On Pruned, Quantized and Compact CNN Architectures for Vision Applications

An Empirical Study

Pius Kwao Gadosey
Computer Science and Technology
Beijing University of Technology
Beijing, China
kwaogad@emails.bjut.edu.cn

Yujian Li*
School of Artificial Intelligence
Guilin University of Electronic
Technology, Guilin, Guangxi
liyujian@guet.edu.cn

Peter T Yamak
Computer Science and Technology
Beijing University of Technology
Beijing, China
peteryamak@emails.bjut.edu.cn

ABSTRACT

With the influx of several kinds of mobile electronic devices alongside the increasing popularity of deep learning networks in performing computer vision tasks, it is natural that demands for delivering them on smaller devices will increase. The authors of this paper review and experiment with compact models (MobileNet V1 and V2, ShuffleNet V1 and V2, FD-MobileNet)) and selected methods of pruning and quantization of popular Convolutional Neural Network (CNN) through transfer learning tasks. They further propose a hybrid technique of per layer pruning and quantization called Pruned Sparse Binary-Weight Network (PSBWN). The performance of these four techniques are evaluated on image classification tasks on the Caltech–UCSD Birds 200, Oxford Flowers 102 and CALTECH256 which are all publicly available benchmark datasets with focus on the trade-offs among the number of Floating Point Operations (FLOPS), model sizes, training and inference times against accuracy using the same computation resources.

CCS CONCEPTS

• Computing Methodologies • Artificial Intelligence

KEYWORDS

Neural networks, Pruning, Quantization, Compact networks

1 Introduction

In recent times, there has been an increased popularity of deep learning solutions coupled with the need to provide them on mobile handheld devices, embedded systems, or any low powered computers. In just a few years, many CNNs have been developed and used effectively in computer vision tasks such as image classification and object detection at human accuracy levels, thanks to the ImageNet challenge. From classic CNNs such as [1-3] to newer networks architectures like ResNeT [4], Inception [5], Xception [6] and so on. These CNNs usually require larger computing power and storage capacity for training and inference. This makes it more difficult for deployment on mobile handheld devices and embedded systems that have minimal resources. For this reason, several techniques have been proposed by the research community for model size reduction, faster training or inference speed, and fewer computation requirements while preserving or minimizing accuracy losses [7-8]. A compression-aware training method that introduces a regularizer to encourage each layer's parameter matrix to have a low rank during the training process [9]. Han et al. proposed a procedure of combining network pruning, quantization, and Huffman coding without hurting the overall accuracy. Their method was efficient for model size reduction but not necessarily for reducing computation time. HashedNets [10] are networks that take advantage of the redundancy present in neural networks to achieve significant reductions in model size. SqueezeNet [11] also has 50 times (50×) fewer parameters than AlexNet [2] and even a smaller model size (510× smaller) but can achieve similar accuracy on ImageNet. Similarly, several quantization techniques have also been introduced with the goal of reducing model architectures from 32-bit floating-point representations to lower (8 bit, 2 bit) representations. Vector quantization is a lossy compression technique applied to neurons in fully-connected layers [12]. In their work, [13, 14] proposed training CNNs with ternary and binary weights, respectively. There is also a technique that uses low-bit approximation algorithms to reduce the computational complexity required for training and accelerates inference [15]. With all these methods, there is usually the problem of significant accuracy losses and require specialized hardware for inference. Trained ternary quantization [16] improves quantization accuracy by using two full-precision scaling coefficients and quantizing the weights to lower precision weights. Compact Networks are modern methods that make use of resources available and size to train

AIIPCC '19, December 19–21, 2019, Sanya, China

ACM ISBN 978-1-4503-7633-4/19/12…$15.00
https://doi.org/10.1145/3371425.3371481

smaller network models from scratch. MobileNets [17, 18], ShuffleNets [19] and their variants mostly make use of separable convolution layers [20] [6] [21] during the model design and training.

1.1 Contribution

This paper makes the following contributions

- A brief review of existing model pruning, quantization methods, and compact deep learning network architectures
- A proposed method of per-layer filter pruning with a ranking method based on a Taylor Expansion Criterion (TEC), followed by trained ternary quantization (TTQ) to reduce the pruned model's full-precision (32-bit) weights to ternary (2-bit) values, thus, obtaining a pruned sparse binary-weight network (PSBWN)
- PSBWN achieves the smallest model size when stored in 2-bits and its high level of sparsity reduces computational requirements significantly with improvement in accuracy over baseline models (models obtained after transfer learning).

Section 2 of this paper gives some background information and an elaborated review of related work. Section 3 describes our proposed methodology. Section 4 describes our experiments and results, and Section 5 provides a brief discussion. Finally, Section 6 presents our conclusions

2 Related Work

In this section, the background and significant previous works that motivated our work are described briefly.

2.1 Pruning

Pruning has become a prevalent technique for reducing large pretrained deep neural networks by removing irrelevant weights and fine-tuning the models to preserve accuracy. Pruning is necessary since deep neural networks are over parameterized [22], with many parameters considered redundant in contributing to the performance of these networks.

Network Slimming proposed a learning scheme for pruning CNNs by enforcing channel-level sparsity [23]. They achieve this by automatically identifying insignificant channels in the network during training. The main idea in network slimming is to introduce a scaling factor for each channel and multiplying it to the output of that channel. The network weights and these scaling factors are trained together, with sparsity regularization imposed on the scaling factors. The channels are finally pruned with small factors, and the network is fine-tuned to retain its accuracy.

Luo et al. [24] also proposed an efficient filter level pruning technique to compress CNN models and increase both training and inference speeds at the same time known as ThiNet. First by feature selection, where they utilize a subset of channels in layer (i + 1)'s input to approximate the output in layer i + 1, so other channels can be safely removed from the input of layer delivering + 1 while still pruning corresponding filters in layer, i. This is followed by pruning away weak channels in layer (i + 1)'s input and their corresponding filters in layer i leading to a much smaller model.

Molchanov et al. [25] adapted large networks pre-trained on a related but different dataset with initialized parameters. They treat pruning as a combinatorial optimization problem. Using Taylor Expansion, change in loss function is approximated directly from removing particular parameters. This helps to find a set of parameters that minimize the change in the loss function.

2.2 Quantization

Research has shown that network weights and parameters could be represented as lower precision (2-bit, 8-bit, 16-bit) weights, instead of the usual 32-bits without any effect on their performance. This saves massively on computation and memory storage. The technique for representing full precision weights as lower precision weights is generally known as quantization.

DoReFa-Net [15] quantizes weights, activations, and gradients of neural networks using different widths of bits. They were able to accelerate both training and inference of neural networks using specifically designed low bit multiplication algorithms. They also introduced a simple technique to quantize 32-bit weights to binary values

Ternary Weight Networks (TWNs) [26] introduced zero as a third quantized value, and this helped to reduce the accuracy loss of binary networks.

In Trained Ternary Quantization(TTQ) [16], full precision weights (usually 32bit floats) are initially normalized to the range [-1, +1]. This is done by dividing each individual weight by the maximum weight. Through a method of thresholding where a thresholding factor t, a hyperparameter that is maintained across all layers is then used to quantize the intermediate, full precision weights to -1,0, 1. Trained quantization is finally performed through the backpropagation of two gradients. The separate gradients are backpropagated to the full precision weights and the scaling co-efficient. The ternary assignments are learned by backpropagation to full resolution weights while learning the ternary values are enabled by backpropagation to the scaling coefficients. During inference, only the ternary weights are used while the full-resolution weights are discarded.

2.3 Compact Network Architectures

While pruning takes advantage of the redundancy of neurons in largely pre-trained CNNS and fine-tune them in transfer learning tasks, recently proposed networks commonly referred to as compact networks. MobileNets [17], [18], ShuffleNets [19], FD-MobileNet [27], allow developers to exploit resource latency and size for their applications. These CNN architectures primarily make use of depthwise separable convolutions used in [5], [6], Sifre, and [28].

In depthwise convolutions, a standard convolution is factorized into a depthwise convolution (usually 3×3) and a pointwise convolution (1×1 convolution). In a standard convolution, the output channel is always one irrespective of the number of input channels available. In depthwise separable

convolutions, features are only learned from the input channels, so the output layer has the same number of channels as the input. This is called a depthwise convolution. This is then followed by a pointwise (1×1) convolution layer, which computes the weighted sum of all output channels into a single output. Depthwise separable convolutions reduce computational complexity significantly.

The MobileNet architecture is built on depthwise separable convolutions but for the first layer, which is a standard convolution. In each mobileNet unit, is a batch normalization layer and a Rectified Linear Unit (ReLU) activation layer then follows each convolution layer. After the original mobileNet (V1) came, the mobileNetv2 with a slight modification, which included inverted residual blocks or bottlenecks initially used in [4]. The mobileNet V2 also introduced another 1x1 convolution layer at the input of each unit termed the expansion layer and the last 1×1 convolution layer (projection) without a ReLU activation ShuffleNet V1 was introduced to have better performance compared to mobileNet in terms of computation requirements. A shuffleNet unit generally makes use of a bottleneck structure consisting of depthwise separable convolutions but the use of grouped convolutions that were first introduced in [2] and channel shuffle layers. The channel shuffle layer shuffles the order of the channels among groups in the grouped. This was because the reduction in FLOPS does not necessarily mean improved speed, which is a direct metric for small devices with limited resource budgets. ShuffleNets are designed following four guidelines which include; Equal channel width minimizes memory access cost (MAC), Excessive group convolution increases MAC, Network fragmentation reduces the degree of parallelism, and elementwise operations are non-negligible.

FD-MobileNet [27] is also an improvement of the mobileNet architecture, which is designed to satisfy devices with very low computation budgets (10-140 FLOPS). The major technique used in FD-MobileNet is the fast downsampling method implemented initially in [3]. 32× downsampling is done within only 12 layers, exactly half of mobileNet v1. According to the paper, FD-MobileNet outperforms mobileNet and obtains comparable results with shuffleNet in terms of accuracy. The downsampling method employed in the depthwise separable convolutions further reduce computation complexity in FD-MobileNet.

3 Methodology

An elaborate description of the proposed training method is given in this section.

3.1 Pruned Sparse Binary-weight Network (PSBWN)

PSBWN is a CNN compression method that involves first pruning using the Taylor Expansion Criterion and then performing TTQ (Algorithm 1). The models are trained via transfer learning, with a large network pre-trained on the ImageNet dataset and then adapted to smaller tasks. Between 65–70% of the lowest-ranked convolutional filters are removed, with about 13–15% removed per pruning iteration. After each iteration, 20–30 fine-tuning iterations are performed, based on the lowest-ranking filters, to compensate for the accuracy lost due to pruning, which is expected to degrade the network's performance. After pruning, we then reduce the resulting full-precision weights to ternary values and fine-tune them using TTQ.

3.1.1 Pruning. For the VGG models, the pre-trained models are fine-tuned using batch normalization layers after each convolution layer. The parameters of the batch normalization layer underlying each pruned convolution layer are also pruned by replacing their input features with the corresponding convolution layer's output features. The goal of batch normalization is to achieve a stable activation value distribution throughout training, and it is also known to accelerate training [29]. The filters in all the convolution layers, as well as the first fully connected layer, are pruned. For ResNet-18, all the convolution layers, including those in the residual blocks' shortcut layers, are also pruned by pruning the shortcut convolution layer followed by the second residual block layer based on the indexes of the pruned filters in the shortcut convolution layer.

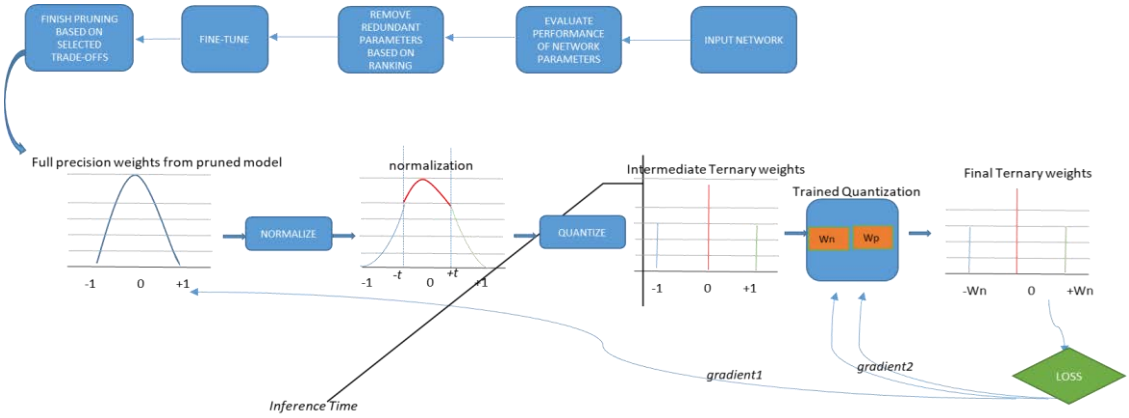

Figure 1: Overview of PSBWN: Applying TTQ to the Full-precision Weights Obtained after Pruning the Model.

3.1.2 Quantization. Given satisfactory pruning results, we initialize the TTQ procedure with the full-precision weights obtained from the pruned network. TTQ first normalizes the full-precision weights to the range [-1, +1] by dividing each weight by the maximum one, then quantizes them to {-1, 0, +1} by thresholding. Finally, trained quantization is performed by back-propagating two gradients to the full-resolution weights and scaling coefficients. For inference, the full-resolution weights are discarded, and only the final ternary weights are used (Figure 1).

Algorithm 1: Pruning + Quantization

Input: baseline model, $X_{\text{train}}, X_{\text{val}}$

1. Train the model by transfer learning with the given data.
2. Assess and rank all neurons via Taylor expansion.
3. Remove the least-important neurons.
4. Fine-tune the model to recover accuracy.
5. Save the pruned model.
6. Initialize the model using the full-precision weights from the pruned model.
7. Normalize by dividing each weight by the maximum weight to obtain the range $[-1, +1]$.
8. Determine a suitable thresholding parameter t to quantize the weights to$\{-1,0, +1\}$.
9. Backpropagate the gradients to W_l^p, W_l^n and the scaling coefficients.

Output: pruned ternary weights, scaling factors

3.1.3 Advantages of PSBWN. From Table 1 and Figure 2, at least 72% of the weights in each quantized convolution layer are zero, making them very sparse (with several zero weights). Avoiding the calculations related to these zero weights, therefore significantly reducing the computational cost and improving energy efficiency. Pruning reduced the model sizes by 2–4×, and TTQ reduced this by a further 16× (due to using 2-bit rather than the original 32-bit weights), giving an overall size reduction of about 32× smaller with better accuracy compared to the baseline models.

4 Setup and Experiments

Pytorch [30], a python based computing package that makes efficient use of GPUs and provides maximum flexibility and speed is used for experiments. The experiments were performed on a workstation enabled with an NVidia Tesla K40c GPU (12GB memory per board) and Intel R Xeon (R) CPU E5-2603 V4 @ 1.70 GHz with 12CPUs. CuDNN 7.0 library.

The Caltech—UCSD Birds 200 (CUB-200) [31] dataset is a challenging image dataset of (mostly North American) birds containing 11,788 images divided into 200 species.

The Caltech-256 dataset [32] contains a total of 30,607 images divided into 256 diverse classes (e.g., faces, beavers, and anchors) with substantial intra-class appearance, shape, and location variability.

The Oxford Flowers 102 dataset [33] consists of 8,189 images of flowers that usually grow in the United Kingdom and is divided into 102 classes, with each class represented unequally distributed between 40 and 258 images.

4.1 Training

For all network architectures (VGG16, VGG19, and ResNet18), pruning during the transfer learning task was done with 200 epochs unless otherwise stated for specific tasks with a batch size of 32 after which 30 epochs of fine-tuning was done to recover the accuracy loss. SGD with a batch size of 32, momentum of 0.9, and weight decay of 1×10^{-4} was chosen. For trained Ternary quantization, a thresholding hyperparameter of t = 0.15 was used and the Adam optimizer to update both the full precision kernels and scaling factors for 200 epochs. MobileNets (V1 and V2) were both trained with a width multiplier of 1.0, ShuffleNets with the number of group convolutions set to two.

4.2 Classification on CUB-200

The proposed method was first tested with the VGG-16, VGG-19 and ResNet18 networks, fine-tuned on the CUB-200 dataset. On VGG-19 for instance, transfer learning began at a learning rate of 1×10^{-4}, allowing for a maximum of 300 epochs. However, training stopped early (due to convergence) at the 88$^{\text{th}}$ epoch, having reduced the learning rate to 1×10^{-5} at the 73$^{\text{rd}}$ epoch, 1×10^{-6} at the 78$^{\text{th}}$ epoch, and finally 1×10^{-7} at the 83$^{\text{rd}}$ epoch. The final Top-1 and Top-5 accuracies of the model were 60.3% and 87.2%, respectively, and it occupied 541MB of disk space. After pruning over 65% of the full model's filters, these improved significantly, to 73.2% and 92.1% for the Top-1 and Top-5 accuracies, respectively.15 epochs of fine-tuning for each pruning iteration (of which 6 were required to prune VGG-19), using SGD with a batch size of 64, momentum of 0.9, and weight decay of 1×10^{-4} was applied. Quantization on all convolution layers except the first, as well as on the three fully-connected layers was implemented on the pruned model, resulting in a PSBWN model, using a learning rate of 1×10^{-4} and a thresholding hyperparameter, $t = 0.15$. The results (Table 2) show slightly lower accuracies compared with quantization and pruning individually, but better than the original baseline model. Similarly, for VGG-16, there was an increase in the top-1 and a slight drop in the top-5 accuracies, respectively, with similar training parameters as for VGG-19 but only five pruning iterations. Table 1 summarizes the number of filters pruned per iteration. PSBWN also improved by 6% and 1.4% in top-1 and top-5 accuracies respectively with ResNet18. (Table 2).

4.3 Classification on Caltech 256

VGG16, after the transfer-learning task, achieved 63.7 and 83.9 of top-1 accuracy and top-5 accuracy respectively occupying a disk space of 541MB. There was a reduction of model size to 238MB after pruning. However, there was an increase in accuracy in this case for top-1 and top-5 accuracies. After performing TTQ on VGG16, the accuracy drop was significantly lower than the pruned. Testing with PSBWN, there was an increase of 1.7% and 1.0% in the Top-1 and Top-5 accuracies, respectively with the model size reduction up to 14.8MB. (Table 3).

Table 1: Per-layer Sparsity for VGG-19, before and after PSBWN.

Layer	Full Precision(Before)		PSBWN	
	Sparsity	width	Sparsity	width
Conv1	0%	32 bit	0%	32 bit
Conv2	0%	32 bit	82.8%	2 bit
Conv3	0%	32 bit	88.4%	2 bit
Conv4	0%	32 bit	81.9%	2 bit
Conv5	0%	32 bit	97.3%	2 bit
Conv6	0%	32 bit	96.2%	2 bit
Conv7	0%	32 bit	87.5%	2 bit
Conv8	0%	32 bit	92.1%	2 bit
Conv9	0%	32 bit	98.0%	2 bit
Conv10	0%	32 bit	95.1%	2 bit
Conv11	0%	32 bit	96.9%	2 bit
Conv12	0%	32 bit	93.7%	2 bit
Conv13	0%	32 bit	96.1%	2 bit
Conv14	0%	32 bit	97.4%	2 bit
Conv15	0%	32 bit	72.9%	2 bit
Conv16	0%	32 bit	75.8%	2 bit
fc-4096	0%	32 bit	0%	32 bit
fc-4096	0%	32 bit	0%	32 bit
Fc-256	0%	32 bit	0%	32 bit
Mean Sparsity	0%	32 bit	90.1%	

Table 2: Classification Results on Caltech–UCSD Birds 200 (CUB-200).

Model	**#Params**	**#FLOPS**	**Inference speed**	**Size**	**Top1/Top5**
Baseline VGG16	135M	30.9B	0.009142	541MB	62.9/85.5
Baseline VGG19	140M	39.2B	0.010327	562.6MB	60.3/87.2
Baseline ResNet18	11.2M	3.6B	0.005730	45.0MB	60.7/87.7
Pruned VGG16	60M	6.3B	0.006941	240.6MB	63.1/85.8
Pruned VGG19	69M	8.1B	0.008159	277.2MB	73.2/92.1
Pruned ResNet18	2.1M	1.1B	0.004912	8.7MB	63.2/88.1
Quantized VGG16	135M(2-bit, 92% zeros)	16x less	0.007741	33.8MB	72.0/77.8
Quantized VGG19	140M(2-bit, 90% zeros)	16x less	0.008950	35.0MB	68.5/90.4
Quantized ResNet18	11.2M(2-bit, 85% zeros)	16x less	**0.002011**	2.8MB	73.9/93.4
MobileNet V1	3.0M	583.0M	0.095247	13.9MB	80.4/91.1
MobileNet V2	2.0M	406.M	0.167193	10.1MB	82.1/89..0
ShuffleNet V1	**1.0M**	146.0M	0.052149	4.9MB	**83.0/96.3**
ShuffleNet V2	1.3M	149.7M	0.019520	6.0MB	79.0/89.9
FD-MobileNet	1.9M	148.4M	0.071754	8.2MB	50.2/77.5
PSBWN(proposed) VGG16	60M(2-bit, 90% zeros)	16-24x less	0.002741	15.0MB	65.1/82.7
PSBWN(proposed) VGG19	69M(2-bit, 91% zeros)	16-24x less	0.003192	17.3MB	66.5/88.3
PSBWN(proposed) ResNet18	2.1M(2-bit, 74% zeros)	16-24x less	0.002551	0.5MB	66.7/89.1

Here, and in Tables 3 and 4, the "Baseline model" is the model obtained by transfer learning, and M and B indicate Million and Billion, respectively. MB also indicates size in Megabytes. The total FLOPS values after TTQ should be determined on custom or specialized hardware, and so exact values have not been listed here. The inference is also measured in the number of seconds per image. Values written in bold represent the best[1].

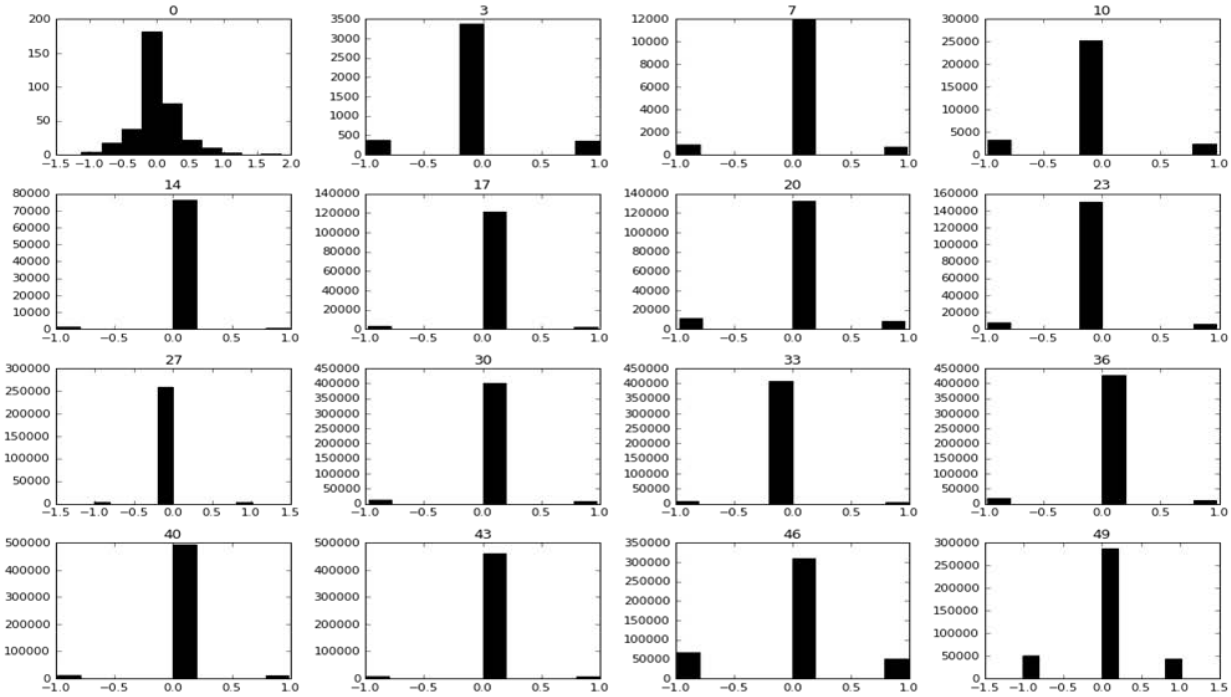

Figure 2: Visualization of the Quantized Convolution Layers in a VGG-19 Network after PSBWN. All Convolution Layers Except the First Convolution Layer are Quantized.

Table 3: Classification Results on CALTECH 256.

Model	#Params	#FLOPS	Inference	Size	Top-1/Top-5
Baseline VGG16	135M	30.9B	0.008294	541MB	63.7/83.9
Baseline ResNet18	11.3M	3.6B	0.004936	45.3MB	63.7/83.9
Pruned VGG16	59M	6.3B	0.007445	238MB	67.6/85.9
Pruned ResNet18	2.3M	1.1B	0.004459	8.8MB	63.6/82.1
Quantized VGG16	135M(2-bit)	16x less	0.002764	33.8MB	55.3/77.8
Quantized ResNet18	11.3M(2-bit)	16x less	0.001645	2.8MB	67.3/86.5
MobileNet V1	3.0M	583.16M	0.108270	14.0MB	45.8/69.3
MobileNet V2	2.0M	406.47M	0.158287	10.6MB	47.6/71.1
ShuffleNet V1	1.1M	146.11M	0.018389	4.8MB	40.0/64.0
ShuffleNet V2	1.3M	149.9M	0.020770	6.2MB	35.2/58.3
FD-MobileNet	1.9M	148.4M	0.006620	8.6MB	23.9/42.5
PSBWN(Proposed)on VGG16	59M (2-bit, 90% zeros)	16-24x less	0.002561	14.8MB	65.4/84.4
PSBWN(Proposed)on ResNet18	11.3M(2-bit, 72% zeros)	16-24x less	0.001382	0.5MB	64.1/82.1

4.4 On Oxford Flowers 102 Dataset

After PSBWN on VGG16 and ResNet18 and fine-tuning on the Oxford flower dataset, there was a slight improvement in top-1 and top-5 accuracies against the baseline model. Details of results for PSBWN and all the other models are shown in Table 4.

5 Discussion

PSBWN generally achieved the smallest model sizes with improved accuracies compared to the baseline model. Further discussions are made on the trade-offs.

5.1 Training Speed

In terms of training times, pruning requires three separate steps of transfer learning, pruning, and fine-tuning to retain accuracy, which significantly takes a longer time than other techniques. Pruning the ResNet18 model in this case with a batch size of 32 required an average of 160seconds/epoch for transfer learning, 900seconds/epoch for the main pruning and fine-tuning process.

In training the compact models on the same dataset with the same batch size, MobileNet was slowest with an average of 470s/epoch while FD-MobileNet has the fastest average training speed of 55seconds/epoch. Tables (2, 3 and 4) however show that FD-MobileNet has the least accuracies with the same number of training iterations. Compact models generally achieved lower accuracies compared to pruned or quantized models. This, in effect, means that compact networks require more training iterations for better performance, compared to pruning and quantization methods. PSBWN however, requires a longer training time since it involves a two-step process of first pruning and then quantizing the pruned network, which involves fine-tuning in an attempt to recover its accuracy.

5.2 Flops VS. Accuracy

The Floating Point Operations per Second (FLOPS) also saw some improvements. For the quantized VGG16, VGG19 and ResNet18, the number of FLOPS is estimated to be 16x less when deployed on custom hardware. With PSBWN, the number of FLOPS is expected to be even less. Experiments show that compact models have lower FLOP requirements compared to pruned models and would be more suitable for training and inference where computational requirements of the device are extremely low.

5.3 Inference

Inference tends to be faster in both pruned and quantized models compared to compact models. This may generally be due to the level of sparsity introduced in these models after pruning and quantization. However, PSBWN is shown to have the fastest inference time and would be suitable where test speed is of most importance.

5.4 Model Size VS. Accuracy

From experiments, it can be seen that both the VGG16 and ResNet18 models can be reduced by up to 40% after pruning with little accuracy degradation. Quantizing the weights of the models to as low as 2-bits also has little effect and in some cases, better Accuracy results. Compact models are small in size after training and do not require pruning or quantization, making them convenient for both training and inference on devices with low memory budgets. PSBWN has the smallest but requires custom hardware as a result of its weights being stored in 2 bits.

Table 4: Classification Results on Oxford Flowers 102 Dataset.

Model	#Params	FLOPS	Inference	Size	Top-1/Top-5
Baseline VGG16	134M	30.9B	0.007569	560.1MB	92.4/98.9
Baseline ResNet18	11.2M	3.6B	0.006880	45.0MB	85.0/97.3
Pruned VGG16	60M	5.4B	0.005995	240.6MB	91.0/98.4
Pruned ResNet18	2.1M	1.2B	0.006394	8.7MB	93.3/99.2
Quantized VGG16	134M(2-bit)	16x less	0.003582	35.0MB	88.6/99.6
Quantized ResNet18	11.2M(2-bit)	16x less	0.002863	2.8MB	67.3/86.5
MobileNet V1	3.3M	583.0M	0.007674	13.4MB	75.6/91.7
MobileNet V2	2.4M	406.2M	0.164391	9.8MB	85.4/91.2
ShuffleNet V1	1.0M	148.0M	0.018006	4.2MB	83.9/94.4
ShuffleNet V2	1.3M	149.7M	0.021813	5.5MB	68.5/88.8
FD-MobileNet	1.9M	148.4M	0.006792	8.0MB	48.5/78.5
PSBWN(Proposed) on VGG16	60M(2-bit, 85% zeros)	16-24x less	0.003126	15.0MB	86.8/97.4
PSBWN(Proposed) on ResNet18	11.2M(2-bit, 70% zeros)	16-24x less	0.002571	0.5MB	95.1/98.9

6 Conclusions

In this study, experiments are done with pruning and quantization of popular Convolutional Neural Networks (CNNs) on large pretrained CNN models. Recent compact networks (MobileNet V1 and V2, ShuffleNet V1 and V2, FD-MobileNet) are also investigated. A combined method of pruning and quantization, PSBWN is proposed. PSBWN occupies the smallest storage space, with its weights stored in 2-bits. Due to the large amount of sparsity present in PSWBN, computations made ignoring the zero weights ensures less computational cost. This allows better energy efficiency while still maintaining and sometimes achieving better accuracy compared to the baseline models. In future work, the authors plan to further investigate combining pruning with other methods, such as Optimal Brain Surgeon [34], tensor train decomposition [35], and alternative quantization methods, on several other network architectures with the goal of finding even more efficient ways to reduce model size and device resource usage, and accelerate inference without requiring custom hardware accelerators.

ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China under grant No.6187601.

REFERENCES

[1] P LeCun Yann, Bottou L, Bengio Y and Haffner (1998). Lecun-01a. Proc. IEEE, vol. 86, no. 11, pp. 2278–2324.
[2] A Krizhevsky, I Sutskever and G E Hinton (2012). ImageNet Classification with Deep Convolutional Neural Networks. Adv. Neural Inf. Process. Syst., pp. 1–9.
[3] K Simonyan and A Zisserman (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. pp. 1–14.
[4] K He, X Zhang, S Ren and J Sun (2016). Deep Residual Learning for Image Recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
[5] C Szegedy, et al. (2015). Going deeper with convolutions. Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., vol. 07-12-June, pp. 1–9.
[6] F Chollet (2017). Xception: Deep learning with depthwise separable convolutions. Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017, vol. 2017-Janua, pp. 1800–1807.
[7] Y Le Cun, J S Denker and S A Solla (1990). Optimal Brain Damage. Adv. Neural Inf. Process. Syst., vol. 2, no. 1, pp. 598–605.
[8] H Li, A Kadav, I Durdanovic, H Samet and H P Graf (2016). Pruning Filters for Efficient ConvNets. No. 2016, pp. 1–13.
[9] J M Alvarez and M Salzmann (2017). Compression-aware Training of Deep Networks. No. Nips.
[10] W Chen, J T Wilson, S Tyree, K Q Weinberger and Y Chen (2015). Compressing Neural Networks with the Hashing Trick.
[11] F N Iandola, S Han, M W Moskewicz, K Ashraf, W J Dally and K Keutzer (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. pp. 1–13.
[12] Y Gong, L Liu, M Yang and L Bourdev (2014). Compressing Deep Convolutional Networks using Vector Quantization. pp. 1–10, 2014.
[13] S Arora, A Bhaskara, R Ge and T Ma (2013). Provable Bounds for Learning Some Deep Representations.
[14] W Liu, et al. (2017). Hybrid Pruning: Thinner Sparse Networks for Fast Inference on Edge Devices. Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017, vol. 2017-Janua, no. September, pp. 1–10.
[15] S Zhou, Y Wu, Z Ni, X Zhou, H Wen and Y Zou (2016). DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. Vol. 1, no. 1.
[16] C Zhu, S Han, H.Mao and W J Dally (2016). Trained Ternary Quantization. pp. 1–10.
[17] A G Howard, et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.
[18] M Sandler, A Howard, M Zhu, A Zhmoginov and L C Chen (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks.
[19] X Zhang, M Lin and J Sun (2017). ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices.
[20] Y D Kim, E Park, S Yoo, T Choi, L Yang and D Shin (2015). Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications. pp. 1–16.
[21] L Sifre and S Mallat (2014). PhD Thesis, Ecole Polytechnique, CMAP Rigid-Motion Scattering For Image Classification.
[22] M Denil, B Shakibi, L Dinh, M Ranzato and N de Freitas (2013). Predicting Parameters in Deep Learning. pp. 1–9.
[23] Z Liu, J Li, Z Shen, G Huang, S Yan and C Zhang (2017). Learning Efficient Convolutional Networks through Network Slimming. Proc. IEEE Int. Conf. Comput. Vis., vol. 2017-Octob, pp. 2755–2763.
[24] J H Luo, J Wu and W Lin (2017). ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. Proc. IEEE Int. Conf. Comput. Vis., vol. 2017-Octob, pp. 5068–5076.
[25] P Molchanov, S Tyree, T Karras, T Aila and J Kautz (2016). Pruning Convolutional Neural Networks for Resource Efficient Inference. No. 2015, pp. 1–17.
[26] F Li, B Zhang and B Liu (2016). Ternary Weight Networks. No. Nips.
[27] Z Qin, Z Zhang, X Chen and Y Peng (2018). FD-MobileNet: Improved MobileNet with a Fast Downsampling Strategy.
[28] L Sifre and S Mallat (2014). PhD Thesis, Ecole Polytechnique, CMAP Rigid-Motion Scattering For Image Classification.
[29] S Ioffe and C Szegedy (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.
[30] A Paszke, et al. (2017). Automatic differentiation in PyTorch. No. Nips, pp. 1–4.
[31] P Welinder, S Branson, T Mita and C Wah (2010). Caltech-UCSD birds 200. CalTech, vol. 200, pp. 1–15.
[32] G Griffin, A Holub and P Perona (2007). Caltech-256 object category dataset. Caltech mimeo, vol. 11, no. 1, pp. 20.
[33] M Nilsback and A Zisserman (2008). Automated flower classification over a large number of classes.
[34] B Hassibi and D Stork (1993). Second order derivaties for network prunning: Optimal brain surgeon. Adv. NIPS5, pp. 164–171.
[35] A Novikov, D Podoprikhin, A Osokin and D Vetrov (2015). Tensorizing Neural Networks. pp. 1–9.