Artificial Intelligence Review (2020) 53:5113–5155
https://doi.org/10.1007/s10462-020-09816-7

A comprehensive survey on model compression and acceleration

Tejalal Choudhary[1] · Vipul Mishra[1] · Anurag Goswami[1] · Jagannathan Sarangapani[2]

Published online: 8 February 2020

Abstract
In recent years, machine learning (ML) and deep learning (DL) have shown remarkable improvement in computer vision, natural language processing, stock prediction, forecasting, and audio processing to name a few. The size of the trained DL model is large for these complex tasks, which makes it difficult to deploy on resource-constrained devices. For instance, size of the pre-trained VGG16 model trained on the ImageNet dataset is more than 500 MB. Resource-constrained devices such as mobile phones and internet of things devices have limited memory and less computation power. For real-time applications, the trained models should be deployed on resource-constrained devices. Popular convolutional neural network models have millions of parameters that leads to increase in the size of the trained model. Hence, it becomes essential to compress and accelerate these models before deploying on resource-constrained devices while making the least compromise with the model accuracy. It is a challenging task to retain the same accuracy after compressing the model. To address this challenge, in the last couple of years many researchers have suggested different techniques for model compression and acceleration. In this paper, we have presented a survey of various techniques suggested for compressing and accelerating the ML and DL models. We have also discussed the challenges of the existing techniques and have provided future research directions in the field.

Keywords Model compression and acceleration · Machine learning · Deep learning · CNN · RNN · Resource-constrained devices · Efficient neural networks

✉ Vipul Mishra
vipul.mishra@bennett.edu.in

Tejalal Choudhary
tejalal.choudhary@gmail.com

Anurag Goswami
anurag.goswami@bennett.edu.in

Jagannathan Sarangapani
sarangap@mst.edu

1 Bennett University, Greater Noida, India

2 Missouri University of Science and Technology, Rolla, MO 65409, USA

1 Introduction

Deep learning is based on an artificial neural network (ANN), and it is part of a broader family of machine learning. A neural network with deeper layers (more than one hidden layer) is known as a deep neural network (DNN). Researchers have seen significant improvement in the accuracy of the DNNs in the last couple of years. In addition, the winning of the ImageNet (Deng et al. 2009) challenge by AlexNet (Krizhevsky et al. 2012) in 2012 also drawn a lot of attention from the artificial intelligence (AI) community. ML and DL have been applied successfully to solve various real-world problems related to text, image, audio and video such as image captioning (Xu et al. 2015), language translation (Sutskever et al. 2014), object detection (Girshick et al. 2014; Ren et al. 2015), speech recognition (Graves and Schmidhuber 2005; Graves et al. 2013), image generation (Goodfellow et al. 2014), classification (Krizhevsky et al. 2012). With the aid of the graphics processing units (GPUs), now we can process the parameters of a DNNs with many layers (He et al. 2016a). At the same time, the number of resource-constrained devices such as mobile phones and edge-devices are increasing every year. It is estimated that there will be 8.9 billion mobile subscriptions (Ericsson-Mobility-Report 2018) by 2024. However, the major constraints that restrict the deployment of the trained DNNs in Internet of Things (IoT) and edge-devices for real-time, on-device, fast inference are the limited computation power, storage capacity, and energy (Yang et al. 2017; Liu et al. 2018).

Previous research (LeCun et al. 1990) has verified that all the parameters of the neural networks (NNs) are not important, and makes NNs over-parameterized (Denil et al. 2013). One of the key challenges in DNN is reducing the computational cost and storage requirement of the trained model so that it can be deployed in resource-constrained devices (Zhu and Gupta 2017; Zhou et al. 2017a). Deployment of the DL model over the cloud has the advantage of having high computation and storage availability, but, the downside is the poor throughput and high response time (Li et al. 2019). However, there is a demand to move the inference step from cloud to edge-devices for real-time applications such as object detection from videos, segmentation, and so forth (Lin et al. 2019; Verhelst and Moons 2017). The current capabilities of the edge-devices restrict the DL model inference for real-time applications. In addition, transferring the data over the network requires more energy than processing it locally because network transmission is expensive. Moreover, mobile computing has a wide range of applications available in every domain (Yuan et al. 2014; Vu et al. 2016). For small devices, it is useful to process the data/images locally on the device rather than sending them to a server for processing, fetching the results, and then making the decision. Before deploying a DL model into production, a DL model needs to be trained on a large dataset. Training a DL model is a time-consuming process and requires GPUs to speed up the training. The variants of popular DL model VGG trained on the ImageNet dataset took 2–3 weeks to train, depending on the network architecture (Simonyan and Zisserman 2015).

Many real-world applications require on-demand decision-making ability like human beings. For instance, a driver-less car (Chen et al. 2015a) must detect different–different objects on the road such as pedestrians, other cars, animals and other vehicles, calculating the distance of distinct objects from the car. All this must occur in real-time locally to make the driver-less car successful. Thus, reducing the number of parameters and making a smaller model that can run under the constraints of the edge-devices is a key challenge. If the number of parameters are more in the trained model, then during inference it will take more time to process the input, require more energy and space compared to a small network having less number of parameters (Liu et al. 2018; Han et al. 2015). So, reducing the number of parameters and removing the unimportant redundant connection can significantly improve the performance of the ML/DL model and will also save the computation time and energy. Due to the above-stated reasons, it becomes necessary to reduce the size of the trained model before deploying them into resource-constrained devices.

In ML and DL research community, model compression and acceleration has received much attention from the researchers and has already gained a lot of progress in the last couple of years. In this paper, we review the techniques, methods, algorithms proposed by various researchers to compress and accelerate the ML and DL models. We have presented a perceptive performance analysis, pros and cons of popular DNN compression and acceleration as well as explored traditional ML model compression techniques. The summary of the various compression and acceleration techniques covered are shown in Fig. 1. In DNN, we have summarized the work done for the compression of feed-forward NNs, CNNs and recurrent neural networks (RNNs) using different methods like pruning, quantization, low-rank factorization (LRF), and knowledge distillation (KD). We have also summarize and compare some of the popular compressed and efficient network architctures like SqueezeNet (Iandola et al. 2017), MobileNets (Howard et al. 2017), ShuffleNet (Zhang et al. 2018b), ESPNet (Mehta et al. 2018), DenseNet (Huang et al. 2017). In ML, various compression methods for Support vector machines (SVM), Decision trees, Random forests, and K-nearest neighbours (KNN) have been discussed in the paper.

The rest of the paper is organized into different sections. Section 2 discusses popular DNN compression and acceleration techniques for feed-forward NNs and CNNs. Section 3 discusses the contribution made by different researchers to compress and accelerate the RNNs. Section 4 contains important techniques suggested for the traditional ML model compression. Section 5 discusses popular efficient network architectures. The experiments performed with existing techniques are included in Sect. 6. Discussion and challenges are covered in Sect. 7. Finally, the conclusion and future directions are discussed in Sect 8.

2 DNN compression techniques

DNN (either fully-connected, CNN or RNN) is composed of a variety of layer types. CNNs like AlexNet, VGG16 (Simonyan and Zisserman 2015) can have both dense layers as well as convolutional layers. In the previous research, it is found that DNNs are storage

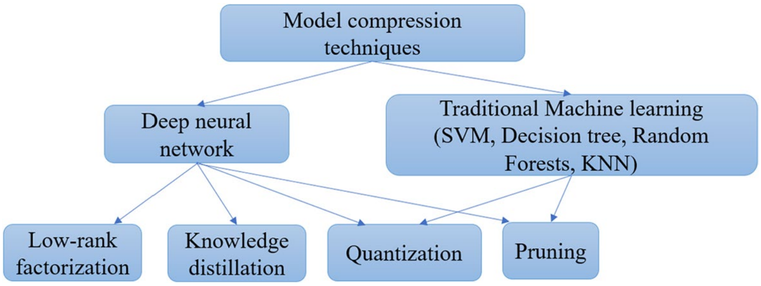
Model compression techniques
Deep neural network
Traditional Machine learning (SVM, Decision tree, Random Forests, KNN)
Low-rank factorization
Knowledge distillation
Quantization
Pruning

Fig. 1 Different types of compression techniques for DNN and traditional ML methods. Here, the arrow shows the types of techniques belonging to the upper-level hierarchy

and computation-intensive (Han et al. 2015). Most of the DNN parameters come from the dense layer and most of the computation time is consumed performing multiply-accumulate (MAC) operations in the convolutional layer (Yang et al. 2015). According to Li et al. (2017), in VGG16 model, the ratio of parameters in dense verses convolutional layer is 90:10 and 99% of MAC operations comes from convolutional layer. Reducing the number of parameters of the dense layer and MAC operations of convolutional layer can significantly reduce the storage and computation overheads of the DNN. In this section, different techniques proposed by researchers for lowering the dense layer parameters and MAC operations of feed-forward NN and CNN are elucidated.

2.1 Pruning

Pruning is a powerful technique to reduce the number of parameters of DNN (LeCun et al. 1990; Han et al. 2015; Li et al. 2017). In DNN, many parameters are redundant that do not contribute much during training to lower the error and generalize the network. So, after training, such parameters can be removed from the network, and the removal of these parameters will have the least effect on the accuracy of the network. The primary motive of pruning was to reduce the storage requirement of the DL model and make it storage-friendly. Pruning parameters from the dense layer helps in making the model smaller (Srinivas and Babu 2015; Ardakani et al. 2016). Pruning is also used to reduce the computation and speed-up the inference process by pruning parameters/filters from the convolutional layer. Pruning the filter reduces the number of MAC operations in the convolutional layer, and reduction in the MAC operations improves the inference time (Li et al. 2017). At the same time, if the model is small enough to fit into the on-board memory, there will be fewer memory accesses from DRAM requiring lesser power during inference (Han et al. 2015; Yang et al. 2017). Moreover, pruning also helps to reduce the overfitting problem of the NNs (Srivastava et al. 2014). In NN, there are several connections which contribute more towards reducing the network error than the other. Previous research (LeCun et al. 1990) has also verified that during the analysis of NN, all parameters or network connections are not equally important. Hence, eliminating lesser effect connections can result in a significant reduction of storage, computation cost, energy and inference time of a DNN model. A model can be pruned either during or after the training. Various techniques exists for pruning like weight pruning, neurons pruning, pruning of filters in the case of CNN, and pruning of layers.

- *Weight pruning* In unimportant weight connection pruning, we prunes (zeros out) the weight connections if they are below some predefined threshold (Han et al. 2015) or if they are redundant.
- *Neuron pruning* Instead of removing the weights one by one, which is a time-consuming process, we can also remove the individual neurons if they are redundant (Srinivas and Babu 2015). In that case, all the incoming and outgoing connection of the neuron will also be removed. There are many other ways of removing individual weight connections or neurons.
- *Filter pruning* In filter pruning, filters are ranked according to their importance, and the least important (least ranking) filters are removed from the network. The importance of the filters can be calculated by $L1/L2$ norm (Li et al. 2017) or some other methods like their influence on the error.

- *Layer pruning* Similarly, from a very deep network, some of the layers can also be pruned (Chen and Zhao 2018).

2.1.1 Pruning for fully-connected layers

Feed-forward NNs have only fully-connected (FC) also known as dense layers, in their simplest form they have input, hidden and output layers. Each input x_i is multiplied with corresponding weight w_i, and a linear sum is calculated at each neuron as:

$$y = \sum_{i=1}^{n} w_i x_i + b$$

where b is the bias and n is the number of inputs. The calculated sum is further processed by an activation function (i.e. sigmoid, ReLU, etc.) as $z = g(y)$, which converts the linear sum to non-linear and finally produce the desired output z. The structure of a feed-forward NN with 3, 2 and 1 neuron in the input layer, hidden layer and output layer respectively, and the candidate weight connection (Fig. 2a) and neuron (Fig. 2b) for pruning are shown in Fig. 2. It is clear from Fig. 2a that currently there are a total of 8 weight connection and if we remove two orange-colored (dashed) connections, then its total weight connection will be reduced to six. Similarly, from Fig. 2b, if we remove the red color neuron, then all its associated weight connections (dashed) will also be removed causing the total weight connections to be reduced to four (50% reduction in the number of parameters). In Fig. 2, $x1$, $x2$, $x3$ are the inputs to the network and $w_{ij}^{[l]}$ is the weights from layer l for the node i in the current layer to node j in the next layer. The pruning of the feed-forward network was first introduced by LeCun et al. (1990). The authors suggested the removal of the weights based on their saliency (if the removal of the weight parameter has a small effect on the training error). The authors elucidated that small magnitude weights (i.e. small saliency) will have less impact on the training error. Therefore, the effect on accuracy and training will be least if small-magnitude parameters are eliminated. To gain the accuracy loss, the network needs to be re-trained, and this process can be repeated several times until the network gets considerable accuracy again. To calculate the small saliency, the second derivative of the objective function is used with respect to the parameters. Their method is named as 'Optimal Brain Damage (OBD)'. The different steps of the algorithm are summarized in Fig. 3. Where, u_k is the weight parameter, s_k is the saliency, and h_{kk} is the second-order derivatives for the parameter (kth diagonal element of the Hessian matrix).

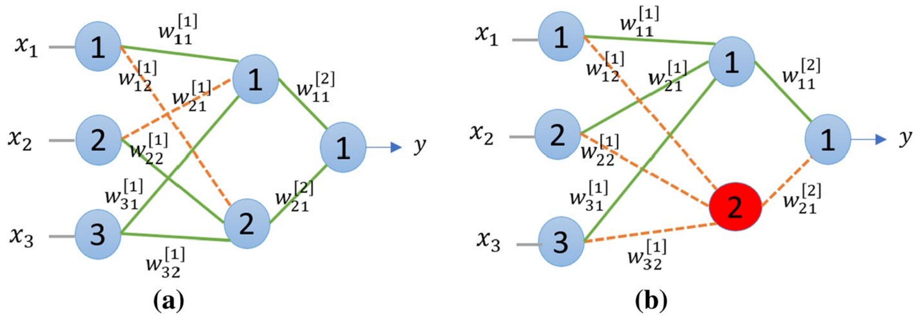

Fig. 2 Candidate weight (left) and neuron (right) pruning

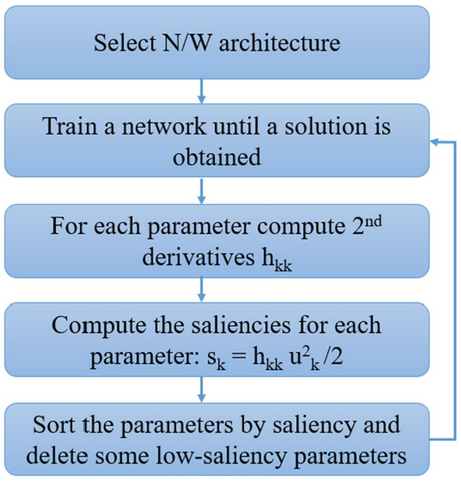
Select N/W architecture
Train a network until a solution is obtained
For each parameter compute 2nd derivatives h_{kk}
Compute the saliencies for each parameter: $s_k = h_{kk} u^2_k/2$
Sort the parameters by saliency and delete some low-saliency parameters

Fig. 3 OBD pruning algorithm (LeCun et al. 1990)

In another research, Hassibi and Stork (1993) proposed 'Optimal Brain Surgeon (OBS)' that extended the idea of OBD (LeCun et al. 1990) to prune the network. The authors claimed that eliminating small magnitude weights (LeCun et al. 1990) results in the elimination of wrong weight connections. Small magnitude weights can be necessary for lowering the error. According to the OBD (LeCun et al. 1990), the Hessian matrix is diagonal, which causes the removal of incorrect weights. However, according to OBS, the authors showed that the Hessian matrix is non-diagonal. The most important benefit of using OBS was that it does not require re-training after the pruning process. OBS reported better results than OBD, but OBS require more computation (Srinivas and Babu 2015). Moreover, Suzuki et al. (2001) remove the network connections based on their influence on the error. In their study, the connections units having the least influence are eliminated first, and then the network is re-trained using back-propagation (BP) algorithm. In another research (Srinivas and Babu 2015), the authors noticed that similar neurons are redundant, and instead of removing individual weight connections one by one, redundant neurons can be removed. The authors found that not only single weight being equal to zero create redundancy in the network, equal weights also create redundancies. If two weight sets $W1$ and $W2$ are similar, then one of them can be removed. A systematic approach was proposed for removing such neurons from the dense layers. The approach was different from OBD, where weights having the least effect on the training/validation error are removed one by one. Instead, here authors prune the weights which changes the output neuron activations the least, removing a set of weights at once. The process of finding and removing similar neuron is shown in Fig. 4. The circles in the figure are neurons, and the edges are the weights. The output is calculated as:

$$z = a_1 h\left(W_1^T X\right) + a_2 h\left(W_2^T X\right) + a_3 h\left(W_3^T X\right) + \cdots + a_n h\left(W_n^T X\right) \tag{1}$$

In Eq. (1), if $W1 = W2$ then

$$h\left(W_1^T X\right) = h\left(W_2^T X\right) \tag{2}$$

resulting equation will become,

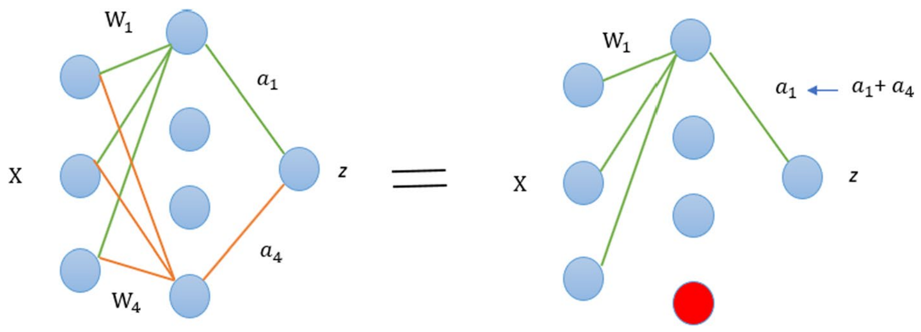

Fig. 4 The effect of equal weight-sets ($W1 = W4$). Same color weights constitute a weight-set (Srinivas and Babu 2015). (Color figure online)

$$z = (a1 + a2)h\left(W_1^T X\right) + 0h\left(W_2^T X\right) + a_3 h\left(W_3^T X\right) + \cdots + a_n h\left(W_n^T X\right) \tag{3}$$

$$z = (a1 + a2)h\left(W_1^T X\right) + a_3 h\left(W_3^T X\right) + \cdots + a_n h\left(W_n^T X\right) \tag{4}$$

To remove the second term from the Eq. (1), $a1$ is calculated as $a1 = a1 + a2$. Next, the neuron with red color, as shown in Fig. 4 can be removed. Moreover, to reduce the number of connections in the dense layer, Ardakani et al. (2016) proposed sparsely-connected networks. To make the network sparse, the authors randomly remove (zeros out) the connections from the dense layers. The connections are zeros out as per the random masks. To generate the random masks during sparsification, linear-feedback shift registers are used. The authors introduced mask matrix M, where each element of M is either 0 or 1 based on the value of linear-feedback shift registers. Using M, a new sparse weight matrix is defined as $W_s = W \cdot M$ (where W is the original weight matrix). Similar to fully-connected NN, the forward pass of the sparsely connected network can be redefined as $y = f(WsX + b)$. The authors also proposed efficient hardware architecture for sparsely-connected networks. Further, to reduce the memory requirements of the sparsely connected networks, the authors also recommended an efficient hardware architecture based on linear-feedback shift registers which results in 90% memory savings compared to conventional fully-connected NNs. In another research, Babaeizadeh et al. (2017) proposed NoiseOut, a simple and effective method for reducing the parameters in the FC layers. The removal of neurons from the network takes place based on the correlation among activations of neurons in the inner layers. The authors also proposed a technique to increase the correlation between neurons by adding extra output nodes (noise outputs). The higher correlation among the neurons will result in efficient pruning. The method can also be utilized for CNNs in which the FC layers are present.

There are several other attempts in which to reduce the number of parameters of the FC layer, the authors replace the FC layer with another type of layer. The work by Yang et al. (2015) replaces the FC layer with an Adaptive Fastfood transform followed by non-linearity. Adaptive Fastfood transform is a generalization of the Fastfood transform for approximating kernels (Le et al. 2013). In another work, Network in Network (Lin et al. 2013) and GoogleNet (Szegedy et al. 2015) replaces the FC layer with global average pooling layer. As there are no learnable parameters in global average pooling layer, no overfitting problem occur at this layer (Lin et al. 2013). On the other hand, the downside of the global average pooling layer is that it makes transfer learning difficult, to remove this limitation, Szegedy et al. (2015) added another linear layer so that the network can be used for the transfer learning task on another dataset. Although, the replacement of the FC layer with global average pooling layer helps in reducing the storage complexity of the model it also increases the computational overhead (Yang et al. 2015).

2.1.2 Pruning for convolutional layer

Convolutional networks such as LeNet (LeCun et al. 1998), AlexNet (Krizhevsky et al. 2012), VGG16 (Simonyan and Zisserman 2015) have the dense as well as the convolutional layers. CNNs are so far the most successful DNNs. In CNN, convolution is the pointwise multiplication of the filter with the input image. The main part of the CNN is the convolutional layer, which is used to extract the features from the input images. In convolutional network, a filter $W \in \mathbb{R}^{h \times w \times ic \times f}$ is applied to each input image I, $I \in \mathbb{R}^{m \times n \times ic}$ and produces the output feature map (activation) T, $T \in \mathbb{R}^{p \times q \times f}$. Where h and w are the dimensions of the filter, ic is the number of input channels in the input image, f is the number of filters applied, m, n are the dimension of the input image, p and q are the output dimensions of the resulting feature map. The shape of the output feature map is calculated as:

$$\frac{m - h + 2p}{s} + 1 \tag{5}$$

where s is the stride and p is the padding. Figure 5 shows the simplest form of CNN in which the size of the input image is $4 \times 4 \times 3$ and the size of the filters applied is $3 \times 3 \times 3 \times 2$ (2 is the number of filters). Generally, CNNs has many number of filters in each convolutional layer, pruning the unimportant filters from the convolutional layer directly reduces the computational overhead and accelerate the model. As noted by Li et al. (2017), more than 90% of the computation comes from the convolutional layer. Further, eliminating parameters from the FC layers will significantly reduce the storage overheads as FC layers are storage-intensive.

Inspired from the earlier pruning methods (LeCun et al. 1990; Hassibi and Stork 1993) and over parameterization (Denil et al. 2013) problem of the neural networks, Han et al.

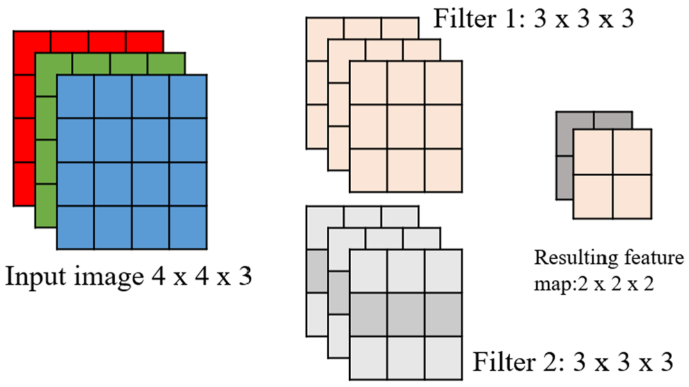

Fig. 5 Filters and feature maps in CNN

(2015) presented a simple magnitude-based three-step method to remove the unimportant connections. The idea was, first identify the important connections after that prune connection which are unimportant (prunes all the weight connections whose activation is less than some predefined threshold), and finally to compensate the loss in the accuracy due to the removal of weight connections, again fine-tune/re-train the pruned model. This process of pruning and re-training is repeated several times to reduce the overall size of the model and convert the dense network into a sparse network. The proposed method was able to prune both the FC and the convolutional layers. The authors also found that the convolutional layers are more sensitive to pruning compared to the FC layers. The problem with the proposed method was that the model structure is not preserved, and this is not supported by off-the-shelf libraries requiring some specialized resources (hardware and/or software) to inference the model. Guo et al. (2016) introduced dynamic network surgery to reduce the complexity of the network by connection pruning. The proposed method has two-part, pruning and splicing. In pruning, the unimportant weight connections are removed while in splicing, if any time the pruned or mistakenly pruned parameters found to be important, splicing enables the recovery of such connections. Pruning and splicing are performed iteratively. Different from the Han et al. (2015), which needs more than 4800K iterations to get a 9x compression rate, dynamic network surgery only requires 700K iterations to get 17.7x compression rate with comparable accuracy.

The systematic study by Li et al. (2017) showed that old magnitude-based techniques are not suitable for the pruning of CNNs. Authors proposed the pruning of convolutional layer filters if the filter has very less effect on the accuracy of the model. The authors rank the filters according to their $L1$-norm and pruned the low ranking filters from each layer. If the complete filter, along with the respective activation map will be removed from the network, then the size of the resulting model and computational cost will also be reduced. The proposed method uses one-shot pruning and re-training strategy to save the time during pruning of filters across layers and does not create the sparse network, which makes it work with the existing libraries and does not require special sparse convolution library or hardware. Using the proposed method, the authors reduces the number of MAC operations of VGG16 model from 3.13×10^8 to 2.06×10^8. In another research, Molchanov et al. (2017b) represents pruning as a combinatorial optimization problem, as shown in Eq. (6).

$$\min_{W'} \quad |C(D|W') - C(D|W)| \qquad \text{s.t.} \|W'\|_0 \leq B, \tag{6}$$

where D is the set of training examples, $C(D|W)$ is the cost value and B is the number of non-zero parameters in W'. To make the efficient inference of NN, the authors proposed an iterative pruning of the CNN filters. The proposed pruning method is shown in Fig. 6. The authors prune and fine-tune the network until a fine balance between desired accuracy and model size is achieved. A Taylor expansion criterion for efficient pruning of the network was also proposed. This criterion was based on approximating the change in loss due to the removal of the feature maps. To eliminate least important feature maps, in the proposed method approximating the absolute difference in the loss has been used, while in OBD (LeCun et al. 1990) the signed difference of loss is approximated, and it is found that the absolute difference-based pruning gives better accuracy as compared to the signed difference loss. Moreover, He et al. (2017) introduces a two-step method for pruning the channels/filters. First, the most representative channels are identified using the lasso (Tibshirani 1996) regression method and after identifying the most representative channels, prune the redundant channels. In the second step, the output is reconstructed with remaining channels with linear least squares.

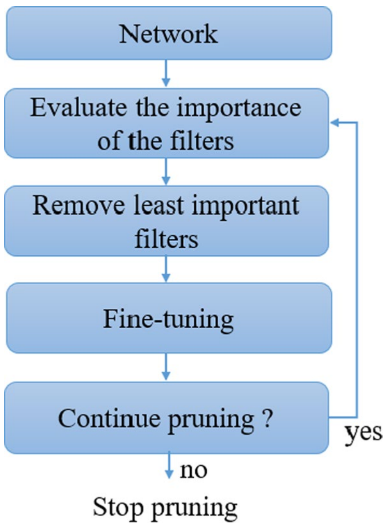
Network
Evaluate the importance of the filters
Remove least important filters
Fine-tuning
Continue pruning ?
yes
no
Stop pruning

Fig. 6 Network pruning (Molchanov et al. 2017b)

To reduce the redundancy in the convolutional network parameters, Liu et al. (2015) proposed a sparse decomposition. To explore the inter-channel and intra-channel redundancy of kernels, two-stage decompositions are applied. With less than 1% accuracy loss on the ImageNet dataset, sparse decomposition sparsify the 90% of the model parameters and converts the convolutional layer operation into sparse matrix multiplication. A new efficient matrix multiplication algorithm is also proposed to process the sparse kernels on CPUs efficiently.

Zhu and Gupta (2017) proposed a simple magnitude-based gradual pruning method that can be incorporated in training and require minimal tuning to achieve the preset level of sparsity. The results show that the proposed algorithm decreases the number of non-zero elements in the network by pruning small magnitude weights. This work is similar with another research (Narang et al. 2017), where authors pruned the RNNs and showed that the sparse RNN performed better than dense RNN of comparable size. In contrast, Zhu and Gupta (2017) worked with a variety of networks, from classification, language modeling, and sequence-to-sequence models. Based on their experimental results, the authors claim that large-sparse networks are better and can achieve the same or higher accuracy than small-dense models of comparable size. Furthermore, to automate the design of NN structures and for resource-constrained optimization of neural network architectures, Gordon et al. (2018) presented MorphNet. MorphNet can learn a structure that improves the performance at the same time, reducing the targeted resource usage. The authors focus on the reduction of a particular resource (the number of FLOPs per inference or the model size). To learn the architecture, initially, a seed network is given. MorphNet uses two hybrid approaches width multiplier and sparsifying regularizer and iterates alternatively between these two. To shrink the network, sparsifying regularizer is applied to neurons, and a width multiplier is used to expand the network.

The work by Luo et al. (2018) presented ThiNet, in which filter pruning is applied based on the statistics calculated in the next layer, not in the current layer. The method mainly consists of two steps pruning and post-processing, where other methods can further compress the pruned model. In post-processing, the authors proposed the group convolution with shuffling (GCOS) to further reduce the model size. ThiNet was different from deep compression (Han et al. 2016b), which lacks loss of universality and flexibility, making less practical for real-world applications. ThiNet reduces the size of AlexNet to 2.66 MB, making it embedded device friendly while preserving the original accuracy. The overall process of pruning and post-processing is shown in Fig. 7. In another work, Lin et al. (2017b) presented a framework called 'runtime neural pruning' for pruning the NN at runtime. The proposed method preserve the network ability and prune the NN as per the input and current feature maps. Whereas, existing methods produce a fixed pruned model for inference. To learn the best policy for pruning, the authors trained an agent based on reinforcement learning (Littman 2015), and each convolutional layer pruning is modelled as a Markov decision process (Puterman 2014). Preserving the original network ability helps in adjusting the balance point according to the available resources. Another advantage is that a single model can be adjusted for constrained-devices to large data centers. Moreover, the use of reinforcement learning in model compression is also explored by He et al. (2018), the authors introduce AMC (AutoML for model compression) and find that conventional model compression techniques are based on hand-crafted heuristics and rule-based policies requiring domain experts which is sub-optimal and time-consuming. AMC uses reinforcement learning to provide the model compression policy. AMC introduces two different policies, resource-constraint compression and accuracy-guaranteed compression for latency-critical applications and quality-critical applications, respectively. The work by Chen and Zhao (2018) presented a layer-wise pruning technique based on feature representation. The proposed method is different from earlier techniques where the parameters were pruned either connection-wise or filter-wise depending upon weight information, instead the proposed method uses the features learned in convolutional layer to identify the unimportant connections and then pruning is performed on layer-level. The overall process is shown in Fig. 8.

According to a research Frankle and Carbin (2019), every large dense, randomly-initialized, feed-forward network contains sub-networks, authors named them as 'winning tickets'. To train the sub-networks, the same initial weight initialization is required to get competitive performance. The limitation of the Frankle and Carbin (2019) is that the authors experimented only with image classification tasks and that also with only small datasets, i.e. CIFAR10 and MNIST, the proposed technique did not experiment with a larger dataset like ImageNet. Moreover, Frankle and Carbin (2019) and Liu et al. (2019) mentioned in their work that, in earlier pruning techniques more emphasis was on training,

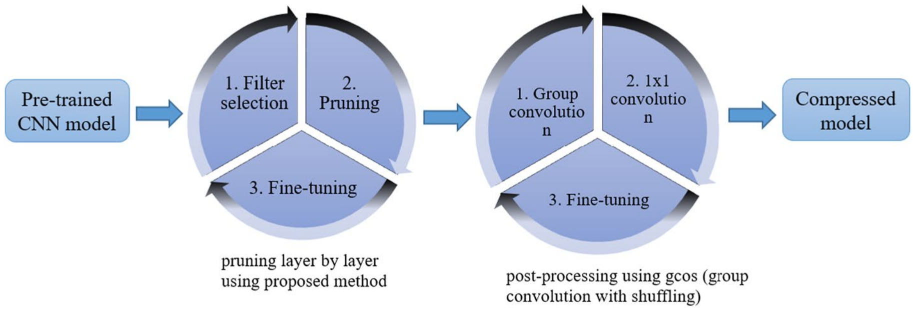

Fig. 7 The overall ThiNet pipeline. Given a pre-trained CNN model, it is pruned layer by layer using proposed filter selection method, followed by post-processing (Luo et al. 2018)

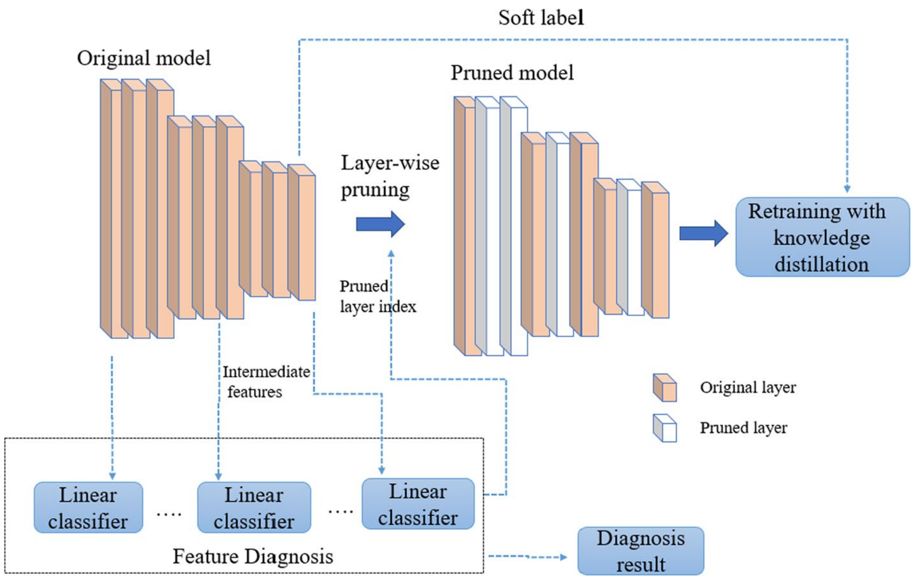

Fig. 8 Layer-wise pruning (Chen and Zhao 2018)

pruning and re-training pipeline. Previously, it was believed that pruned architecture and weights are more important for getting the final efficient model (Han et al. 2015).

However, Frankle and Carbin (2019) and Liu et al. (2019) claimed that, once we have the pruned architecture, it can be trained from scratch to get the equivalent or better performance compared to pruned and fine-tuned one. Both studies show the importance of pruning as a form of neural architecture search. Research Frankle and Carbin (2019) suggested that the same weight initialization should be used while training the pruned architecture from scratch that was used when training the original network. In contrast, the weights can be randomly initialized while training the pruned architecture from scratch in Liu et al. (2019). In addition, Crowley et al. (2018) applied $L1$-norm (Li et al. 2017) and Fisher (Theis et al. 2018) pruning on ResNets (He et al. 2016a) and DenseNets (Huang et al. 2017) architectures and found that for a given parameter budget, reduced networks (smaller version of the original network) trained from scratch perform better than large pruned and fine-tuned networks. The authors also reported that, if the pruned architectures are trained from scratch instead of fine-tuning the network after pruning, performs better than its pruned and fine-tuned equivalents. The experimental results of the authors also support (Frankle and Carbin 2019; Liu et al. 2019) pruning as a form of architecture search. Table 1 summarizes the comparison of various methods on the ImageNet (Deng et al. 2009) dataset and Fig. 9 shows the plot of the number of parameters and top5 accuracy. In Fig. 9, the blue color points show the plot of the method as per the parameters (M) and accuracy(%), and the lines connect the method name to the respective point. Where, PR: pruning, PQ: pruning and quantization, LRF: low-rank factorization. In addition, Table 2 summarizes the compression and acceleration results on the CIFAR10, CIFAR100 and MNIST dataset.

Table 1 Comparison of different DNN compression techniques on the ImageNet dataset and standard pre-trained models

Model	Actual para./size	Method	Para./size after compression	Compression achieved	Error top1:top5	Speedup achieved	Method type
AlexNet	61M/240 MB	Han et al. (2015)	6.7M	9×	42.77:19.67	3×	Pruning
		Guo et al. (2016)	3.45M	17.7×	43.09:19.99	–	Pruning
		Han et al. (2016b)	6.9 MB	35×	42.78:19.70	3×	Pruning and quantization
		Kim et al. (2016)	11M	5.46×	21.67	2.67×	LRF
		Tai et al. (2016)	12.2M	5×	20.34	–	LRF
		Yu et al. (2017)	6.1M	10×	19.69	–	LRF
VGG16	138M/512 MB	Han et al. (2015)	10.3M	13×	31.34:10.88	5×	Pruning
		Han et al. (2016b)	11.3 MB	49×	31.17:10.91	3× to 4×	Pruning and quantization
		Cheng et al. (2017)	28 MB	19.60×	–	4.94×	Quantization
		Tai et al. (2016)	50.2M	2.75×	9.69	2.05×	LRF
		Yu et al. (2017)	9.7M	15×	10.94	–	LRF
CaffeNet	60.9M	Srinivas and Babu (2015)	39.6M	1.5×	44.40	–	Pruning
ResNet34	21.6M	Li et al. (2017)	19.9M	1.09×	–	1.18×	Pruning
ResNet34	21.6M	Mishra and Marr (2017)	–	–	29.7		Quantization and KD
ResNet18	–	Zhou et al. (2017a)	–	–	31.02:10.90	–	Quantization

LRF Low-rank factorization, *KD* knowledge distillation

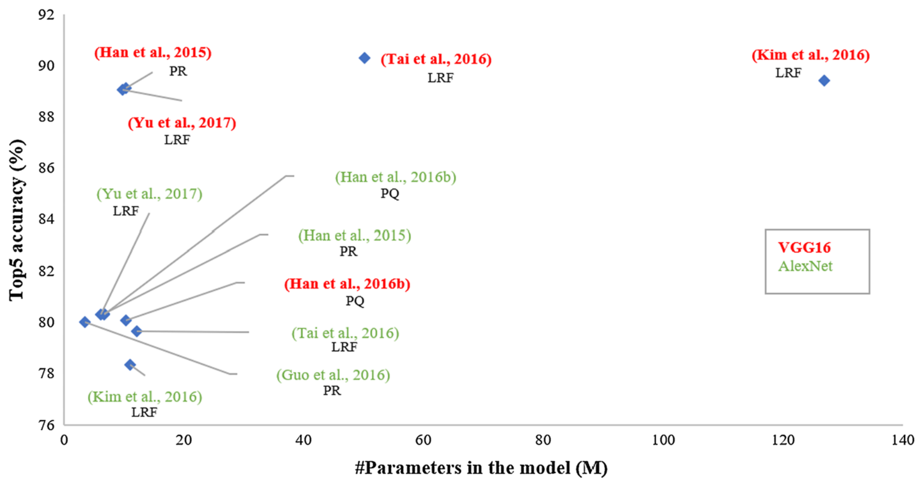

Fig. 9 The number of parameters and top5 accuracy of the AlexNet and VGG16 architecture on the ImageNet dataset after applying various compression methods listed in Table 1. The red (bold) label show the results for VGG16 model and green labels show the AlexNet model results. (Color figure online)

2.2 Quantization

In DNN, weights are stored as 32-bit floating-point numbers. Reducing the number of bits used to represent the weights and activations can lead to a significant reduction in the number of MAC operations required and reducing the size of the trained DNN. The idea of quantizing neural networks is not new, it was introduced earlier by Fiesler et al. (1990) and Balzer et al. (1991) to make the implementation of neural networks easier on hardware. Since the AlexNet won the ImageNet challenge in 2012, the use of quantization methods is again increasing. As the number of parameters increases in the DNN, the amount of storage required to store the weights and energy consumption also increases. Each 32-bit floating-point addition and multiplication operation requires 0.9 pJ and 3.7 pJ of energy respectively (Horowitz 2014). In quantization, we represent weights by reducing the number of bits required per weight to store each weight. This idea can also be further extended to represent gradient and activation in the quantized form. The weights can be quantized to 16-bit, 8-bit, 4-bit or even with 1-bit (which is a particular case of quantization, in which weights are represented with binary values only, known as weight binarization).

In addition, we can create the clusters of the weights, and all the weights which fall into that cluster can share the same weight value. In that case, we only need to fine-tune those shared weights instead of actual 32-bit full precision weights. If there are 16 original weights in the matrix and if we create four clusters, then each original 32-bit weight values can be quantized to 2-bit value as there would be four cluster indices only. This idea of weight sharing and quantization inspired the researchers to apply these concepts on different network architectures and reduce the storage, computation overheads and improve the inference speed of the DNNs. Besides, Different quantization schemes can lead to different acceleration performance. Uniform quantization (with uniformly-separated quantization points Li and Liu 2016; Zhu et al. 2017) can speed-up the inference phase on certain hardware, while non-uniform quantization (Chen et al. 2015b) can only reduce the model size. One of the main techniques affecting the performance of the quantized DNN

Table 2 Comparison of different DNN compression techniques on various datasets and pre-trained models

Method	Model	Dataset	Param. before	Param. after	Compression	Computation reduction	Method
Guo et al. (2016)	LeNet-5	MNIST	431K	4.0K	108×	–	Pruning
Li et al. (2017)	VGG16	CIFAR10	15M	5.4M	64.0% Pruned	3.13×10^8 to 2.06×10^8	Pruning
Chen and Zhao (2018)	VGG16	CIFAR100	–	–	78.9% Pruned	32.9%	Pruning
Romero et al. (2015)	–	CIFAR10	9M	2.5M	3.60×	1.52×	LRF
Li et al. (2017)	ResNet56	CIFAR10	0.85M	0.73M	13.7% Pruned	27.6%	Pruning
Chen and Zhao (2018)	ResNet56	CIFAR10	0.85M	0.49M	42.3% Pruned	34.8%	Pruning

LRF Low-rank factorization

is the batch normalization (Ioffe and Szegedy 2015). Batch normalization helps in reducing dependency across layer and at the same time, improves the model accuracy (Lin et al. 2016a). Although, the batch normalization was not invented for quantization optimization, it increases the efficiency of nearly all of the quantization techniques. Quantization can be applied during or after the training of the neural network. Generally, the floating-point NN model is developed and then quantized for efficient inference (Zhou et al. 2017a). Quantization after training is applied to reduce the inference time and save energy. While, quantization during training is applied to reduce the network size and make the training process more computational efficient (Courbariaux et al. 2015b). Based on this knowledge, we have categorized the quantization methods into two parts, quantization during training and quantization for efficient inference.

2.2.1 Quantization during training

The authors Soudry et al. (2014) presented expectation back-propagation (EBP) algorithm to support the training of multi-layer NN with discrete (i.e $+1$ to -1) or continuous (i.e. real numbers) weights. The authors found that traditional back-propagation (BP) or gradient descent-based methods for training NN with continuous or discrete weights are not suitable. The authors tested the EBP algorithm for eight binary text classification tasks and found that EBP performs better than BP. EBP trained multi-layer NN with binary weights outperforms continuous weights. By restricting the weight to continuous or discrete opens the possibility of implementation of multi-layer NN in small, constrained devices. The systematic study by Cheng et al. (2015) showed that binary multi-layer NN could be implemented on hardware more efficiently than real weights. Different from Soudry et al. (2014) where they use EBP algorithms for binary text classification and all the tasks were limited to high dimensional text dataset. Instead, the authors show that EBP can be used for multi-class image classification on MNIST (LeCun 1998) dataset. The effect of dropout was also evaluated and found that EBP works better with dropouts.

The concept of shared weight is explored by Chen et al. (2015b). The authors proposed HashedNets, in which the concept of a hash function was used to group the weight connection into hash buckets randomly, and all the weight connections which belong to the same hash bucket share the same parameter value. All the connections grouped to the ith hash bucket share the same weight value w_i. During training, these parameters are fine-tuned. The shared weight of each connection is determined by a hash function. HashedNets targeted the FC networks, but the authors claimed that the same idea can be extended to CNNs and RNNs. Several low-precision methods can also be applied on the resulting model to further reduce the model size. The authors assign V^l_{ij} an element of w^l indexed using hash function $h^l(i,j)$, as:

$$V^l_{ij} = w^l_{h^l(i,j)} \tag{7}$$

where the hash function $h^l(\cdot,\cdot)$ maps a key (i,j) to a natural number within $1,\dots,K^l$. Figure 10 shows the NN with random weight sharing where the compression factor is 0.25, in which six real weights represent 24 virtual weights. In Fig. 10, the elements that share the same weight values are represented by the same color. Using Eq. (7), $h^1(2,1) = 1$ and therefore $V^1_{2,1} = w^1 = 3.2$. There are 16 weight connections between input and hidden layer represented by (V^1), which are mapped to only three weights (W^1), while between hidden and output layer there are six weights (V^2) which are replaced by three real weights (W^2).

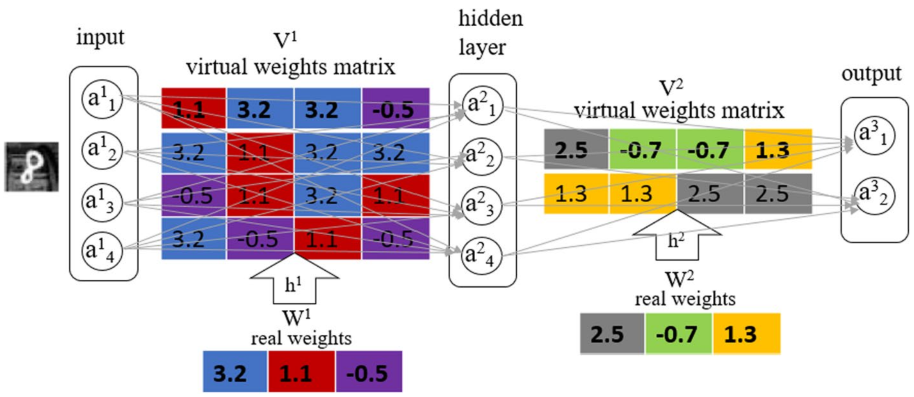

Fig. 10 NN with random weight sharing (Chen et al. 2015b)

Gong et al. (2015) applied vector quantization to compress the DNNs, which has an advantage over matrix factorization methods (covered in Sect. 2.4). Different from other methods where some of the researchers tried to compress the convolutional layer for speeding up DNNs, authors focus on dense layers to reduce the storage requirement of the model as dense layers are storage-intensive. For compression of the network, the authors applied three different quantization methods: scalar quantization using K-means, product quantization and residual quantization, and it was found that product quantization performs superior to other methods. The limitation of the proposed approach was that it can only quantize the FC layers and cannot be used with the convolutional layers.

In a systematic study (Courbariaux et al. 2015a), the authors trained NN with floating-point, fixed-point and dynamic fixed-point weights. Higher precision was used for the weights during the updates, and the results show that dynamic fixed points are well suited for the DNN training. The work by Courbariaux et al. (2015b) introduces BinaryConnect, in which weights are restricted to have only two values (−1 or +1). The advantage of using the binary weights was that many MAC operations can be replaced by simple accumulations. In BinaryConnect, the DNN was trained with binary weights during forward and backward propagations. BinaryConnect was different from Soudry et al. (2014) and Cheng et al. (2015), where authors use EBP to train the network instead of BP and does not works with convolutional networks. BinaryConnect transforms the real-valued weights into binary weights stochastically as:

$$w_b = \begin{cases} +1\, with\ probability\, p = \sigma(w), \\ -1\, with\ probability\ 1 - p \end{cases} \tag{8}$$

where w_b is the binarized weight, w is the real-valued weights and σ is the hard-sigmoid function. Moreover, Lin et al. (2016b) showed that many weights are zero or close to zero in a trained network. The authors extended the idea of Courbariaux et al. (2015b) to let the weight values to be lie in the interval of $[-1, 0]$ and $[0, +1]$ instead of $[-1$, and $+1]$ to eliminate all the multiplications in the forward pass as in BinaryConnect (Courbariaux et al. 2015b). In the proposed work, quantized backpropagation was introduced to remove the multiplications during the backward pass. The authors experimented with MNIST, CIFAR10 (Krizhevsky and Hinton 2009) and SVHN (Netzer et al. 2011) dataset and claimed state-of-the-art results. In another research, Hubara et al. (2016) proposed a binarized neural network (BNN) in which the weight and activations are constrained to have the values between +1 and −1. Different from Courbariaux et al. (2015b) where authors binarized the weights, however, here authors extended the idea to activations. Programmed optimized GPU kernel is proposed for binary matrix multiplication to run the proposed BNN on MNIST dataset 7× faster than unoptimized GPU kernel. BNN achieves state-of-the-art performance and speed-up on Theano (Al-Rfou et al. 2016) and PyTorch (Paszke et al. 2017) deep learning framework with MNIST, CIFAR10, and SVHN datasets.

In another work, Hou et al. (2017) noticed that earlier quantization techniques do not consider the quantization effect on loss. The authors introduced loss-aware binarization scheme that considers the effect of binarization on the loss. Loss-aware binarization outperforms other weight binarization methods including BinaryConnect (Courbariaux et al. 2015b), BNN (Hubara et al. 2016) and binary weight network (Rastegari et al. 2016). Inspired by Hou et al. (2017), Hou and Kwok (2018) further extend the idea of Hou et al. (2017) scheme to ternarization. More specifically, authors represent the effect of ternarization on loss as an optimization problem and solve the optimization problem using the proximal Newton method (Lee et al. 2014; Rakotomamonjy et al. 2015). The proposed loss-aware ternarization scheme outperforms other state-of-the-art methods.

In another study, Sung et al. (2015) observed some performance gap between floating-point and retraining-based ternary networks. The authors experimented with feed-forward deep NN and CNN on phoneme recognition and CIFAR10 dataset respectively by varying the complexity of the network (number of connections, feature maps and varying the number of layers). It was found that the performance difference between floating-point and quantized networks depends on the network size. The bigger models are resilient to quantization while the smaller networks are less resilient to quantization.

The work by Rastegari et al. (2016) introduced two different variations for approximating CNN; namely, binary-weight-networks (weight filter contain binary values, brings approx. 32× reduction in the network size) and XNOR-networks (input and filters both have binary values, brings approx. 58× speedup). The proposed method was different from Hubara et al. (2016) in the binarization method and the network structure. The proposed method was general and can be applied to any CNN based model. The authors applied the proposed method on the ImageNet dataset, and different network architectures AlexNet, ResNet and GoogleNet (Szegedy et al. 2015) are compared.

In another work, Zhou et al. (2016) proposed DoReFa-Net, for training CNN with low bit-width weights and activations with low bit-width gradients. DoReFa-Net was different from BNN (Hubara et al. 2016) and XNOR-Net [98], where weights were binarized, but gradients were still in full-precision, resulting in convolution between 1-bit numbers and 32-bit real-value. It was observed in DoReFa-Net that weights and activations can be quantized deterministically whereas gradients need to be quantized stochastically. Experiments on the ImageNet dataset showed that DoReFa-Net derived from AlexNet with 1, 2 and 6-bit weights, activations and gradients respectively can achieve 46.1% top1 accuracy on the validation set. In another research, to make the easy implementation of the CNN in hardware, Lin et al. (2017a) proposed binarized CNN with separable filters (BCNNwSF). To reduce the parameters and size of the CNN, BCNNwSF applies singular value decomposition (SVD) on BCNN kernels. The authors also compared the performance of the BCNN and BCNNwSF on FPGA hardware and found that BCNNwSF outperforms BCNN with some logic overhead. To represent the compact form of the network, Kim and Smaragdis (2018) proposed bitwise NNs in which the NN uses only bitwise operations XNOR to replace the MAC operations on binary weights and quantized input signals. An alternative to the convolutional layers in standard CNN, Juefei-Xu et al. (2017) proposed local binary convolution (LBC) layer with fixed binary weights. In addition, Bitwise NN proposed by Kim and Smaragdis (2016) uses binary values to represent everything, i.e. inputs, biases, weights and intermediate hidden outputs. The authors re-train the trained network using noisy BP for binarized NN.

The output of the ReLU is unbounded, requiring high bit-precision due to which the quantization of activation becomes difficult with the ReLU activation function. This problem can be solved by using a clipping activation function (Hubara et al. 2017; Zhou et al. 2016). Due to the difference among layers and models, it is difficult to find a global optimal clipping value. To solve this problem, Cai et al. (2017) applied half-wave Gaussian quantization scheme to activations. The authors noticed that after batch normalization, activation is near to Gaussian distribution. To find the optimized quantization scale for this Gaussian distribution, authors use Lloyd's (Lloyd 1982) algorithm and use that scale for all the layers. The disadvantage of the proposed approach is that, due to offline fixing of the quantization parameter during the whole training process, it fails to learn the clipping level optimally. In another work Choi et al. (2018), to find the optimal quantization scale during training, the author proposed a new activation scheme called PArameterized Clipping acTivation (PACT) function. To find the appropriate quantization scale, a new activation clipping parameter α is introduced and optimized during training. The value of α is adjusted dynamically with SGD based training, α limits the range of activation to [0, 1]. The conventional ReLU activation is replaced with:

$$y = PACT(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0, & x \in (-\infty, 0) \\ x, & x \in (0, \alpha) \\ \alpha, & x \in (\alpha, +\infty) \end{cases} \tag{9}$$

PACT reported the highest accuracy compared to previous work with 2-bit quantization and with 4-bit achieves similar accuracies to single-precision floating-point representation.

2.2.2 Quantization for efficient inference

Kim et al. (2014) presented VLSI based implementation of DNN for phoneme recognition in which they represented each weight with $+\Delta$, 0 or $-\Delta$. The authors used fixed-point optimization to represent each weight with 2 bits, and the authors conclude that VLSI based implementation of DNN is advantageous for resource-constrained devices. The study by Hwang and Sung (2014) extended the idea of Kim et al. (2014) where the authors employ retraining-based fixed-point optimization. Two sets of weight are equipped in the proposed method, one is the floating-point master weight and the second is the quantized version. Since the gradient values are small and they cannot easily alter the value of quantized weights, the method equips floating-point weights and re-quantize them for forward and backward propagation. The proposed method is considered to be a standard for all following quantization methods.

The work by Han et al. (2016b) proposed three-stage pipeline; pruning, quantization (multiple connections share the same weight) and Huffman coding to reduce the size of the pre-trained model. First, unimportant weight connections are removed, followed by weight quantization. After pruning and quantization network is re-trained to gain accuracy. Huffman coding is applied to the quantized model to reduce the size of the model further. The proposed three-stage pipeline is shown in Fig. 11. The size of the VGG16 model trained on the ImageNet dataset was reduced from 550 to 11.3 MB. Weight sharing was different from HashedNets (Chen et al. 2015b) where a hash function was used to group

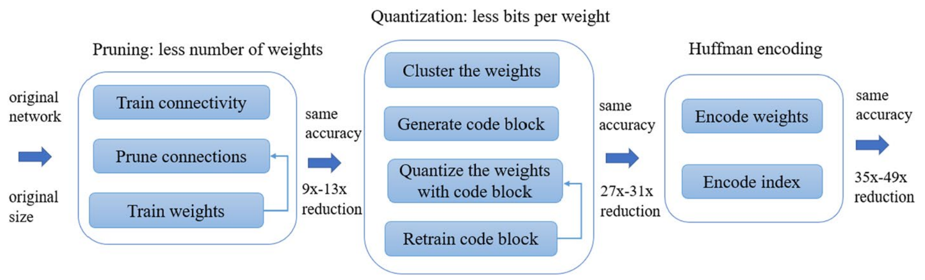

Fig. 11 Pruning, quantization and Huffman encoding (Han et al. 2016b)

the weights. Here instead, K-means clustering is used to form the cluster of the weights and all the weights which fall into that cluster share the same weights resulting in a significant reduction of bits used to represent the weights. Three different methods (forgy(random), density-based, and linear) for centroid initialization are suggested and it was found that linear initialization performs best among others. According to the authors, proper centroid initialization is important for clustering the weights because it affects the clustering quality and prediction accuracy. For updating the weights in the back-propagation phase, calculated gradient for each shared weight is used. The compression rate (r) is given by Eq. (10).

$$r = \frac{nb}{n\log_2(k) + kb} \tag{10}$$

where k is the number of clusters, n is connections, and each connection is represented by b bits. In another research, Gupta et al. (2015) trained DNN with 16-bit wide fixed-point and stochastic rounding. In another research, to reduce the binary network's accuracy loss, Li and Liu (2016) proposed ternary weight networks (TWN). To quantize weights into ($-W_l$, 0, $+W_l$), symmetric thresholds $\pm\Delta_l$ and for each layer l, scaling factor W_l is used as shown in Eq. 11.

$$w_l^t = \begin{cases} W_l : \tilde{w}_l > \Delta_l \\ 0 : |\tilde{w}_l| \leq \Delta_l \\ -W_l : \tilde{w}_l < -\Delta_l \end{cases} \tag{11}$$

After that, the authors solve the optimization problem of minimizing the $L2$ distance between the ternary and full precision weights to get layer-wise values of W_l and Δ_l as given in Eq. 12.

$$\begin{aligned} \Delta_l &= 0.7 \times E(|\tilde{w}_l|) \\ W_l &= \underset{i \in \{i | \tilde{w}_l(i)| > \Delta\}}{E} (|\tilde{w}_l(i)|) \end{aligned} \tag{12}$$

Ternary weights and scaling factors are required during inference, making resulting model 16× smaller than the original full precision model. Results show that the ternary model outperformed the original precision model on the ImageNet (Deng et al. 2009) dataset with ResNet (He et al. 2016a) model having 32, 44 and 56 layers. Moreover, to reduce the precision of the weight in NN to ternary values, Zhu et al. (2017) introduce trained ternary quantization (TTQ) in which during inference time, two-bit weights and scaling factors are needed. For AlexNet (Krizhevsky et al. 2012) trained on the ImageNet dataset, the proposed method performs better than Li and Liu (2016).

The work by Zhou et al. (2017b) showed that NN parameters are not uniformly distributed. To improve the prediction accuracy of the quantized NN, the authors proposed a balanced quantization method for the distribution of quantized values using histogram equalization. In addition, to accelerate and compress CNN simultaneously by minimizing the approximation error of the specific layer's response, Q-CNN (quantized CNN) is proposed by Cheng et al. (2017). Q-CNN can quantize both the FC and the convolutional layer successfully and reported 4× to 6× acceleration and 15× to 20× compression.

In addition, Anwar et al. (2017) introduces structured pruning of the convolutional layer (channel-wise, kernel-wise and intra-kernel strided sparsity) at different scales. The proposed method uses the particle filtering approach, also known as sequential Monte-Carlo in which the configurations are weighted by misclassification rate. Particle filter help in locating pruning candidates. After pruning fixed-point optimization (4-bit and 5-bit precision) is applied to further reduce the size of the model and make the model on-chip based implementation friendly for embedded devices. The proposed method produced good results with small CNNs, but, pruning of the deep CNNs is not addressed properly. In another research, Zhao et al. (2017) presented the binarized NN accelerator synthesized for FPGA based implementation. Hubara et al. (2017) proposed quantized NN for quantizing the weights and activations while training the model as well as during inference. To make the gradient updates using bit-wise operation, parameters are quantized to 6-bit. In the proposed method, the AlexNet quantized to 1-bit weights and 2-bit activations. While, RNN quantized to 4-bits and reported the same accuracy as of its 32-bit full precision model.

2.3 Knowledge distillation (KD)

In KD, the knowledge learned by the bigger cumbersome network (teacher model) trained on large dataset which can generalize well on unseen data is transferred to a smaller and lighter network known as a student model. The cumbersome model can be a single large model, or it can be an ensemble of separately trained models. The main objective of training a student model from the teacher model is to learn the teacher model's generalization capability while being lighter. KD makes a significant reduction in the number of parameters, multiplications required and make the model smaller. As discussed in the Sect. 2, DNN models are storage and computation intensive (Han et al. 2015). If we can have the smaller version of the model, which has the same/comparable performance as the larger model. In such cases, we can replace the larger model with a smaller model having a smaller model size and less computation. KD is useful when we need a smaller and less computational-intensive model. KD is different from transfer learning, where we use the same model architecture, learned weights and only replaces some of the FC layers with new layers as per the requirement of the application.

Earlier, knowledge transfer was first introduced by Buciluă et al. (2006), to compress the large, complex ensemble model into a smaller and faster models, without substantial loss in the performance. The authors claimed that the knowledge learned by the bigger ensemble model can be transferred to a smaller model. Model compression results on eight test problems reflect that compressed NN are 1K times smaller and faster with negligible performance loss. Moreover, Ba and Caruana (2014) extended the idea of Buciluă et al.

(2006) and empirically found that the complex functions learned by the larger DNN can be learned by the small shallow networks and can achieve the same accuracy as DNN with the same number of parameters. This can be done by first training a large DNN and then transferring the knowledge learned by the deeper NN to shallow NN. In the proposed method, the student model was trained with logits (logarithms of predicted probabilities of teacher model before softmax) and found to be better acquire the internal model learned by the teacher. The authors minimize the squared difference between the logits produced by the bigger model and logits produced by the smaller model.

Another systematic study (Hinton et al. 2015) showed that we should use a large, cumbersome model if it makes the extraction of features easier from the data. After that, the knowledge acquired by the cumbersome model can be transferred to a smaller model, as smaller models make deployment easier. The authors introduced the concept of temperature (T) to generate the soft targets and use the same temperature while training the smaller model from the larger model to match the soft targets produced by the two models. In DNNs, softmax layer is used to produce the class probability by converting logits z_i (of each class) into a probability q_i, Eq. 13. If $T = 1$, then it works as normal softmax function, higher values of T produces better softer probability distribution over classes. Hinton et al. (2015) also suggests that the use of the hard targets during training the smaller model along with the soft targets helps in generalizing the model. In this case, the objective function is the combination of the two objective functions (first, between the teacher model's soft targets and soft targets produced by the student model, and second between the actual hard targets and hard targets produced by the student model).

$$q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} \tag{13}$$

In FitNets, Romero et al. (2015) introduced a two-stage strategy to train the deep networks. In the proposed method, the teacher's middle layer provides 'hint' to guide the training of the deeper and thinner student model. FitNets was inspired by Hinton et al. (2015) and further extended the idea for a thinner and deeper student model. Apart from the outputs, the intermediate representations learned by the teacher model are provided as hints to improve the student model performance. Subsequently, Chen et al. (2016) in their work Net2Net proposed that knowledge transfer can significantly help in accelerating and saving the training efforts. They presented function-preserving-transformation based technique for transferring the knowledge of larger network into a smaller network. This makes it different from FitNets (Romero et al. 2015) which does not offer considerable speedup. Furthermore, Shen et al. (2016) utilized the KD and hint frameworks (hint layer) in learning a compact object detection model. For the detection of pedestrians, they represent object detection as a binary classification task. The distilled model was eight times faster and twenty one times smaller compared to the teacher model. Inspired by Shen et al. (2016), in another research, Chen et al. (2017) introduces KD based fast end-to-end trainable framework for multi-class object detection, which makes it different from Shen et al. (2016), where the proposed method was limited to single pedestrian detection only, making infeasible for multi-class object detection.

In a research Kim et al. (2018), a paraphrasing based knowledge transfer method is proposed. The proposed method uses convolution operations to paraphrase the larger model (teacher) knowledge and translate it to a smaller model (student). The proposed method was based on paraphraser and translator; two convolutional modules. The authors found that, if the teacher's knowledge is directly provided to a student model without any explanation, then a student cannot learn it properly. Instead, if the teacher's knowledge can be translated into simpler terms, a student network can learn more quickly. The paraphraser extracts the paraphrased information known as teacher factors. On the other hand, the translator extracts the student factors and helps to translate the teacher factors by mimicking them. The experimental results on ResNet, WideResNet and VGG16 architecture with CIFAR10, CIFAR100, and ImageNet dataset show that the proposed factor transfer method outperforms classical knowledge distillation and attention transfer (Zagoruyko and Komodakis 2017) methods.

In existing knowledge distillation methods, the learning of a student model relies on the teacher model and, it is a two-stage process. In a study, Lan et al. (2018) proposed On-the-fly Native Ensemble (ONE), an efficient and effective single-stage online knowledge distillation method. During training, ONE adds auxiliary branches to create the multi-branch variant of the target network, after which the native ensemble teacher model is created from all the branches. For the same target label constraints, ONE simultaneously learns student and each branch. Two loss terms are used to train each branch; usual softmax cross-entropy loss along with the distillation loss. Experiments with four classification datasets show that ONE can help in training more generalizable target network in one phase method. In another study, Srinivas and Fleuret (2018) proposed the matching of Jacobian (the gradients) to transfer the knowledge from the teacher model to a student model. Matching Jacobian is similar to soft-targets matching with noise added to the inputs during training. The authors applied Jacobian matching for knowledge distillation and transfer learning task. For the distillation task, authors trained a 9-layer VGG-like teacher model on CIFAR100 dataset and from that, smaller 4-layer student model is trained with Jacobian matching on a subset of the dataset instead of the full dataset. The experimental results show that when Jacobian matching is applied with regular cross-entropy and activation matching, it improves the performance of the student model.

In a research Wu et al. (2016), the author combined the knowledge distillation with quantization (discussed in Sect. 2.2) to train the binarized neural network and achieves a top5 classification accuracy of 84.1 on the ImageNet validation set which was better than Rastegari et al. (2016) and Hubara et al. (2017). The authors also agree with Hinton et al. (2015) that only training the network with soft targets does not improve the performance of the network. If the network is trained with regular targets along with soft targets that result in the improvement of network performance. It was also observed that slow learning rate during training helps in significant acceleration of convergence speed.

Moreover, to improve the performance of the low-precision networks, another research Mishra and Marr (2017) also applied knowledge distillation, they call their proposed method 'Apprentice'. By applying 3-bit and 4-bit precision on the variants of ResNets networks trained on the ImageNet dataset, Apprentice achieved new state-of-the-art accuracies. While selecting the depth for the student network, the depth of the teacher network was either same or larger. The only difference is that the teacher network has full-precision, whereas the student has low-precision. The authors proposed three different schemes. First, the low-precision student and full-precision teacher both networks are trained from the scratch using KD. In the second scheme, the distilled knowledge from trained full-precision teacher model is used to train the low-precision student network. It is found that using the second scheme, the student network converges faster. In the third scheme, both student and teacher networks are trained with full-precision, but student network is fine-tuned after reducing its precision. In addition, Polino et al. (2018) proposed two methods. First, quantized distillation in which the lower-precision student is trained with distilled knowledge from the teacher. Second, differentiable quantization, in which to better fit the behavior of the teacher model optimizes the quantized points through SGD. The authors experimented with different datasets and different classification and recurrent networks. The results of the study showed that the quantized small student networks can reach the same accuracy levels as that of full-precision larger teacher networks.

2.4 Low-rank factorization

In low-rank factorization, a weight matrix A with $m \times n$ dimension and having rank r is replaced by smaller dimension matrices. In feed-forward NN and CNN, singular value decomposition (SVD) is a common and popular factorization scheme for reducing the number of parameters. SVD factorize the original weight matrix into three smaller matrices, replacing the original weight matrix. For any matrix $A \in \mathbb{R}^{m \times n}$, there exists a factorization, $A = USV^T$. Where, $U \in \mathbb{R}^{m \times r}$, $S \in \mathbb{R}^{r \times r}$, and $V^T \in \mathbb{R}^{r \times n}$. S is a diagonal matrix with the singular values on the diagonal, U and V are orthogonal matrices. Each entry in the S is larger than the next diagonal entry. When reducing the model size is necessary, low-rank factorization techniques help by factorizing a large matrix into smaller matrices. The factorization of the dense layer matrices mainly improves the storage requirement and makes the model storage-friendly. While, the factorization of convolutional filters makes the inference process faster. This section contains the details of various approaches proposed using low-rank factorization that helps in reducing the storage and accelerating DNNs. Low-rank factorization can be applied to the FC layer weight matrices as well as the convolutional layer.

To reduce the number of parameters in the DNNs, Sainath et al. (2013) proposed low-rank matrix factorization of the final weight layer as most of the parameters lie in the final weight layer. Low-rank factorization reduces the number of parameters and improves the speed-up upto 30–50% compared to the full-rank matrix representation. The idea was, if the original weight matrix has the dimension $m \times n$ and rank r, then the full rank matrix can be factorized in two weight matrices as $m \times r$ and $r \times n$. In the proposed work, to reduce the number of parameters by a factor p, following holds

$$r < \frac{pmn}{m+n} \tag{14}$$

Rigamonti et al. (2013) approximate already trained CNN with low-rank filters. The authors proposed two different ways to approximate the CNN. Learning low-rank filters by $L1$-based regularization and using the decomposition of low-rank filters into several rank one kernels. The second approach found to be more successful than the first. Furthermore, Denton et al. (2014) uses a low-rank approximation to reduce the number of computations in the convolutional and the FC layer. The authors motivated from Denil et al. (2013), in which it was claimed that the weights within the layer can be approximated from a subset of them (approx. 5%). The proposed method was able to reduce the memory requirement of the weights in first two convolutional layer by 2–3 times and for the dense layers 5–13 times. In another research Jaderberg et al. (2014), the authors proposed tensor decomposition using SVD for CNN to reduce the number of parameters and speed-up the network. According to the authors, most of the computational time spent in first and second convolutional layer of the network. Moreover, Zhang et al. (2015) proposed a method for accelerating non-linear CNN. The experimental results show that the proposed method for accelerating the CNN found to be more accurate than Jaderberg et al. (2014) where the authors showed improved speed-up only for the first two convolutional layers.

To speed-up the convolutional layers of the larger CNNs, Lebedev et al. (2015) introduced a two-step method based on the tensor decomposition and discriminative fine-tuning. Non-linear least squares was used to compute low-rank CP-decomposition of the 4-D tensor into a sum of rank-one tensors. Using this decomposition, original convolutional layer is replaced by a sequence of 4 convolutional layers with smaller filters. Finally, fine-tuning is performed with backpropagation on the training data. The authors fine-tune the whole network based on the original discriminative criterion. In addition, Novikov et al. (2015) converts the FC layer weight matrices into tensor train representation reducing the number of parameters. The proposed work was similar to Lebedev et al. (2015) in some cases, but Lebedev et al. (2015) works with convolutional layer tensors instead of the FC layer matrices. Kim et al. (2016) introduced one-shot compression to compress the whole network, including the convolutional and the FC layers. The proposed approach consists of three steps, rank selection, low-rank tensor decomposition, and fine-tuning. To determine the rank of each layer kernel, variational Bayesian matrix factorization (VBMF) (Nakajima et al. 2012) was applied. In the proposed method, the authors applied tucker decomposition (Tucker 1966) for whole model compression. Also, instead of minimizing the non-linear response, the proposed method minimized the reconstruction error of the tensor.

To eliminate the redundancy from the convolutional filters (Tai et al. 2016) proposed a new algorithm inspired from Jaderberg et al. (2014) for computing the low-rank tensor decomposition. Empirical results show that low-rank tensor decomposition is helpful in speeding up the CNNs. The proposed method was different from Jaderberg et al. (2014) and Lebedev et al. (2015) in terms of the network size, earlier method experimented with smaller networks. To create computationally efficient CNNs, Ioannou et al. (2016) introduced a new low-rank representation-based method. The authors found that in earlier methods, filters were approximated in already trained networks, while in the proposed work these filters are learned from scratch during training. A novel weight initialization method was also introduced. In addition, Alvarez and Salzmann (2017) also suggests the low-rank decomposition of the filters during training instead of a pre-trained network. To encourage the low-rank value for each layer matrix parameters, the authors introduce regularizer and to achieve more compression, regularizer is further combined with sparsity-inducing.

One of the research Zhang et al. (2016) found that earlier methods of compressing and accelerating CNNs are only limited to shallow models, Zhang et al. (2016) proposed a method for compression and acceleration of very deep CNNs (number of layers > 10). The proposed method takes non-linear units into consideration instead of linear filters or linear responses for approximating the network. To solve the non-linear problem, without using stochastic gradient descent (SGD), the authors implemented a solution based on generalized SVD. In another systematic study Yu et al. (2017), a unified framework for the compression of DNN was proposed to integrate the low-rank and sparse decomposition of the weight matrix. The proposed method decomposes the original weight matrix into low-rank and sparse components.

Moreover, Li and Shi (2018) proposed a systematic method for finding an optimal low-rank approximation of trained CNN as constrained optimization based low-rank approximation (COBLA), constraining the number of MAC operations and memory. Taking the number of MAC operations and model storage size as constraints, the authors find the optimal low-rank approximation of the trained network that meets the specified constraints.

Recently Shi et al. (2018) combined low-rank factorization and quantization to compress the acoustic event detection (AED) models. The method was tested with long short-term memory (LSTM) network for AED and found that the method can compress the model to less than 1% of its original size. First, the NN is trained as usual with regular precision. After that, the low-rank factorization is applied, followed by the quantization and fine-tuning. Furthermore, Povey et al. (2018) represented a factored one-dimensional CNN, like time delay NN (Peddinti et al. 2015) (1-dimensional CNNs) in which the network layers were compressed using SVD but, trained by considering one of the factored matrices constrained to be semi-orthogonal. The authors also suggested other improvements like skip connections, shared dropout mask, and obtain state-of-the-art results.

Apart from the various compression and acceleration techniques discussed above, there are several hardware-oriented approaches proposed to acclerate the DL model. However, the problem with hardware-oriented approaches is that, the methods are limited to work with small models like AlexNet, and are not efficient for large models like VGG16, Inception, etc. After compression of the DL model, for efficient inference, Han et al. (2016a) implemented an efficient inference engine (EIE). To accelerate the DNN, EIE handles the weight sharing and performs sparse multiplication of matrices. Another systematic study by Ioffe and Szegedy (2015) introduced batch normalization to normalize layer inputs for accelerating the training of DNNs by reducing internal covariance shift between layers. The proposed method makes normalization a part of NN architecture, and apply normalization for each mini-batching.

3 RNN compression techniques

In recent years, many useful applications of RNNs has come like speech recognition (Graves and Schmidhuber 2005; Graves et al. 2013), language translation (Sutskever et al. 2014), text generation (Sutskever et al. 2011), sentiment analysis (Agarwal et al. 2011), natural language processing (NLP) (Collobert et al. 2011) to name a few. RNNs takes a longer time to train and require more memory and computational power than CNNs, which increases hardware and software implementation complexity. After getting the promising model compression and acceleration result on the feed-forward NNs and CNNs, researchers tried to reduce the complexity of the RNNs. Reducing the word-length of weights and signals can help in reducing hardware complexity. This section contains the details of the work done by the researchers to compress and acclerate the RNNs. Table 3 shows the comparison of the popular RNN compression and acceleration techniques.

The work in Ott et al. (2016) introduces limited numerical precision that can help in reducing weight precision for RNN. The authors worked with several stochastic and deterministic training methods for RNN and found that weight binarization does not work well with RNNs. The authors experimented with three RNN types: Vanilla RNN, LSTM (Collobert et al. 2011) and gated recurrent unit (GRU) (Cho et al. 2014) and found that limited numerical precision significantly reduces the computations and storage overhead of RNN and speed-up the training as well as inferencing process. In another research, Shin et al. (2016) proposed a retraining-based quantization method to optimize weights and signals word-length for fixed-point LSTM. The three main parts of the proposed method was floating-point training, sensitivity analysis, and re-training. The authors experimented

Table 3 Comparison of the different RNN compression and acceleration techniques

Task	Dataset	Method	Findings	Technique
Language modeling	PTB (Marcus et al. 1993)	Ott et al. (2016)	0.133 better BPC (bits-per-character)	Limited numerical precision
		Xu et al. (2018)	2-bit quantization: 16× reduced model and upto 6× speedup, and with 3-bit quantization: 10× reduced model and 3× speed-up	Multi-bit quantization (Both weights and activations)
		Lobacheva et al. (2017)	Achieved 87% sparsity	Sparse variational dropout
		Zhang et al. (2018a)	Achieved 6.7× compression and 6.04× speed up on mobile CPU	Fast dictionary learning algorithm
Speech recognition	LibriSpeech (Panayotov et al. 2015)	Zhang et al. (2018a)	Achieved 7.9× compression and 7.6× speed up on mobile CPU	Fast dictionary learning algorithm
	Speech data	Narang et al. (2017)	Achieved 8× reduction in network size and 2× to 7× speed-up	Weight pruning
	Speech data	Lu et al. (2016)	75% parameter reduction with only 0.3% increase in WER (word-error-rate)	LRF and parameter sharing
	Speech data	Narang et al. (2018)	10× reduction in model size	Block pruning and lasso regularization
Phoneme recognition	TIMIT (Garofolo et al. 1993)	Shin et al. (2016)	Requires only 10% weights of original model, and weights can be quantized to 3-bits without increasing the error rate	Retraining based quantization
Acoustic model compression	Hand-transcribed utterances	Prabhavalkar et al. (2016)	With negligible accuracy loss, compresses the model to one third of its original version	SVD-based factorization

with phoneme recognition and language modeling dataset and results show that reducing the word-length of weights and signals helps in reducing the model size and speed-up the LSTM. It was also found that cell signal, Ct, quantization demands a fairly large number of bits when compared to activation or weight quantization. Cell quantization usually demands 10-bit or more, while the activation and weight demands under 8-bit.

In another study Xu et al. (2018), the authors proposed a new multi-bit alternating quantization scheme for reducing the model size and speed-up the LSTM and GRU network. The authors achieved approximately 16× reduced model and 6× speed-up using 2-bit weights and activations while with 3-bit achieved approximately 10× reduced model and 3× speed-up, better than full precision. For RNN acoustic models, Prabhavalkar et al. (2016) proposed SVD-based factorization techniques for the compression of recurrent and non-recurrent (inter-layer) weight matrices. The authors successfully compressed the LSTM acoustic model to one-third of its original size. The proposed method can compress traditional RNN and LSTM. Reducing amounts of weight parameters reduce the size of the model, motivated by this, in a research Narang et al. (2017) proposed pruning of weights during training to make the weight matrix sparse. The experimental results show that pruning helps in reducing the model size, and the sparse matrix also reduces the inference-time. Another advantage of the proposed pruning method was that, it does not require re-training of the model because the model is pruned during the training. The authors used simple monotonically increasing threshold method to make the weight matrix sparse. Pruning resulted in about 90% reduction in the model and 2× to 7× speed-up. The proposed method worked with vanilla RNN and GRU. In addition, the pruned model can be processed by the quantization method to further reduce the model size. Inspired from Narang et al. (2017), Narang et al. (2018) proposed a method which during the training of the network zeros the blocks of weights in the matrix using lasso regularization. The proposed method helped in building 80–90% block-sparse RNN, causing 10× reduction in the model size and worked efficiently with the vanilla RNN and GRU.

Lobacheva et al. (2017) applied sparse VD (variational dropout) (Molchanov et al. 2017a), that was earlier applied to feed-forward NN and CNNs. The sparse VD generate the highly sparse model, on sentiment analysis task, the authors achieved 99.5% sparsity. While applying sparse VD on RNN, binary variational dropout (Gal and Ghahramani 2016) is used. Furthermore, Zhang et al. (2018a) introduced DirNet, a compression scheme for RNN based on the fast dictionary learning algorithm. Different from earlier methods which failed to adjust the compression rate as per the target requirement, DirNet can dynamically adjust compression rate for different layers. In another work, to make the easy implementation of RNN on hardware, Ardakani et al. (2019) introduced a training method which can learn binary and ternary weights. The proposed method was able to replace all MAC operations by accumulations. Replacing MAC with accumulations brings significant acceleration. LSTM with binary/ternary weights achieved 12× reduction in the model size and 10× speed-up compared to the full-precision network on Application-Specific Integrated Circuits (ASIC) platform. He et al. (2016b) proposed a method to quantize the gates structure, GRU, and LSTM cells interlinks. To lower the performance degradation, the authors proposed balanced quantization. In another research, a new strategy was proposed by Demeester et al. (2018) for the design of word embedding layers and networks with predefined sparseness. For efficient training of the RNN, Kusupati et al. (2018) proposed FastRNN and FastGRNN. Lobacheva et al. (2018) applied Bayesian sparsification approach for Gated RNN and sparsify individual weights of the RNN. Moreover, a new efficient sparse persistent RNN algorithm is proposed by Zhu et al. (2018) for the acceleration of the pruned RNNs.

4 Traditional ML algorithm compression techniques

This section contains the details of the techniques suggested to lower the complexity of popular ML algorithms. Li et al. (2001) apply dynamic programming for pruning of decision trees. Cost complexity-based pruning techniques were popular for decision tree pruning. The authors compare dynamic programming with cost complexity-based pruning techniques and found that dynamic programming-based pruning is superior in terms of classification accuracy. The limitation of dynamic programming pruning was that it work only with binary trees. The work by Shotton et al. (2013) showed that the memory requirement of conventional decision trees increases exponentially as the depth of the tree increases. To overcome this problem, the authors introduced ‘decision jungles’. Decision jungles reduces the memory requirement and help in improving the generalization. For efficient optimization of features and the directed acyclic graph (DAG) structure, authors introduce two new node merging methods. Besides, an optimal constrained pruning strategy is presented by Sherali et al. (2009) for decision trees. Authors presented a 0–1 programming approach with a variety of objective functions.

Moreover, for real-time prediction on resource-scarce devices Gupta et al. (2017) introduces ProtoNN. ProtoNN is inspired by KNN and can be deployed on devices having low computation power and small storage capacity like Arduino UNO boards with acceptable accuracy. ProtoNN uses the sparse low-dimension projection of dataset, prototypes and joint optimization to achieve the state-of-the-art results over existing techniques. The systematic study by Moshtaghi et al. (2011) introduced a robust, low computational complexity clustering algorithm hyperellipsoidal clustering for resource-constrained environments (HyCARCE). The main steps of the proposed algorithm are namely: initializing, cell pruning, expansion and re-adjusting, and removal of redundant ellipsoids. The results show that HyCARCE performs better than other existing methods and have better classification accuracy and low computational cost.

For efficient implementation of SVM on an 8-bit microcontroller (low-cost and low-power), a new algorithm was proposed by Boni et al. (2007) for model selection, which permits the fitting of the imposed constraints by the hardware. Experimental results show that 8-bit microcontroller SVM implementation have the same accuracy as the floating-point representation. In another work, pruning of random forests is proposed by Nan et al. (2016) for making a prediction in resource-constrained devices. The constructed random forests is pruned to optimize the accuracy and feature cost. The authors proposed a two-stage algorithm. First, the random forests is trained using impurity function like entropy. In the second stage, the trained random forests is taken as input and to satisfy the global resource-constraints, each tree in the forests is pruned. Furthermore, Joly et al. (2012) explores the compressibility of random forests and apply $L1$ norm-based regularization. The experimental results show that $L1$-norm regularization can leads to storage reduction without compromising accuracy. In another work, to deploy trained ML model into Arduino Uno board and similar resource-scarce devices for making local predictions (Kumar et al. 2017) suggests ‘Bonsai’, a tree-based algorithm. Bonsai can fit into a few KB of flash memory and on a slow microcontroller can predict in milliseconds. Bonsai was very similar to Shotton et al. (2013) and Jose et al. (2013).

5 Efficient network architectures

In the literature, several efficient network architectures have become popular due to their compact size and low computational requirement. SqueezeNet, developed by Iandola et al. (2017), is a small CNN based architecture. It has 50× less parameter than the AlexNet and achieves the same accuracy on the ImageNet dataset. Moreover, by applying a deep compression (Han et al. 2016b) technique, authors compress it to less than 0.5 MB, which is 510× smaller in size compared to the original AlexNet model. Howard et al. (2017) introduced MobileNets, an efficient architecture specifically for mobile devices. To create the light-weight DNN, MobileNets extends the idea of depth-wise separable convolutions (a combination of depth-wise and 1×1 (point-wise) convolution). To create the depth-wise separable filter, standard filters are replaced by two layers; depth-wise convolution and point-wise convolution. MobileNets also introduces two hyper-parameter width multiplier (α), which can reduce the computational cost and the number of network parameters. Second is the resolution parameter p, p is applied to the input image, and reduces the computational cost by an order of p^2. MobileNets is tested on various tasks like classification, object detection, and experimental results show the superiority of MobileNets over other existing methods. Recently, Sandler et al. (2018), introduces MobileNetsV2 with more improved results. Table 4 summarizes the number of parameters, FLOPs, error, accuracy and inference latency of popular, efficient network architectures on the ImageNet dataset as reported into their base papers. Zhang et al. (2018b) created a computational efficient CNN based architecture for mobile devices named ShuffleNet. To make the CNN model computational efficient, new operations are introduced; pointwise group convolution and channel shuffle. ShuffleNet achieves approximately 13× speed-up over AlexNet with similar accuracy. Tested on the ImageNet and the MS-COCO (Lin et al. 2014) object detection dataset, ShuffleNet reported better performance than the MobileNets (Howard et al. 2017). In addition, ShuffleNetV2 (Ma et al. 2018) further improves the network accuracy and efficiency by channel shuffle and split. In another research, Huang et al. (2017) introduced DenseNet, in which each layer of the network is connected to every alternate layer. DenseNet has several observable benefits like reduction in the number of parameters, feature-reuse and in lowering the vanishing gradient issue. DenseNet contains multiple dense blocks each dense block contains multiple layers. The experiments results with SVHN, CIFAR10, CIFAR100 and ImageNet datasets shows that DenseNet has performance improvements over other existing architectures.

Table 4 Comparison of different efficient network architectures on the ImageNet dataset

Model	#Para. (M)	FLOPs (M)	Error-rate (top1)	Accuracy (top1)	Inference latency
SqueezeNet	1.25	1700	42.5	57.5	–
MobileNets	4.2	569	29.4	70.6	113 ms
MobileNetsV2	3.4	300	28.12	72.0	75 ms
ShuffleNet	3.4	527	28.5	71.5	108 ms
ShuffleNetV2	5.3	292	26.3	73.7	–
DenseNet-201 (k = 32)	22	295	22.58	77.42	–
CondenseNet (G = C = 4, k = 32)	4.8	529	26.2	73.2	1.89 s

In another study, Mehta et al. (2018) introduces ESPNet; fast, low-power, smaller, low-latency network architecture for segmentation. Based on convolution factorization, a new efficient spatial pyramid (ESP) module is introduced. ESPNet decomposes the standard convolution into a spatial pyramid of dilated and point-wise convolutions. On similar constraints (memory and computation), ESPNet outperforms another factorization-based network architecture i.e. MobileNets (Howard et al. 2017) and ShuffleNet (Zhang et al. 2018b). In addition, ESPNetv2 (Mehta et al. 2019) network extends the ESPNet (Mehta et al. 2018) and reported better generalization ability than ShuffleNetV2. There are several other network architectures proposed for resource-constrained devices like SqueezeDet (Wu et al. 2017) for real-time object detection for autonomous driving, SlimNets (Oguntola et al. 2018), that combines several compression techniques like KD with pruning and achieves 85× smaller model than original model size. Another work by Huang et al. (2018) introduces CondenseNet; it uses one-tenth of the computation compared to DenseNet (Huang et al. 2017) with similar accuracy. Figure 12 shows the comaparison of various efficient network architectures listed in Table 4 on the basis of number of parameters and top1 accuracy on the ImageNet dataset.

We proposed a quality of model (QoM) metric in Eq. 15 based on the accuracy and number of parameters (weights) of the model, to evaluate the quality of the ith model. In Eq. 15, α and β are the weightage of model accuracy and parameters (in our case, we set α and β to 0.5), acc_i is the accuracy of the ith model, $para_i$ is the number of parameters of the ith model, $maxpara$ is the maximum of all the models parameters. Based on the proposed QoM, we have plotted the quality of all the efficient network architectures in Fig. 13 The architecture whose quality score is near to 1 is better compared to the other architectures.

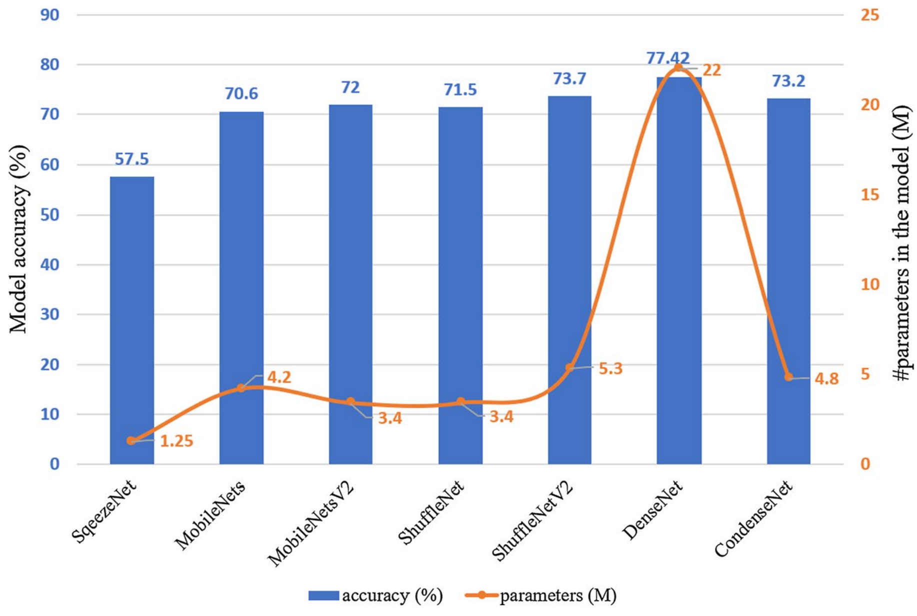

Fig. 12 Comparison of efficient network architectures on the ImageNet dataset: accuracy (%) versus number of parameters (M)

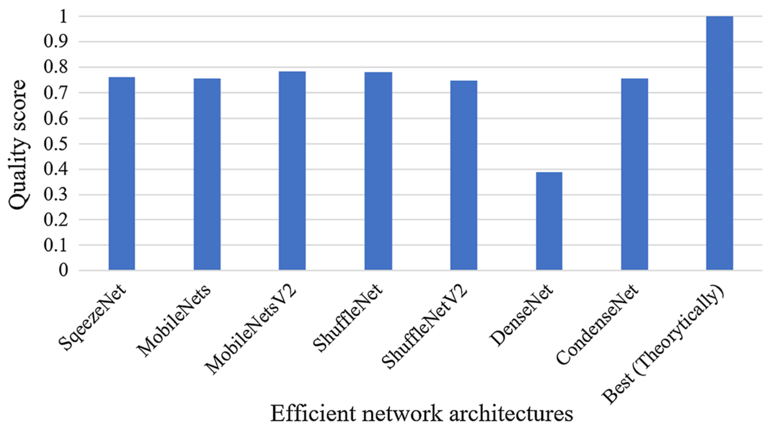

Fig. 13 Quality of different efficient network architectures, the quality of the architectures are plotted based on the proposed quality of model (QoM_i) metric given in Eq. 15

$$QoM_i = \alpha \frac{acc_i}{100} + \beta \frac{(1 - para_i)}{maxpara} \tag{15}$$

6 Experiments

6.1 Model and dataset

We performed experiments with existing pruning (weight and filter pruning) and quantization methods. For pruning, we perform all the experiments on AlexNet (Krizhevsky 2014) architecture. AlexNet has five convolutional layer and three fully-connected layers. The original AlexNet model was trained on the ImageNet (Deng et al. 2009) dataset and has 1000 nodes in the last layer. In our experiments, we modify the last layer of the AlexNet as per the number of classes in the CIFAR10 (Krizhevsky and Hinton 2009) and the MNIST (LeCun 1998) dataset. The CIFAR10 dataset has 50,000 training images and 10,000 test images. It has ten different classes, and the dimension of each colored image is 32×32. The training set includes 5000 images from each class, and the test set includes 1000 images from each class. On the other hand, the MNIST dataset has 60,000 training images and 10,000 test images (black and white, dimension: 28×28) of hand-written digits. The training set includes 6000 images from each class, and the test set includes 1000 images from each class. All the experiments are performed on the same DL framework, PyTorch (Paszke et al. 2017).

6.2 Results

Tables 5 and 6 shows the result of the experiments performed on the magnitude-based (Han et al. 2015) and structured (Li et al. 2017; Liu et al. 2017, 2019) pruning techniques. Table 7 shows the results of the quantization techniques (Hubara et al. 2016; Zhou et al. 2017a) on AlexNet, ResNet18 with CIFAR10 and ImageNet dataset. In Li et al. (2017), we

Table 5 Comparison of the experiments performed on different DNN pruning techniques on AlexNet (Krizhevsky 2014) architecture with CIFAR10 (Krizhevsky and Hinton 2009) and MNIST (LeCun 1998) dataset

Dataset	Method	Acc. before pru.		Acc. after pru.		# Para. (M)		FLOPs (M)		Filter/weights pruned
		Top1	Top5	Top1	Top5	Before	After	Before	After	
CIFAR10	$L1$-norm (Li et al. 2017)	74.0	97.61	73.92	97.55	2.87	1.64	92.21	56.78	30% filters pruned from all layers
		74.0	97.61	74.52	97.93	2.87	1.05	92.21	38.56	50% filters pruned from all layers
		78.45	98.45	76.48	98.40	2.87	0.99	92.27	31.19	First layer: 30%, other 50%, model with BND
		78.45	98.45	76.24	98.32	2.87	0.8	92.27	30.74	First layer: 0%, other 60%, model with BND
	Magnitude based (Han et al. 2015)	74.0	97.61	74.09	97.58	2.87	1.35	–	–	Conv: 60%, FC: 70%, model trained in $L1$-norm is used
		78.45	98.45	77.74	98.37	2.87	1.11	–	–	Conv: 60%, FC: 70%, model trained in $L1$-norm with BND is used
	N/W slimming (Liu et l. 2017)	77.91	98.30	77.93	98.29	2.47	0.75	89.60	48.49	Filters prunued: 40%
		77.08	98.43	78.22	98.44	2.47	0.47	89.60	41.18	Filters prunued: 50%
MNIST	$L1$-norm (Li et al. 2017)	98.68	99.99	98.53	99.98	2.85	1.03	86.26	32.61	First layer: 0%, other 50%
		98.58	100	98.63	100	2.85	0.98	86.33	27.1	First layer: 30%, other 50%, model with BND
		98.58	100	98.66	100	2.85	0.8	86.33	24.8	First layer: 0%, other 60%, model with BND
	Magnitude based (Han et al. 2015)	98.68	99.99	98.53	99.97	2.85	1.1	–	–	Conv: 60%, FC: 70%, model trained in $L1$-norm is used
		98.58	100	98.43	99.99	2.85	1.1	–	–	Conv: 60%, FC: 70%, model trained in $L1$-norm with BND is used
	N/W slimming (Liu et l. 2017)	98.72	99.99	98.63	100	2.45	0.74	83.91	37.10	Filters prunued: 40%
		98.72	99.99	98.77	99.99	2.45	0.47	93.91	28.9	Filters prunued: 50%

We have reported the top1 and top5 validation accuracy. BND: AlexNet with batch normalization and dropout layers

Table 6 Results of Liu et al. (2019) with Alexnet (Krizhevsky 2014) architecture

Dataset	Model	Top1 accuracy	Top5 accuracy	# Para. (M)	FLOPs (M)	Model used
CIFAR10	Base	78.45	98.45	2.87	92.27	$L1$-norm trained with BND
	Pruned	76.24	98.32	0.99	30.74	
	Scratch trained	76.22	98.18	0.99	30.74	
MNIST	Base	98.58	100.0	2.85	86.33	$L1$-norm trained with BND
	Pruned	98.66	100.0	0.80	24.80	
	Scratch trained	98.64	100.0	0.80	24.80	

We have reported the top1 and top5 validation accuracy. BND: AlexNet with batch normalization and dropout layers

Table 7 Quantization results with AlexNet and ResNet18 architecture

Dataset, Model	Method	Precision	Top1 accuracy	Top5 accuracy
CIFAR10, AlexNet	INQ (Zhou et al. 2017a)	32-bit	78.53	98.56
		8-bit	62.14	95.62
		5-bit	62.35	95.50
	BNN (Hubara et al. 2016)	32-bit	73.92	97.66
		1-bit	71.08	97.75
		1-bit, hinge loss	71.06	97.51
ImageNet, ResNet18	INQ (Zhou et al. 2017a)	32-bit	69.64	88.98
		8-bit	68.96	88.85
		5-bit	65.59	86.87
ImageNet, AlexNet	INQ (Zhou et al. 2017a)	32-bit	56.62	79.05
		8-bit	55.70	78.57
		5-bit	55.96	78.62

first train the AlexNet on CIFAR10 and MNIST dataset. After that, we rank all the convolutional layer filters as per their $L1$-norm and perform one-shot pruning of the low ranking filters. The pruned model is fine-tuned to compensate for the accuracy loss due to the removal of the filters. Two types of AlexNet variations are used, first, without dropout and batch normalization layer. Second, with dropout and batch normalization layers. A variety of pruning percentage is applied while pruning AlexNet on both MNIST and CIFAR10 dataset. In magnitude-based pruning (Han et al. 2015), we sort all the weights of the convolutional and dense layers by their magnitude and pruned (zeros out) the small-magnitude weights as per the threshold. In magnitude-based pruning, we use the model trained in Li et al. (2017) method as a base model for both MNIST and CIFAR10 experiments. Different pruning % for convolutional and dense layer was used. In network slimming (Liu et l. 2017), the AlexNet model with batch normalization layer is trained. The method has a scaling factor γ for each channel, which is multiplied with respective channel output. The model and scaling factors are trained with $L1$ sparsity regularizer. Filters with smaller scaling factors are pruned (40% and 50% in two different experiments) globally. Table 6 shows the pruning results of Liu et al. (2019). We took the models trained and pruned in Li et al. (2017) (with dropout and batch norm layer) in our experiments. After that, we use only pruned architecture and train the architecture from scratch using both CIFAR10 and MNIST dataset.

Table 7 shows the result of the incremental network quantization (INQ) (Zhou et al. 2017a) and binarized neural network (BNN) (Hubara et al. 2016). In INQ, AlexNet is trained on the CIFAR10 dataset with full 32-bit precision weights. After that, the trained model is quantized to 5-bit and 8-bit. We also test INQ on the ImageNet dataset. For the ImageNet experiment, we took the pre-trained AlexNet and ResNet18 architecture available in the Pytorch framework. Table 7 shows the result of quantizing AlexNet and ResNet18 on ImageNet for 8-bit and 5-bit. AlexNet with 5-bit can achieve even better top1 accuracy than its 32-bit precision.

7 Discussion and challenges

The model compression and acceleration methods of DNNs has achieved significant momentum over the last couple of years. However, the major challenge that still exists with DNNs is finding the fine balance between varying resource availability and system performance for resource-constrained devices. There is no doubt that pruning is one of the main popular technique for removing the redundancy in the network, where the pruning of individual weight connection and neurons makes the resultant model architecture sparse (unstructured pruning).

Due to the limitation of the sparse matrices in terms of their efficient processing by CPUs and GPUs, the resulting model architecture requires specialized hardware and software for processing (Liu et al. 2019; Han et al. 2015) one such work is efficient-inference-engine (Han et al. 2016a). In many existing pruning techniques, pruning percentage is manual, which makes it challenging to prune deep networks. There is a need for more sophisticated techniques where the pruning sensitivity is decided automatically by tuning some hyper-parameter; how much a layer can be pruned (weight connections, neurons, filters). Similarly, structured pruning also need better techniques for deciding the importance of the filters/layers which can be pruned.

To reduce the storage and computational overhead, pruning can be followed by quantization. With quantization the floating-point operations can be replaced with bitwise operations, making their implementation easier on FPGA and ASIC based architectures. However, the use of rounding function for quantizing values makes the network harder to converge. Currently, the use of vector quantization methods is limited to pre-trained models only. The design of new vector quantization methods which can work for quantizing the network during training will add another milestone in the research. There is also a need for new clustering and optimization algorithms for quantizing the weights. Most of the quantization techniques work with the dense layers only. The outcome of these techniques with CNNs and RNNs may also provide some interesting insights. Another issue with quantization techniques is updating the gradient after quantizing weights. The introduction of more efficient methods which can quantize weights, gradient, and activation will open tremendous opportunity for quantized networks as quantizing the gradient and activation is more challenging than quantizing the weights.

Moreover, low-rank factorization of the FC layer weight matrices and convolutional layer filters is also popular among model compression techniques. The accuracy and model performance depends on proper factorization and rank selection. Here, the main challenge is that the decomposition process results in harder implementation and computational intensive. In addition, it is difficult to decompose the models in which a global average pooling layer replaces the dense layer. Knowledge distillation helps in transferring the knowledge learn by the teacher model to a student model, but currently, the use of KD techniques is limited to classification-based applications only. It is challenging to train a student model from the teacher for tasks like object detection, segmentation. Because these networks do not produce logits, which is used to train the student model. The introduction of more KD approaches other than the classification task would be interesting to witness.

Lastly, it is believed that depending upon the need of the application, one can decide what to optimize and what to satisfy. For application, where final model is going to be deployed in devices where computational power and energy is the most critical parameter, one can concentrate more on reducing the parameters/filters from convolutional layers, in which case energy consumption becomes the satisfying parameter and accuracy becomes the optimizing parameter. However, if the final model is going to be deployed on devices with limited storage, reducing the parameters from the dense layer will be more helpful. Moreover, model compression and acceleration techniques are orthogonal to each other. There are several hardware boards available in the market like FPGA, ASIC and other embedded small boards like Raspberry Pi, Arduino, etc. Recently NVIDIA has launched powerful small device Jetson-Nano, which has GPU enabled into it, to make the inference of DNN model faster.

8 Conclusion and future directions

In this paper, various aspects of model compression and acceleration techniques varying from DNNs to traditional machine learning algorithms are discussed. Model compression and acceleration need efforts from different research communities. The actual power of model compression and acceleration will come from a combination of different techniques with combined research outcome in the form of software, efficient algorithm as well as specially designed hardware. The use of deep reinforcement learning to learn the best strategy for compression and acceleration and other optimization techniques will open another scope of the research. In order to achieve acceleration and reduce energy consumption for small devices, the pruning of convolutional layer filters is essential. The introduction of new innovative filter pruning methods, keeping in mind the accelerating and saving energy for small devices will attract significant attention in the future. It was also found that not much work is done in pruning the layers from the very deep network. Therefore, another possible future direction in this area could be the exploration of on-demand model compression and acceleration. The research shall explore on the lines of how much a model can be compressed and accelerated subject to given resource-constraints (storage, computational power, and energy) and user-specified performance goals (accuracy, latency). The development of the generalized model compression and acceleration framework would add another value to it. We hope that in the near future, more research efforts will be seen from hardware, software, and algorithm research community. The results that will make deployment and inferencing of machine learning models (specially DNNs) in embedded devices easier and devices friendly. We also believe that the traditional machine learning algorithms have not been explored much in terms of compression and acceleration, the introduction of new compression and acceleration algorithms in this field will add another milestone.

References

Agarwal A, Xie B, Vovsha I, Rambow O, Passonneau R (2011) Sentiment analysis of twitter data. In: Proceedings of the workshop on language in social media (LSM 2011), pp 30–38

Al-Rfou R, Alain G, Almahairi A, Angermueller C, Bahdanau D, Ballas N, Bastien F, Bayer J, Belikov A, Belopolsky A et al (2016) Theano: a python framework for fast computation of mathematical expressions. ArXiv preprint arXiv:1605.02688

Alvarez JM, Salzmann M (2017) Compression-aware training of deep networks. In: Advances in neural information processing systems, pp 856–867

Anwar S, Hwang K, Sung W (2017) Structured pruning of deep convolutional neural networks. ACM J Emerg Technol Comput Syst (JETC) 13(3):32

Ardakani A, Condo C, Gross WJ (2016) Sparsely-connected neural networks: towards efficient vlsi implementation of deep neural networks. In: Published as a conference paper at ICLR 2017

Ardakani A, Ji Z, Smithson SC, Meyer BH, Gross WJ (2019) Learning recurrent binary/ternary weights. In: International conference on learning representations. https://openreview.net/forum?id=HkNGYjR9FX

Babaeizadeh M, Smaragdis P, Campbell RH (2017) A simple yet effective method to prune dense layers of neural networks. In: Under review as a conference paper at ICLR

Ba J, Caruana R (2014) Do deep nets really need to be deep? In: Advances in neural information processing systems, pp 2654–2662

Balzer W, Takahashi M, Ohta J, Kyuma K (1991) Weight quantization in boltzmann machines. Neural Netw 4(3):405–409

Boni A, Pianegiani F, Petri D (2007) Low-power and low-cost implementation of svms for smart sensors. IEEE Trans Instrum Meas 56(1):39–44

Buciluǎ C, Caruana R, Niculescu-Mizil A (2006) Model compression. In: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 535–541

Cai Z, He X, Sun J, Vasconcelos N (2017) Deep learning with low precision by half-wave gaussian quantization. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5918–5926

Chen S, Zhao Q (2018) Shallowing deep networks: layer-wise pruning based on feature representations. IEEE Trans Pattern Anal Mach Intell 41:3048–3056

Chen G, Choi W, Yu X, Han T, Chandraker M (2017) Learning efficient object detection models with knowledge distillation. In: Advances in neural information processing systems, pp 742–751

Cheng J, Wu J, Leng C, Wang Y, Hu Q (2017) Quantized CNN: a unified approach to accelerate and compress convolutional networks. IEEE Trans Neural Netw Learn Syst 29:4730–4743

Chen T, Goodfellow I, Shlens J (2016) Net2net: accelerating learning via knowledge transfer. In: Published as a conference paper at ICLR

Cheng Z, Soudry D, Mao Z, Lan Z-Z (2015) Training binary multilayer neural networks for image classification using expectation backpropagation. CoRR arXiv:1503.03562

Chen C, Seff A, Kornhauser A, Xiao J (2015a) Deepdriving: learning affordance for direct perception in autonomous driving. In: Proceedings of the IEEE international conference on computer vision. pp 2722–2730

Chen W, Wilson J, Tyree S, Weinberger K, Chen Y (2015b) Compressing neural networks with the hashing trick. In: International conference on machine learning, pp 2285–2294

Choi J, Wang Z, Venkataramani S, Chuang PI-J, Srinivasan V, Gopalakrishnan K (2018) Pact: Parameterized clipping activation for quantized neural networks. ArXiv preprint arXiv:1805.06085

Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp 1724–1734

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12(1):2493–2537

Courbariaux M, Bengio Y, David J-P (2015a) Training deep neural networks with low precision multiplications. In: Accepted as a workshop contribution at ICLR

Courbariaux M, Bengio Y, David J-P (2015b) Binaryconnect: training deep neural networks with binary weights during propagations. In: Advances in neural information processing systems, pp 3123–3131

Crowley EJ, Turner J, Storkey A, O'Boyle M (2018) A closer look at structured pruning for neural network compression. ArXiv preprint arXiv:1810.04622

Demeester T, Deleu J, Godin F, Develder C (2018) Predefined sparseness in recurrent sequence models. In: Proceedings of the 22nd conference on computational natural language learning, pp 324–333

Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L (2009) Imagenet: a large-scale hierarchical image database. In: IEEE conference on computer vision and pattern recognition. IEEE, pp 248–255

Denil M, Shakibi B, Dinh L, De Freitas N et al (2013) Predicting parameters in deep learning. In: Advances in neural information processing systems, pp 2148–2156

Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R (2014) Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in neural information processing systems, pp 1269–1277

Ericsson-Mobility-Report (2018) Ericsson mobility report. https://www.ericsson.com/assets/local/mobility-report/documents/2018/ericsson-mobility-report-november-2018.pdf

Fiesler E, Choudry A, Caulfield HJ (1990) Weight discretization paradigm for optical neural networks. In: Optical interconnections and networks, volume 1281. International Society for Optics and Photonics, pp 164–174

Frankle J, Carbin M (2019) The lottery ticket hypothesis: finding, trainable neural networks. In: Published as a conference paper at ICLR 2019

Gal Y, Ghahramani Z (2016) A theoretically grounded application of dropout in recurrent neural networks. In: Advances in neural information processing systems, pp 1019–1027

Garofolo JS, Lamel LF, Fisher WM, Fiscus JG, Pallett DS (1993) DARPA TIMIT acoustic–phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1. In: NASA STI/Recon technical report N 93

Girshick R, Donahue J, Darrell T, Malik J (2014) Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 580–587

Gong Y, Liu L, Yang M, Bourdev L (2015) Compressing deep convolutional networks using vector quantization. In: Under review as a conference paper at ICLR

Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680

Gordon A, Eban E, Nachum O, Chen B, Wu H, Yang T-J, Choi E (2018) Morphnet: fast and simple resource-constrained structure learning of deep networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1586–1595

Graves A, Schmidhuber J (2005) Framewise phoneme classification with bidirectional lstm and other neural network architectures. Neural Netw 18(5–6):602–610

Graves A, Mohamed A-R, Hinton G (2013) Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 6645–6649

Guo Y, Yao A, Chen Y (2016) Dynamic network surgery for efficient dnns. In: Advances in neural information processing systems, pp 1379–1387

Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P (2015) Deep learning with limited numerical precision. In: International conference on machine learning, pp 1737–1746

Gupta C, Suggala AS, Goyal A, Simhadri HV, Paranjape B, Kumar A, Goyal S, Udupa R, Varma M, Jain P (2017) Protonn: compressed and accurate KNN for resource-scarce devices. In: International conference on machine learning, pp 1331–1340

Han S, Liu X, Mao H, Pu J, Pedram A, Horowitz MA, Dally WJ (2016a) EIE: efficient inference engine on compressed deep neural network. In: 2016 ACM/IEEE 43rd annual international symposium on computer architecture (ISCA). IEEE, pp 243–254

Han S, Mao H, Dally WJ (2016b) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In: Published as a conference paper at ICLR

Han S, Pool J, Tran J, Dally W (2015) Learning both weights and connections for efficient neural network. In: Advances in neural information processing systems, pp 1135–1143

Hassibi B, Stork DG (1993) Second order derivatives for network pruning: optimal brain surgeon. In: Advances in neural information processing systems, pp 164–171

He Y, Lin J, Liu Z, Wang H, Li L-J, Han S (2018) Amc: automl for model compression and acceleration on mobile devices. In: Proceedings of the European conference on computer vision (ECCV), pp 784–800

He Q, Wen H, Zhou S, Wu Y, Yao C, Zhou X, Zou Y (2016b) Effective quantization methods for recurrent neural networks. ArXiv preprint arXiv:1611.10176

He K, Zhang X, Ren S, Sun J (2016a) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE international conference on computer vision, pp 1389–1397

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

Horowitz M (2014) 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC). IEEE, pp 10–14

Hou L, Kwok JT (2018) Loss-aware weight quantization of deep networks. In: Published as a conference paper at ICLR 2018. https://openreview.net/forum?id=BkrSv0lA-

Hou L, Yao Q, Kwok JT (2017) Loss-aware binarization of deep networks. In: Published as a conference paper at ICLR

Howard AG, Zhu AG, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. ArXiv preprint arXiv:1704.04861

Huang G, Liu Z, Van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4700–4708

Huang G, Liu S, Van der Maaten L, Weinberger KQ (2018) Condensenet: an efficient densenet using learned group convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2752–2761

Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2017) Quantized neural networks: Training neural networks with low precision weights and activations. J Mach Learn Res 18(1):6869–6898

Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks. In: Advances in neural information processing systems, pp 4107–4115

Hwang K, Sung W (2014) Fixed-point feedforward deep neural network design using weights +1, 0, and −1. In: 2014 IEEE workshop on signal processing systems (SiPS). IEEE, pp 1–6

Iandola FN, Han S, Moskewicz MW, Ashraf K, Dally WJ, Keutzer K (2017) Squeezenet: alexnet-level accuracy with 50× fewer parameters and < 0.5 MB model size. In: International conference on learning representations

Ioannou Y, Robertson D, Shotton J, Cipolla R, Criminisi A (2016) Training cnns with low-rank filters for efficient image classification. In: Published as a conference paper at ICLR

Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, pp 448–456

Jaderberg M, Vedaldi A, Zisserman A (2014) Speeding up convolutional neural networks with low rank expansions. In: Proceedings of the British machine vision conference. BMVA Press

Joly A, Schnitzler F, Geurts P, Wehenkel L (2012) L1-based compression of random forest models. In: 20th European symposium on artificial neural networks

Jose C, Goyal P, Aggrwal P, Varma M (2013) Local deep kernel learning for efficient non-linear svm prediction. In: International conference on machine learning, pp 486–494

Juefei-Xu F, Boddeti VN, Savvides M (2017) Local binary convolutional neural networks. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR). IEEE, pp 4284–4293

Kim Y-D, Park E, Yoo S, Choi T, Yang L, Shin D (2016) Compression of deep convolutional neural networks for fast and low power mobile applications. In: Published as a conference paper at ICLR

Kim J, Hwang K, Sung W (2014) X1000 real-time phoneme recognition vlsi using feed-forward deep neural networks. In: 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 7510–7514

Kim J, Park S, Kwak N (2018) Paraphrasing complex network: network compression via factor transfer. In: Advances in neural information processing systems, pp 2760–2769

Kim M, Smaragdis P (2016) Bitwise neural networks. In: International conference on machine learning (ICML) workshop on resource-efficient machine learning

Kim M, Smaragdis P (2018) Efficient source separation using bitwise neural networks. In: Audio source separation. Springer, pp 187–206

Krizhevsky A (2014) One weird trick for parallelizing convolutional neural networks. ArXiv preprint arXiv:1404.5997

Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. In: Technical report. Citeseer

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

Kumar A, Goyal S, Varma M (2017) Resource-efficient machine learning in 2 kb ram for the internet of things. In: International conference on machine learning, pp 1935–1944

Kusupati A, Singh M, Bhatia K, Kumar A, Jain P, Varma M (2018) Fastgrnn: a fast, accurate, stable and tiny kilobyte sized gated recurrent neural network. In: Advances in neural information processing systems, pp 9031–9042

Lan X, Zhu X, Gong S (2018) Knowledge distillation by on-the-fly native ensemble. In: Proceedings of the 32nd international conference on neural information processing systems. Curran Associates Inc., pp 7528–7538

Lebedev V, Ganin Y, Rakhuba M, Oseledets I, Lempitsky V (2015) Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In: Published as a conference paper at ICLR

LeCun Y (1998) The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/

LeCun Y, Bottou L, Bengio Y, Haffner P et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324

LeCun Y, Denker JS, Solla SA (1990) Optimal brain damage. In: Advances in neural information processing systems, pp 598–605

Lee JD, Sun Y, Saunders MA (2014) Proximal newton-type methods for minimizing composite functions. SIAM J Optim 24(3):1420–1443

Le Q, Sarlós T, Smola A (2013) Fastfood-approximating kernel expansions in loglinear time. In: Proceedings of the international conference on machine learning, volume 85

Li X-B, Sweigart J, Teng J, Donohue J, Thombs L (2001) A dynamic programming based pruning method for decision trees. INFORMS J Comput 13(4):332–344

Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. In: Published as a conference paper at ICLR

Li F, Liu B (2016) Ternary weight networks. In: 30th conference on neural information processing systems (NIPS). Barcelona

Lin C-Y, Wang T-C, Chen K-C, Lee B-Y, Kuo J-J (2019) Distributed deep neural network deployment for smart devices from the edge to the cloud. In: Proceedings of the ACM MobiHoc workshop on pervasive systems in the IoT era. pp 43–48

Lin J-H, Xing T, Zhao R, Zhang Z, Srivastava MB, Tu Z, Gupta RK (2017a) Binarized convolutional neural networks with separable filters for efficient hardware acceleration. In: CVPR workshops, pp 344–352

Lin T-Y, Maire M, Belongie S, Hays J, Perona P, Ramanan D, Dollár P, Zitnick CL (2014) Microsoft coco: Common objects in context. In: European conference on computer vision. Springer, pp 740–755

Lin M, Chen Q, Yan S (2013) Network in network. ArXiv preprint arXiv:1312.4400

Lin Z, Courbariaux M, Memisevic R, Bengio Y (2016b) Neural networks with few multiplications. In: Published as a conference paper at ICLR

Lin J, Rao Y, Lu J, Zhou J (2017b) Runtime neural pruning. In: Advances in neural information processing systems, pp 2181–2191

Lin D, Talathi S, Annapureddy S (2016a) Fixed point quantization of deep convolutional networks. In: International conference on machine learning, pp 2849–2858

Li C, Shi CJR (2018) Constrained optimization based low-rank approximation of deep neural networks. In: European conference on computer vision. Springer, pp 746–761

Littman ML (2015) Reinforcement learning improves behaviour from evaluative feedback. Nature 521(7553):445

Liu S, Lin Y, Zhou Z, Nan K, Liu H, Du J (2018) On-demand deep model compression for mobile devices: a usage-driven model selection framework. In: Proceedings of the 16th annual international conference on mobile systems, applications, and services. ACM, pp 389–400

Liu Z, Li J, Shen Z, Huang G, Yan S, Zhang C (2017) Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE international conference on computer vision, pp 2736–2744

Liu Z, Sun M, Zhou T, Huang G, Darrell T (2019) Rethinking the value of network pruning. In: Published as a conference paper at ICLR

Liu B, Wang M, Foroosh H, Tappen M, Pensky M (2015) Sparse convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 806–814

Li E, Zeng L, Zhou Z, Chen X (2019) Edge AI: on-demand accelerating deep neural network inference via edge computing. IEEE Trans Wirel Commun 19:447–457

Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–137

Lobacheva E, Chirkova N, Vetrov D (2017) Bayesian sparsification of recurrent neural networks. In: Published in workshop on learning to generate natural language. ICML

Lobacheva E, Chirkova N, Vetrov D (2018) Bayesian sparsification of gated recurrent neural networks. In: Published in workshop on compact deep neural networks with industrial applications. NeurIPS

Luo J-H, Zhang H, Zhou H-Y, Xie C-W, Wu J, Lin W (2018) Thinet: pruning cnn filters for a thinner net. IEEE Trans Pattern Anal Mach Intell 41:2525–2538

Lu Z, Sindhwani V, Sainath TN (2016) Learning compact recurrent neural networks. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 5960–5964

Marcus M, Santorini B, Marcinkiewicz MA (1993) Building a large annotated corpus of english: the penn treebank. Comput Linguist 19(2):313–330

Ma N, Zhang X, Zheng H-T, Sun J (2018) Shufflenet v2: practical guidelines for efficient cnn architecture design. In: Proceedings of the European conference on computer vision (ECCV), pp 116–131

Mehta S, Rastegari M, Caspi A, Shapiro L, Hajishirzi H (2018) Espnet: efficient spatial pyramid of dilated convolutions for semantic segmentation. In: Proceedings of the European conference on computer vision (ECCV), pp 552–568

Mehta S, Rastegari M, Shapiro L, Hajishirzi H (2019) Espnetv2: a light-weight, power efficient, and general purpose convolutional neural network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 9190–9200

Mishra A, Marr D (2017) Apprentice: using knowledge distillation techniques to improve low-precision network accuracy. ArXiv preprint arXiv:1711.05852

Molchanov D, Ashukha A, Vetrov D (2017a) Variational dropout sparsifies deep neural networks. In: Proceedings of the 34th international conference on machine learning volume 70. JMLR.org, pp 2498–2507

Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2017b) Pruning convolutional neural networks for efficient inference. In: Published as a conference paper at ICLR

Moshtaghi M, Rajasegarar S, Leckie C, Karunasekera S (2011) An efficient hyperellipsoidal clustering algorithm for resource-constrained environments. Pattern Recognit 44(9):2197–2209

Nakajima S, Tomioka R, Sugiyama M, Babacan SD (2012) Perfect dimensionality recovery by variational Bayesian PCA. In: Advances in neural information processing systems, pp 971–979

Nan F, Wang J, Saligrama V (2016) Pruning random forests for prediction on a budget. In: Advances in neural information processing systems, pp 2334–2342

Narang S, Elsen E, Diamos G, Sengupta S (2017) Exploring sparsity in recurrent neural networks. In: Published as a conference paper at ICLR

Narang S, Undersander E, Diamos GF (2018) Block-sparse recurrent neural networks. CoRR arXiv:1711.02782

Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng AY (2011) Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. Granada, p 5

Novikov A, Podoprikhin D, Osokin A, Vetrov DP (2015) Tensorizing neural networks. In: Advances in neural information processing systems, pp 442–450

Oguntola I, Olubeko S, Sweeney C (2018) Slimnets: an exploration of deep model compression and acceleration. In: 2018 IEEE high performance extreme computing conference (HPEC). IEEE, pp 1–6

Ott J, Lin Z, Zhang Y, Liu S-C, Bengio Y (2016) Recurrent neural networks with limited numerical precision. ArXiv preprint arXiv:1608.06902

Panayotov V, Chen G, Povey D, Khudanpur S (2015) Librispeech: an asr corpus based on public domain audio books. In: IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 5206–5210

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) PyTorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems, pp 8024–8035

Peddinti V, Povey D, Khudanpur S (2015) A time delay neural network architecture for efficient modeling of long temporal contexts. In: Sixteenth annual conference of the international speech communication association

Polino A, Pascanu R, Alistarh D (2018) Model compression via distillation and quantization. In: Published as a conference paper at ICLR 2018

Povey D, Cheng G, Wang Y, Li K, Xu H, Yarmohamadi M, Khudanpur S (2018) Semi-orthogonal low-rank matrix factorization for deep neural networks. In: Proceedings of the 19th annual conference of the international speech communication association (INTERSPEECH). Hyderabad

Prabhavalkar R, Alsharif O, Bruguier A, McGraw L (2016) On the compression of recurrent neural networks with an application to lvcsr acoustic modeling for embedded speech recognition. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 5970–5974

Puterman ML (2014) Markov decision processes: discrete stochastic dynamic programming. Wiley, Hoboken

Rakotomamonjy A, Flamary R, Gasso G (2015) Dc proximal newton for nonconvex optimization problems. IEEE Trans Neural Netw Learn Syst 27(3):636–647

Rastegari M, Ordonez V, Redmon J, Farhadi A (2016) Xnor-net: imagenet classification using binary convolutional neural networks. In: European conference on computer vision. Springer, pp 525–542

Ren S, He K, Girshick R, Sun J (2015) Faster r-cnn: towards real-time object detection with region proposal networks. In: Advances in neural information processing systems, pp 91–99

Rigamonti R, Sironi A, Lepetit V, Fua P (2013) Learning separable filters. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2754–2761

Romero A, Ballas N, Kahou SE, Chassang A, Gatta C, Bengio Y (2015) Fitnets: hints for thin deep nets. In: Published as a conference paper at ICLR

Sainath TN, Kingsbury B, Sindhwani V, Arisoy E, Ramabhadran B (2013) Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: 2013 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 6655–6659

Sandler M, Howard A, Zhu M, Zhmoginov A, Chen L-C (2018) Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 4510–4520

Shen J, Vesdapunt N, Boddeti VN, Kitani KM (2016) In teacher we trust: learning compressed models for pedestrian detection. ArXiv preprint arXiv:1612.00478

Sherali HD, Hobeika AG, Jeenanunta C (2009) An optimal constrained pruning strategy for decision trees. INFORMS J Comput 21(1):49–61

Shin S, Hwang K, Sung W (2016) Fixed-point performance analysis of recurrent neural networks. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 976–980

Shi B, Sun M, Kao C-C, Rozgic V, Matsoukas S, Wang C (2018) Compression of acoustic event detection models with low-rank matrix factorization and quantization training. In: 32nd conference on neural information processing systems. Montreal

Shotton J, Sharp T, Kohli P, Nowozin S, Winn J, Criminisi A (2013) Decision jungles: compact and rich models for classification. In: Advances in neural information processing systems, pp 234–242

Simonyan K, Zisserman A (2015) Very deep convolutional networks for large-scale image recognition. In: Published as a conference paper at ICLR

Soudry D, Hubara I, Meir R (2014) Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in neural information processing systems, pp 963–971

Srinivas S, Babu RV (2015) Data-free parameter pruning for deep neural networks. ArXiv preprint arXiv:1507.06149

Srinivas S, Fleuret F (2018) Knowledge transfer with jacobian matching. ArXiv preprint arXiv:1803.00443

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958

Sung W, Shin S, Hwang K (2015) Resiliency of deep neural networks under quantization. ArXiv preprint arXiv:1511.06488

Sutskever I, Martens J, Hinton GE (2011) Generating text with recurrent neural networks. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp 1017–1024

Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: Advances in neural information processing systems, pp 3104–3112

Suzuki K, Horiba I, Sugie N (2001) A simple neural network pruning algorithm with application to filter synthesis. Neural Process Lett 13(1):43–53

Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1–9

Tai C, Xiao T, Zhang Y, Wang X et al (2016) Convolutional neural networks with low-rank regularization. In: Published as a conference paper at ICLR

Theis L, Korshunova I, Tejani A, Huszar F (2018) Faster gaze prediction with dense networks and fisher pruning. ArXiv preprint arXiv:1801.05787

Tibshirani R (1996) Regression shrinkage and selection via the lasso. J R Stat Soc Ser B (Methodol) 58(1):267–288

Tucker LR (1966) Some mathematical notes on three-mode factor analysis. Psychometrika 31(3):279–311

Verhelst M, Moons B (2017) Embedded deep neural network processing: algorithmic and processor techniques bring deep learning to iot and edge devices. IEEE Solid State Circuits Mag 9(4):55–65

Vu TH, Dung L, Wang J-C (2016) Transportation mode detection on mobile devices using recurrent nets. In: Proceedings of the 24th ACM international conference on multimedia. ACM, pp 392–396

Wu B, Iandola FN, Jin PH, Keutzer K (2017) Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: CVPR workshops, pp 446–454

Wu X, Wu Y, Zhao Y (2016) Binarized neural networks on the imagenet classification task. ArXiv preprint arXiv:1604.03058

Xu K, Ba J, Kiros R, Cho K, Courville A, Salakhudinov R, Zemel R, Bengio Y (2015) Show, attend and tell: Neural image caption generation with visual attention. In: International conference on machine learning, pp 2048–2057

Xu C, Yao J, Lin Z, Ou W, Cao Y, Wang Z, Zha H (2018) Alternating multi-bit quantization for recurrent neural networks. In: Published as a conference paper at ICLR

Yang T-J, Chen Y-H, Sze V (2017) Designing energy-efficient convolutional neural networks using energy-aware pruning. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 5687–5695

Yang Z, Moczulski M, Denil M, de Freitas N, Smola A, Song L, Wang Z (2015) Deep fried convnets. In: Proceedings of the IEEE international conference on computer vision, pp 1476–1483

Yuan Z, Lu Y, Wang Z, Xue Y (2014) Droid-sec: deep learning in android malware detection. In: ACM SIGCOMM computer communication review, volume 44. ACM, pp 371–372

Yu X, Liu T, Wang X, Tao D (2017) On compressing deep models by low rank and sparse decomposition. In: 2017 IEEE conference on computer vision and pattern recognition (CVPR). IEEE, pp 67–76

Zagoruyko S, Komodakis N (2017) Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. In: Published as a conference paper at ICLR

Zhang X, Zou J, He K, Sun J (2016) Accelerating very deep convolutional networks for classification and detection. IEEE Trans Pattern Anal Mach Intell 38(10):1943–1955

Zhang J, Wang X, Li D, Wang Y (2018a) Dynamically hierarchy revolution: dirnet for compressing recurrent neural network on mobile devices. In: Proceedings of the 27th international joint conference on artificial intelligence. AAAI Press, pp 3089–3096

Zhang X, Zhou X, Lin M, Sun J (2018b) Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 6848–6856

Zhang X, Zou J, Ming X, He K, Sun J (2015) Efficient and accurate approximations of nonlinear convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 1984–1992

Zhao R, Song W, Zhang W, Xing T, Lin J-H, Srivastava M, Gupta R, Zhang Z (2017) Accelerating binarized convolutional neural networks with software-programmable fpgas. In: Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays. ACM, pp 15–24

Zhou S-C, Wang Y-Z, Wen H, He Q-Y, Zou Y-H (2017b) Balanced quantization: an effective and efficient approach to quantized neural networks. J Comput Sci Technol 32(4):667–682

Zhou S, Wu Y, Ni Z, Zhou X, Wen H, Zou Y (2016) Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. ArXiv preprint arXiv:1606.06160

Zhou A, Yao A, Guo Y, Xu L, Chen Y (2017a) Incremental network quantization: towards lossless cnns with low-precision weights. ArXiv preprint arXiv:1702.03044

Zhu M, Gupta S (2017) To prune, or not to prune: exploring the efficacy of pruning for model compression. ArXiv preprint arXiv:1710.01878

Zhu C, Han S, Mao H, Dally WJ (2017) Trained ternary quantization. In: Published as a conference paper at ICLR

Zhu F, Pool J, Andersch M, Appleyard J, Xie F (2018) Sparse persistent RNNs: Squeezing large recurrent networks on-chip. In: International conference on learning representations. https://openreview.net/forum?id=HkxF5RgC-