 Introduction

Since the advent of deep neural network architectures and their massively parallelized implementations [37, 39], deep learning based methods have achieved state-of-the-art performance in many applications such as face recognition, semantic segmentation, object detection, etc. In order to achieve these performances, a high computation capability is needed as these models have usually millions of parameters. Moreover, the implementation of these methods on resource-limited devices is difficult due to memory consumption and size constraints. For example, AlexNet [37], is over 200MB and all the milestone models that followed such as VGG [54], GoogleNet [59] and ResNet [28] performed well however are not necessary time or memory efficient. Thus finding solutions to implement deep models on resource-limited platforms is essential. For instance, smartphones have a limited memory and each device has a different computational capacity. Therefore, to run these applications on embedded devices the deep models need to be less-parametrized in size and time efficient.

Few works have been done focusing on dedicated hardware or FPGA with a fixed specific architecture. Having a specific hardware is helpful to optimize a given application. However, it is difficult to generalise. The CPU architectures of the smartphones are different from each other. Thus, it is important to develop generic methods to help optimize neural networks. This paper aims to describe general compression methods for deep models that can be implemented on a large range of hardware architectures, especially on various generic-purpose CPU architectures. These compression techniques are reducing the size and computation requirements of a model by using different algorithms. We classify these methods in several parts and describe them briefly. Firstly, knowledge distillation methods are explained to tackle the problem of transfer learning (Section 2.1). Followed are the hashing (Section 2.2), pruning (Section 2.3) and quantization (Section 2.4) methods which explore the redundancy of a network. Numerical precision (Section 2.5) and binarization (Section 2.6) are mentioned by introducing the use of data with lower precision. In each of these parts is presented existing methods, their strengths, weaknesses and in which context they may be applied.

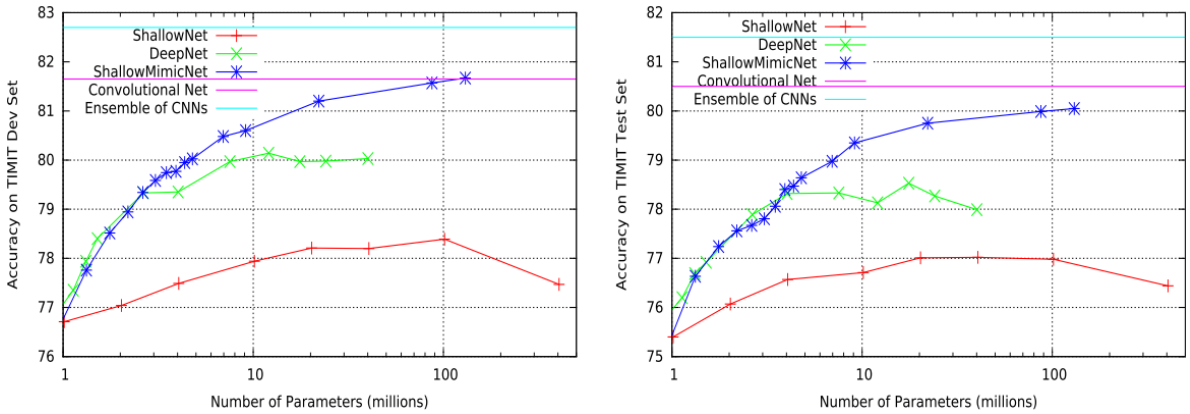

FIGURE 1 – Accuracy of different deep neural networks, shallow neural networks and shallow mimic neural networks against their number of parameters on TIMIT speech database Dev (left) and Test (right) sets. Results and figures are from [5].

2 Compression techniques

2.1 Knowledge distillation

To design a neural network, it is important to evaluate how deep the network needs to be. A neural network is composed of an input, an output and intermediate layers. A shallow neural network is a network with a lower number of intermediate layers as opposed to a deep neural network. A deeper network has more parameters and can potentially learn more complex functions e.g. hierarchical representations [16]. The theoretical work from [16] revealed the difficulty involved to train a shallow neural network with the same accuracy as a deep network. However, an attempt was made to train a shallow network on SIFT features in order to classify the Imagenet dataset [37]. The authors concluded that it was a challenging task to train highly accurate shallow models [16].

In spite of that, Ba et al. [5] reported that neural networks with a shallower architecture are able to learn the same function as deep networks, with a better accuracy and sometimes with a similar number of parameters (see Figure 1). Inspired from [7], their model compression consists in training a compact model to approximate, to mimic, the function learned by a complex model. The preliminary step is to train a deep network (the teacher network) to generate automatically labelled data by sending unlabelled data through this deep network. Next, this "synthetic" dataset is then used to train a smaller mimic model (the student network), which assimilates the function that was learned by the larger model. It is expected that the mimic model should produce same predictions and mistakes as the deep network. Thus, similar accuracy can be achieved between an ensemble of neural networks and its mimic model with 1000 times fewer parameters. In [5], the authors demonstrated this assertion on the CIFAR-10 dataset. An ensemble of deep Convolutional Neural Network (CNN) models was used to label some unlabeled data of the dataset. Next, the new data were used to train a shallow model with a single convolution and maxpooling layer followed by a fully connected layer with 30k non-linear units. In the end, the shallow model and the ensemble of CNN acquired the same level of accuracy. Further improvements have been made on student-teacher techniques, especially with the work of Hinton et al. [29]. Their framework utilizes the output from the teacher's network to penalize the student network. Additionally it is also capable of retrieving an ensemble of teacher networks to compress their knowledge into a student network of similar depth.

In recent years, other compression methods that are described in this paper are preferred. However, some works are coupling transfer learning techniques with their own methods to achieve strong improvements. For example, the works of Chen et al. [10] and Huang et al. [34] follow this approach employing additional pruning techniques (see section 2.3). The former uses a deep metric learning model, whereas the latter handles the student-teacher problem as a distribution matching problem by trying to match neuron selectivity patterns between them to increase the performance. These methods are efficient. However their performances can vary largely according to the application. Classification tasks are easy to learn for a shallow model, but tasks like segmentation or tracking are difficult to apprehend even with a deep model. Thus, distilling this knowledge is also a difficult mission.

2.2 Hashing

Hashing is employed to regroup data in a neural network to avoid redundancy and access the data faster. Through empirical studies, hashing methods have proven themselves to be an effective strategy for dimensionality reduction [61]. HashedNets [9] is a hashing methods utilized and developed by Nvidia. In this model, a hash function is used to uniformly and randomly group network connections into hash buckets. As a result, every connection that is in the i^{th} hash bucket has the same weight value w_i. This technique is especially efficient on fully connected feed forward neural networks. Moreover, It can also be used in conjunction with other neural network compression methods.

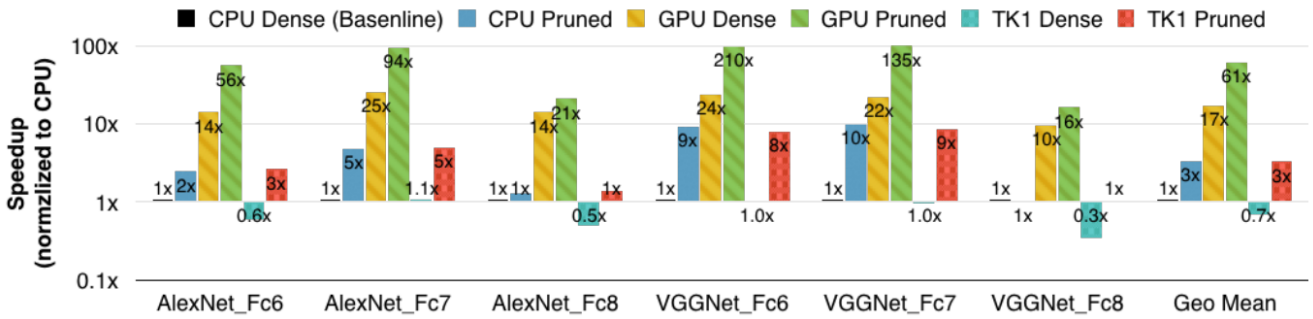

FIGURE 2 – Comparison of the speed of AlexNet and VGG before and after pruning on CPU, GPU and TK1. Figure from [25].

Several other hashing methods have been developed in the past few years. Spring et al. [55] proposed an approach where adaptive dropout [6] (i.e. choosing nodes with a probability proportional to some monotonic functions of their activations) and hash tables based on locality-sensitive hashing (LSH) [20, 52, 58, 33] are utilized. These techniques once combined allowed the authors to construct a smart structure for maximum inner product search [53]. This technique exhibits better results, reducing computational costs for both training and testing. Furthermore, this kind of structure leads to sparse gradient updates and thus a massively asynchronous model. Thereby, models can be easily parallelized as the data dispersion could be wider. However, wider data dispersion can result in a slow down of the model. A trade-off between these criteria is necessary.

2.3 Pruning

The compression of neural networks by using pruning techniques has been widely studied. These techniques enable to remove parameters of a network that are not necessary for a good inference. The early work in this domain was aiming to reduce the complexity and the over-fitting in networks [15, 27]. In these papers, the authors used pruning techniques based on the Hessian of the loss function to reduce the number of connections inside the network. The method finds a set of parameters whose deletion would cause the least increase of the objective function by measuring the saliency of these parameters. The authors use numerous approximations to find these parameters. For instance, the objective function is approximated by a Taylor series. Finding parameters whose deletion does not increase this function is a difficult problem that involves, for example, the computation of huge matrices as well as second derivatives. Also, these methods suggest that reducing the number of weights by using the Hessian of the loss function is more accurate than magnitude-based pruning like weight decay. Additionnally, it reduces the network over-fitting and complexity. However, the second-order derivatives introduce some computational overhead.
Signorini et al. [26] utilized an intuitive and efficient method to remove parameters. The first step is to learn the connectivity of the network via a conventional training of the network i.e. to learn which parameters (or connections) are more important than the other. The next step consists in pruning those connections with weights below a threshold i.e. converting a dense network into a sparse one. Further, the important step of this method is to retrain (fine-tune) the network to learn the weights of the remaining sparse connections. If the pruned network is not retrained, then the resulting accuracy is considerably lower.
Anwar et al. [4] used a similar method. However, they state that pruning has the drawback of constructing a network that has "irregular" connections, which is inefficient for parallel computing. To avoid this problem, the authors introduced a structured sparsity at different scales for CNN. Thus, pruning is performed at : the feature map, the kernel and the intra-kernel levels. The idea is to force some weights to zero but also to use sparsity at well defined activation locations in the network. The technique consists in constraining each outgoing convolution connection for a source feature map to have similar stride and offset. This results in a significant reduction of both feature and kernel matrices. Usually, sparsity has been studied in numerous works in order to penalize non-essential parameters [62, 66, 3, 38].
Similar pruning approach is seen in Molchanov et al. [48]. However different pruning criteria and technical considerations are defined to remove features maps and kernel weights, e.g. the minimum weight criteria [26]. They assume that if an activation value (an output feature map) is small, then the feature detector is not important in the application. Another criteria involves the mutual information which measures how much information is present in a variable about another one. Further, the Taylor expansion is used similar to LeCun [15], to minimise the computational cost between the pruned and the non-pruned network. In this case, pruning is treated as an optimization problem.
A recent pruning method [41] consists in removing filters that are proven to have a small impact on the final accuracy of the network. This results in automatically removing the filter's corresponding feature map and related kernels in the next layer. The relative importance of a filter in each layer

is measured by calculating the sum of its absolute weights, which gives an expectation of the magnitude of the output feature map. At each iteration, the filters with the smallest values are pruned. Recently, Jian-Hao et al. [45] developed a pruning network called ThiNet which, instead of using information of the current layer to prune unimportant filters of that layer, uses information and statistics of the subsequent layer to prune filters from a given layer. Not only weights and filters but also channels can be pruned [44] using complex thresholding methods.

Numerous pruning methods exist and each of them has strength and weaknesses. The main disadvantage of these methods is that it takes a long time to prune networks due to the constant retraining that they demand. Recent techniques like [42] try to bypass some steps by pruning neural networks during their training by using recurrent neural networks. However, all of them result in considerable reduction of parameters. Pruning methods allow to eliminate 10 to 30 percent of the network's weights. Regardless of the method, the size of a network can be decreased with pruning without change or significant drop in accuracy. The inference with the resulting models will also be faster (see Figure 2) but the actual speed depends on which method has been utilized and the sparsity of the network after pruning.

2.4 Quantization

Network quantization is similar to pruning as this is a common technique in the deep learning community. It aims to reduce the number of bits required to represent every weight. In other words, it decreases the number of parameters by exploiting redundancy. Quantization reduces the storage size with minimal loss in performance. In a neural network, it means that parameters will be stacked into clusters. As a result, the parameters in the same cluster will share the same value.

Gong et al. [22] performed a study on a series of vector quantization methods and found that performing scalar quantization on parameter values using a simple k-means is sufficient to compress them 8 to 16 times without a huge loss in accuracy. Few years later, Han et al. [25] utilized a trivial quantization method using k-means clustering. They performed a pruning step before and a Huffman coding step after the quantization in order to perform a larger compression of the network. In their experiments, the authors were able to reduce network storage by 35 to 49 times across different networks. Pruning and quantization are methods that are often used together to achieve a solid compression rate. For example, for a LeNet5-like network [40], pruning and quantization compressed the model 32 times and with huffman coding even 40 times.

It is possible to apply several quantization methods on neural networks. Choi Y. et al. [11] defined a Hessian-weighted distortion measure as an objective function in order to decrease the quantization loss locally. Further, a Hessian-weighted k-means clustering is used for quantization purposes to minimize the performance loss. Recent neural network optimizers can provide alternatives to the Hessian and thus reduce the overall computation cost, like Adam [36], AdaGrad [18], Adadelta [64] or RMSProp [30]. However one of the advantages of using the Hessian-weighted method is that the parameters of all layers in a neural network can be quantized together at once compared to the layer-by-layer quantization used previously [25, 22].

Quantization techniques are efficient as they achieve an impressive compression rate and can be coupled with other methods to compress the models further. Their efficiency is integrated in some frameworks and tools to directly quantify a network and port it on mobile devices [1, 2].

2.5 Reducing numerical precision

Although the number of weights can be considerably reduced using pruning or quantization methods, the overall number of parameters and costly matrix multiplications might still be enormous. A solution is to reduce the computational complexity by limiting the numerical precision of the data. Deep neural networks are usually trained using 32-bit floating-point precision for parameters and activations. The aim is to decrease the number of bits used (16, 8 or even less) and to change from floating-point to a fixed-point representation. Selecting the precision of data has always been a fundamental choice when it comes to embedded systems. When committed to a specific system, the models and algorithms can be optimized for the specific computing and memory architecture of the device [19, 21, 60].

However, applying quantization for deep neural networks is a challenging task. Quantization errors might be propagated and amplified throughout the model and thus have a large impact on the overall performance. Since the beginning of the 90's, experiments have been made in order to limit the precision of the data in a neural network, especially during backpropagation. Iwata et al. [35] created a backpropagation algorithm with 24-bit floating-point processing units. Hammerstrom [24] presented an architecture for on-chip learning using 8-16 bits fixed-point arithmetic. Furthermore, Holt and Hwang [31] showed empirically that only 8-16 bits are enough for backpropagation learning. Nonetheless, even if all these works are helping to understand the impact of limited numerical precision on neural networks, they are done on rather small models such as multilayers perceptron with only a single hidden layer and very few units. More sophisticated algorithms are required for more complex deep models.

In 2015, Gupta et al. [23] trained deep CNN using 16-bit fixed-point instead of 32-bit floating-point precision. It constrained neural networks parameters such as bias, weights and other variables used during the backpropagation such as activations, backpropagated error, weight updates and bias updates. Different experimentations have been made with this 16-bit fixed-point word length, e.g. varying the number of bits that encode the fractional (integer) part between 8 (8), 10 (6) and 14 (2), respectively. In other

terms, the number of integer bits IL added to the number of fractional bit FL is always equal to 16. Tested on the MNIST and CIFAR-10 datasets with a fully connected and a convolutional network, the results were nearly the same as the floating-point baseline when decreasing the fractional part to 12-bit precision.
The crucial part in this method is the conversion of a floating point number (or higher precision format) into a lower precision representation. To achieve this, [23] describe two rounding schemes. The first one is the round-to-nearest method. It consists of defining $\lfloor x \rfloor$ as the largest integer multiple of $\epsilon = 2^{-FL}$ less than or equal to x. So given a number x and the target representation (IL,FL), the rounding is done as follows :

$$\begin{cases} \lfloor x \rfloor & if \lfloor x \rfloor \leq x \leq \lfloor x \rfloor + \frac{\epsilon}{2} \\ \lfloor x \rfloor + \epsilon & if \lfloor x \rfloor + \frac{\epsilon}{2} \leq x \leq \lfloor x \rfloor + \epsilon \ . \end{cases} \tag{1}$$

The second rounding scheme is stochastic rounding. It is a statistic and unbiased rounding where the probability of x to be rounded to $\lfloor x \rfloor$ is proportional to its proximity to $\lfloor x \rfloor$:

$$\begin{cases} \lfloor x \rfloor & w.p. \quad 1 - \frac{x - \lfloor x \rfloor}{\epsilon} \\ \lfloor x \rfloor + \epsilon & w.p. \quad \frac{x - \lfloor x \rfloor}{\epsilon} \ . \end{cases} \tag{2}$$

Courbariaux et al. [12] investigated the impact of numerical precision, especially to reduce the computational cost of multiplications. Their experiments were performed with three formats : floating point, fixed point [23] and dynamic fixed point [63] (which is a compromise of the first two). Instead of having a single scaling factor with a fixed number for the integer part and another fixed number for the fractional part, several scaling factors are shared between grouped variables and are updated from time to time. The authors achieved similar conclusions as [23] : a low precision is sufficient to run and train a deep neural network. However, limited precision can be efficient when it is paired and optimized with a specific hardware. Gupta et al. [23] achieved good results when they paired the fixed point format with FPGA-based hardware but the hardware optimization of dynamic fixed point representations is not as simple. Neural networks with limited-precision parameters and their optimized integration on hardware have already been studied in the past. For example, Mamalet et al. [47] and Roux et al. [51] developed optimized CNNs to detect faces and facial features in videos on embedded platforms in real-time. They used a fixed-point parameter representation but also optimized the inference algorithms for specific platforms. This allowed them to exploit parallel computing and memory locality.
To conclude, a limited numerical precision is sufficient to train deep models. It is helpful to save memory storage and computation time, even more if a dedicated hardware is used. However, not every step can be done with low precision in a neural network. For instance, a higher precision must be used to update the parameters during training.

2.6 Binarization

In recent works, limited numerical precision was extended to binary operations. In a binary network, the weights and the activations at least are constrained to either $+1$ or -1. Following the same idea as previously with limited numerical precision [12], the same authors decided to apply two rounding schemes to binarize a variable : deterministic and stochastic rounding [14]. The most common rounding method is to maintain the sign of the variable. So for a variable x, its binary value x^b will be the sign of x ($+1$ if $x \geq 0$, -1 otherwise). The second binarization scheme is a stochastic rounding. Thus $x^b = +1$ with probability $p = \sigma(x)$ and $x^b = -1$ with probability $1 - p$ where σ is the hard sigmoid function [14]. The stochastic method is difficult to implement as it requires randomly generating bits from the hardware. As a result, the deterministic method is commonly used. However, recent works like [43] are focusing on alternative methods to approximate the weight values in order to obtain a more accurate network.
Nevertheless, just like limited numerical precision, a higher precision is required at some point and real-valued weights are required during the backpropagation phase. Adding noise to weights and activations is beneficial to generalization when the gradient of the parameters are computed (as with dropout [56, 57]). Binarization can also be seen as a regularization method [14].
With all these observations, Courbariaux et al. [13] developed a method called BinaryConnect to train deep neural networks using binary weights during the forward and backward propagation, while storing the true precision of the weights in order to compute the gradients. Firstly the forward propagation : layer-by-layer, the weights are binarized and the computation of the neuron's activation is faster because multiplications are becoming additions. Secondly the backward propagation : the training objective's gradient is computed in function of each layer's activation (from the top layer and going down layer-by-layer until the first hidden layer). Lastly the parameter update : the parameters are updated using their previous values and their computed gradients. During this final step more precision is needed.
As a consequence the real values are used (the weights are binarized only during the first two steps). Tests on datasets like MNIST, CIFAR-10 and SVNH can achieve state-of-the-art results with two-thirds less multiplications, training time accelerated by a factor of 3 and a memory requirement decreased by at least 16.
In a binary weight network, only weight values are approximated with binary values. This also works on CNNs where the models are significantly smaller (up to 32 times). Then, the operation of convolution can be simplified as follows :

$$I * W \approx (I \oplus B)\alpha \ , \tag{3}$$

where, I is the input, W the real-value weight filter, B

Technique	Method	Pros	Cons
Knowledge distillation	Using a deep CNN to train a smaller CNN.	Small models with comparable performances.	Models can only be trained from scratch ; Difficult for the tasks other than classification.
Hashing	Indexing neurons into a hash table.	Better parallelization ; Better data dispersion ; Less computation time.	Considerably slower if the model is too sparse.
Pruning	Deleting neurons that have minor influence on the performance.	Significant speed up and size reduction ; Compression rate is 10x to 15x (up to 30x).	Pruning process is time consuming ; Less interesting for too sparse model.
Quantization	Reducing the number of distinct neurons by gathering them into clusters.	High compression rate : 10x to 15x ; Can be coupled with pruning.	Considerably slower if the model is too sparse.
Numerical Precision	Decreasing the numerical precision of the neurons.	High compression rate and speed up.	Higher precision is needed during the parameters update ; Could require specific hardwares.
Binarization	Decreasing the numerical precision of the data to 2 bits.	Very high compression rate (30x) and speed up (50x to 60x).	Higher precision is needed during the parameters update.

TABLE 1 – Summary of different compression methods.

the binary filter ($sign(W)$), α a scaling factor such that $W \approx \alpha B$ and $\oplus$ indicates a convolution without multiplications. Further improvements have been done with the XNOR-Net proposed by Rastegari et al. [50] where both the weights and the input to the convolutional and fully connected layers are binarized. In this case, all the operands of the convolutions are binary, and thus the convolution can be performed by only XNOR and bitcounting operations :

$$I * W \approx (sign(I) \oplus sign(W)) \odot K\alpha \ , \qquad (4)$$

where, I is the input, W is the real-value weight filter and K is composed of the scaling factors for all sub-tensors in the input I.
The resulting network in [50] is as accurate as a single-precision network. It also runs faster (58 times on GPU) and is smaller (AlexNet is reduced to 7MB). Many existing models (like the hourglass model [49]) have been enhanced with the XNOR-Net method to achieve state-of-the-art results [8]. Recently, the XNOR-Net method has been studied to be transformed from a binarization task to a ternarization task [17]. Values are constrained in a ternary space -1, 0, +1. It allows to remove the need for full-precision values during the training by using a discretization method.

2.7 Discussion

In Table 1, we summarized and compared various deep-learning model compression methods from the literature discussed in this paper. These methods aim to reduce the size, computation time or the memory employed by deep models. However, there is no golden rule for which methods works best. Pruning and quantization can be utilized to achieve impressive performances on trained models. However, a sparse model may not always be computationally efficient. In this case, binarization or reducing the numerical precision method can be one of the solutions. The speed gained for limiting the numerical precision is important, especially if the structure is well designed. Nevertheless, higher precision is needed in some steps and accuracy could vary significantly. In the end, compressing a deep model will always lead to a trade-off between accuracy and computational efficiency.

3 Conclusion

Computation efficiency remains a constraint for deep learning algorithms. There are no universal methods to optimize a deep model because every model is specific and is designed for a peculiar application. Further work needs to be done in this direction to leverage all of their power on mobile devices. Algorithmic optimisations like [46] and recent works such as Mobilenet[32] and Shufflenet[65] have shown that it is promising to not only compress models but also to construct them intelligently. Thus a well-designed architecture is the first key to optimized networks. Works like Neural Architecture Search, which aims to construct and search the best architectural design of network topology, is the next step towards optimized models.

References

[1] Abadi, M. : TensorFlow : Large-scale machine learning on heterogeneous systems (2015). Software available from tensorflow.org

[2] Ahmad, J., Beers, J., Ciurus, M., Critz, R., Katz, M., Pereira, A., Pringle, M., Rames, J. : iOS 11 by Tutorials : Learning the New iOS APIs with Swift 4, 1st edn. Razeware LLC (2017)

[3] Alvarez, J.M., Salzmann, M. : Learning the number of neurons in deep networks pp. 2270–2278 (2016)

[4] Anwar, S., Hwang, K., Sung, W. : Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (2017)

[5] Ba, J., Caruana, R. : Do deep nets really need to be deep ? pp. 2654–2662 (2014)

[6] Ba, J., Frey, B. : Adaptive dropout for training deep neural networks pp. 3084–3092 (2013)

[7] Buciluǎ, C., Caruana, R., Niculescu-Mizil, A. : Model compression pp. 535–541 (2006)

[8] Bulat, A., Tzimiropoulos, G. : Binarized convolutional landmark localizers for human pose estimation and face alignment with limited resources (2017)

[9] Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y. : Compressing neural networks with the hashing trick pp. 2285–2294 (2015)

[10] Chen, Y., Wang, N., Zhang, Z. : Darkrank : Accelerating deep metric learning via cross sample similarities transfer (2017)

[11] Choi, Y., El-Khamy, M., Lee, J. : Towards the limit of network quantization (2016)

[12] Courbariaux, M., Bengio, Y., David, J.P. : Training deep neural networks with low precision multiplications (2014)

[13] Courbariaux, M., Bengio, Y., David, J.P. : Binaryconnect : Training deep neural networks with binary weights during propagations pp. 3123–3131 (2015)

[14] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y. : Binarized neural networks : Training deep neural networks with weights and activations constrained to+ 1 or-1 (2016)

[15] Cun, Y.L., Denker, J.S., Solla, S.a. : Optimal Brain Damage. Advances in Neural Information Processing Systems pp. 598–605 (1990)

[16] Dauphin, Y.N., Bengio, Y. : Big neural networks waste capacity (2013)

[17] Deng, L., Jiao, P., Pei, J., Wu, Z., Li, G. : Gated xnor networks : Deep neural networks with ternary weights and activations under a unified discretization framework (2017)

[18] Duchi, J., Hazan, E., Singer, Y. : Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research pp. 2121–2159 (2011)

[19] Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., Lecun, Y. : NeuFlow : A runtime reconfigurable dataflow processor for vision. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (2011)

[20] Gionis, A., Indyk, P., Motwani, R. : Similarity Search in High Dimensions via Hashing. Proceedings of the 25th International Conference on Very Large Data Bases pp. 518–529 (1999)

[21] Gokhale, V., Jin, J., Dundar, A., Martini, B., Culurciello, E. : A 240 G-ops/s mobile coprocessor for deep neural networks. IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops pp. 696–701 (2014)

[22] Gong, Y., Liu, L., Yang, M., Bourdev, L. : Compressing deep convolutional networks using vector quantization (2014)

[23] Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P. : Deep Learning with Limited Numerical Precision. Proceedings of the 32nd International Conference on Machine Learning pp. 1737–1746 (2015)

[24] Hammerstrom, D. : A VLSI architecture for high-performance, low-cost, on-chip learning. IJCNN International Joint Conference on Neural Networks pp. 537–544 (1990)

[25] Han, S., Mao, H., Dally, W.J. : Deep Compression - Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ICLR pp. 1–13 (2016)

[26] Han, S., Pool, J., Tran, J., Dally, W. : Learning both weights and connections for efficient neural network pp. 1135–1143 (2015)

[27] Hassibi, B., Stork, D.G. : Second order derivatives for network pruning : Optimal brain surgeon pp. 164–171 (1993)

[28] He, K., Sun, J. : Convolutional neural networks at constrained time cost pp. 5353–5360 (2015)

[29] Hinton, G., Vinyals, O., Dean, J. : Distilling the Knowledge in a Neural Network. NIPS 2014 Deep Learning Workshop pp. 1–9 (2014)

[30] Hinton, G.E., Srivastava, N., Swersky, K. : Lecture 6a- overview of mini-batch gradient descent. COURSERA : Neural Networks for Machine Learning p. 31 (2012)

[31] Holt, J.L., Hwang, J.N. : Finite precision error analysis of neural network hardware implementations. IEEE Transactions on Computers pp. 281–290 (1993)

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

[33] Huang, Q., Feng, J., Zhang, Y., Fang, Q., Ng, W. : Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proceedings of the VLDB Endowment pp. 1–12 (2015)

[34] Huang, Z., Wang, N. : Like what you like : Knowledge distill via neuron selectivity transfer (2017)

[35] Iwata, A., Yoshida, Y., Matsuda, S., Sato, Y., Suzumura, Y. : An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors. Proc. IJCNN pp. 171–175 (1989)

[36] Kingma, D., Ba, J. : Adam : A method for stochastic optimization (2014)

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

[38] Lebedev, V., Lempitsky, V. : Fast convnets using group-wise brain damage pp. 2554–2564 (2016)

[39] LeCun, Y., Bengio, Y., Hinton, G. : Deep learning. Nature pp. 436–444 (2015)

[40] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. : Gradient-based learning applied to document recognition. Proceedings of the IEEE pp. 2278–2324 (1998)

[41] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P. : Pruning filters for efficient ConvNets. ICLR pp. 1–10 (2017)

[42] Lin, J., Rao, Y., Lu, J., Zhou, J. : Runtime neural pruning pp. 2178–2188 (2017)

[43] Lin, X., Zhao, C., Pan, W. : Towards accurate binary convolutional neural network pp. 344–352 (2017)

[44] Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C. : Learning Efficient Convolutional Networks through Network Slimming. ICCV pp. 2736–2744 (2017)

[45] Luo, J.H., Wu, J., Lin, W. : Thinet : A filter level pruning method for deep neural network compression (2017)

[46] Mamalet, F., Garcia, C. : Simplifying convnets for fast learning. ICANN 2012 pp. 58–65 (2012)

[47] Mamalet, F., Roux, S., Garcia, C. : Real-time video convolutional face finder on embedded platforms. Eurasip Journal on Embedded Systems (2007)

[48] Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J. : Pruning convolutional neural networks for resource efficient transfer learning (2016)

[49] Newell, A., Yang, K., Deng, J. : Stacked hourglass networks for human pose estimation pp. 483–499 (2016)

[50] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A. : Xnor-net : Imagenet classification using binary convolutional neural networks pp. 525–542 (2016)

[51] Roux, S., Mamalet, F., Garcia, C., Duffner, S. : An embedded robust facial feature detector. Proceedings of the 2007 IEEE Signal Processing Society Workshop, MLSP pp. 170–175 (2007)

[52] Shinde, R., Goel, A., Gupta, P., Dutta, D. : Similarity search and locality sensitive hashing using ternary content addressable memories pp. 375–386 (2010)

[53] Shrivastava, A., Li, P. : Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips) pp. 2321–2329 (2014)

[54] Simonyan, K., Zisserman, A. : Very deep convolutional networks for large-scale Image recognition. CoRR pp. 1–14 (2015)

[55] Spring, R., Shrivastava, A. : Scalable and sustainable deep learning via randomized hashing pp. 445–454 (2017)

[56] Srivastava, N. : Improving Neural Networks with Dropout. Master's thesis (2013)

[57] Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. : Dropout : a simple way to prevent neural networks from overfitting. Journal of machine learning research pp. 1929–1958 (2014)

[58] Sundaram, N., Turmukhametova, A., Satish, N., Mostak, T., Indyk, P., Madden, S., Dubey, P. : Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. Proceedings of the VLDB Endowment pp. 1930–1941 (2013)

[59] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. : Going deeper with convolutions pp. 1–9 (2015)

[60] Vanhoucke, V., Senior, A., Mao, M.Z. : Improving the speed of neural networks on cpus. In : Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011 (2011)

[61] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J. : Feature hashing for large scale multitask learning pp. 1113–1120 (2009)

[62] Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H. : Learning structured sparsity in deep neural networks pp. 2074–2082 (2016)

[63] Williamson, D. : Dynamically scaled fixed point arithmetic. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing Conference Proceedings pp. 315–318 (1991)

[64] Zeiler, M.D. : ADADELTA : An Adaptive Learning Rate Method p. 6 (2012)

[65] Zhang, X., Zhou, X., Lin, M., Sun, J. : Shufflenet : An extremely efficient convolutional neural network for mobile devices (2017)

[66] Zhou, H., Alvarez, J.M., Porikli, F. : Less Is More : Towards Compact CNNs, pp. 662–677. Springer International Publishing, Cham (2016)