DISCRIMINATIVE SPARSIFICATION AND BINARIZATION OF CONVOLUTION NEURAL NETWORKS

A Thesis

by

QING JIN

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Xiaoning Qian
Committee Members,	P. R. Kumar
	I-Hong Hou
	Jianhua Huang
Head of Department,	Miroslav M. Begovic

December 2018

Major Subject: Computer Engineering

ABSTRACT

Convolution neural networks have become one of the dominating deep learning models, especially for computation vision tasks such as image classification and segmentation. Dense convolution filters are inefficient, due to huge amount of full precision multiplications involved in their computation. Motivated by previous successes on sparse models and binary models, we propose a Max-Min technique to train sparse and binary convolution neural networks with fixed structures. As opposed to previous methods, the network structure is consistent during the training phase, which has potential advantage with respect to power and memory management for hardware implementation. Computation complexity of the proposed techniques is analyzed to compare with the dense structures, showing significant reduction on the number of floating point operations (FLOPs). Specific techniques, including training tricks and structural augmentation, are discussed to facilitate fast and correct convergence, and to alleviate potential accuracy degradation introduced by these accelerating techniques. The proposed techniques are applied on several most successful architectures to obtain their sparse and binary versions, including AlexNet for both techniques, VGGNet, Inception-v3, and ResNet for sparsification. Extensive experiments on benchmark datasets (MNIST and CIFAR10) are conducted, demonstrating the effectiveness of the proposed techniques empirically with comparable prediction performance between the original dense models and their sparse or binary versions.

DEDICATION

To my parents and my sister.

ACKNOWLEDGMENTS

It is my greatest honor to have the opportunity to be a student of Professor Qian. Working with him is a comfortable enjoyment, and I always felt being cared by him on different aspects, including my study and life. I have acquired a broad range of knowledge from countless technical discussions with him, and learned a lot from his working style and his attitude to students. This will be the most valuable treasure I obtained in the Texas A&M University.

I would like to thank the hard-working professors in the university for their intuitive and responsible teaching, especially Professor P. R. Kumar, John Keyser, Ulisses Braga-Neto, Anxiao Jiang, Laszlo B. Kish and Shuiwang Ji. I learnt a lot of insights from taking their courses. I would also appreciate other students in our group, especially Weizhi Li, Chungchi Tsai, Kai He, Xueting Liu, Xiaoqian Jia and Ehsan Hajiramezanali, and also my friends Tingjui Chang, Yukun Tan, Dongzuo Tian, Zishuo Li, and Boyuan Gong for their help during my living in College Station. Finally, I would like to express my gratitude to my parents and my sister for their continuous encouragement during my study.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Xiaoning Qian and Professors P. R. Kumar and I-Hong Hou of the Department of Electrical & Computer Engineering and Professor Jianhua Huang of the Department of Statistics.

All work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Recent advancement in deep learning, thanks to much more accessible and powerful computing resources, has considerably promoted the development of several real-world applications, including computer vision, speech recognition, natural language processing, and biomedical data analysis. Among the state-of-the-art deep model architectures, convolution neural networks (CNNs) have had enormous successes in various applications, especially in computer vision tasks, such as image classification and segmentation. Recent development of computer vision almost synchronizes the evolution of CNNs, with many successful architectures proposed in the literature, such as AlexNet [1], VGGNet [2], Inception-v3 [3] and ResNet [4, 5] to name a few. However, the most effective models consist of tens to thousands of convolution layers, and training these models is computationally expensive, demanding prohibitive computing resources, which can be up to thousands of GPUs or TPUs. These advanced devices are still costly and power-consuming, not accessible to general users, and not suitable for mobile or embedded devices with tight resource constraints. Moreover, even inference using huge models may pose stringent requirements on these portable devices, regarding energy consumption and computation speed. Therefore, to further expand the territory of deep learning applications, it is critical to implement efficient architectures with high prediction performance and reduced computation complexity.

Several software and hardware speed-up solutions have been proposed recently to reduce model size and computation complexity, and two most popular techniques makes models sparse or quantized [6–19]. As shown in Fig. 1.1, generic CNNs use full-precision kernel weights with dense structure. Since each kernel convolution constitutes a huge amount of real-value multiplications, training and inference with these models can be very inefficient. On the other hand, sparse CNNs only use a certain portion of kernel weights, and thus have the potential to reduce the number of multiplications significantly. Binary CNNs adopt binary values of weights, wherein real-value multiplications are eliminated and only additions are necessary for convolution operations.

To improve computation efficiency, sparse models only have a certain portion of model weights

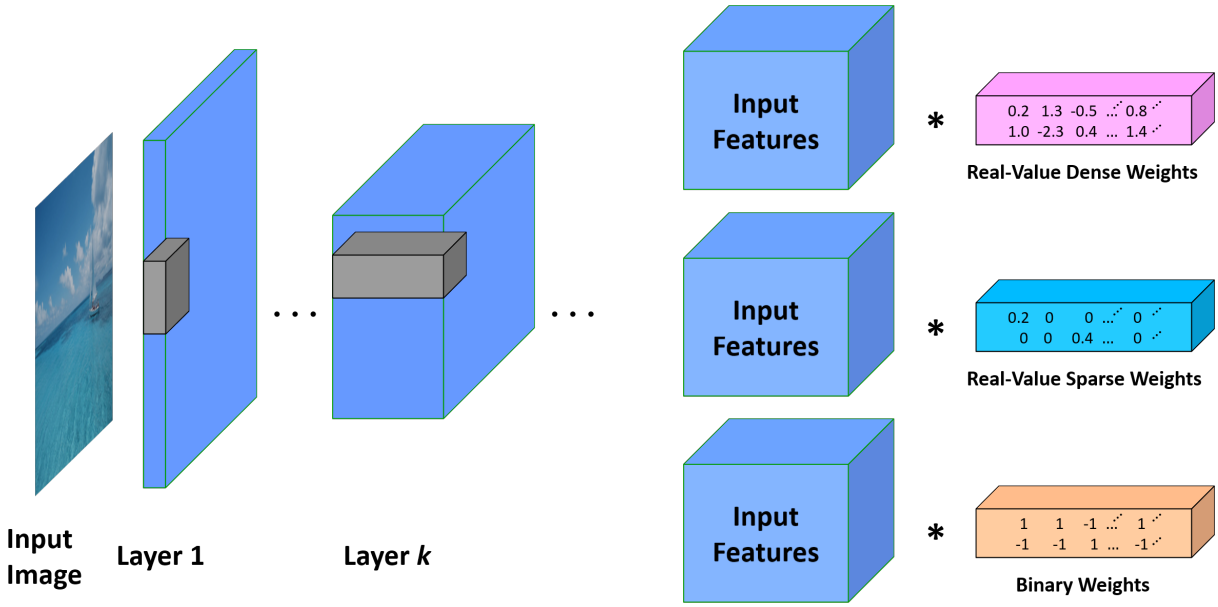
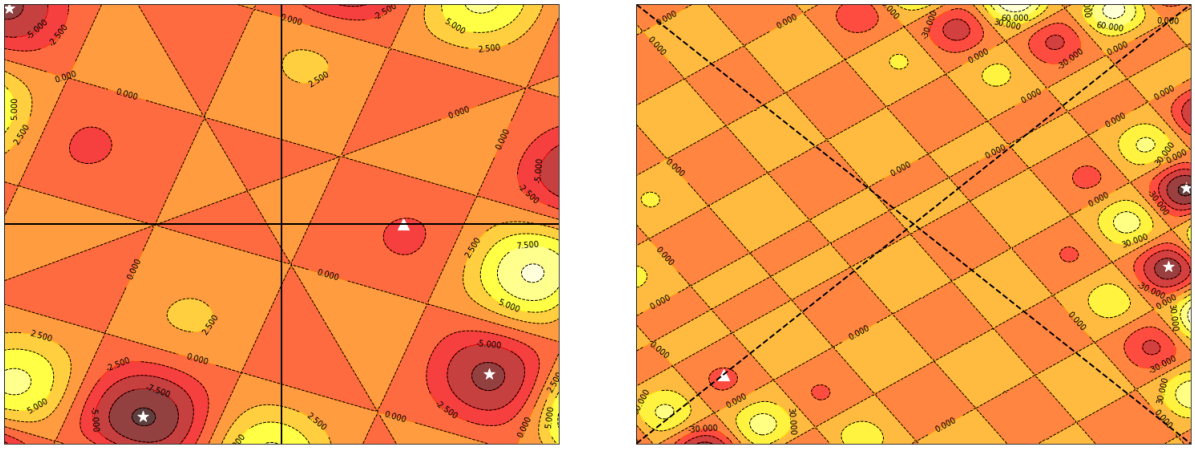


Figure 1.1: Comparison between dense, sparse and binary CNNs. (a) Generic convolution neural networks (the gray part is convolution kernel); (b) Illustration of dense, sparse and binary convolution operation.

as nonzero values, leaving the remaining part vanishing [6–11]. This way, the computation complexity of convolution can be reduced sharply, as only multiplication and addition with nonzero values are required. The underlying principle is that the original model might be redundant, and some weights are therefore unnecessary and can be eliminated by setting to zero. Two methods are widely adopted to achieve sparse models, namely regularization and approximation. Regularization penalties models directly for its non-vanishing parameters, where penalty terms are included in the objective value, which is optimized and is typically the loss function. ℓ_1 regularization and its variants are the most popular methods. Regularization attains sparse models through training, and is beneficial for complex models to avoid overfitting problem. However, the resulting structure is not predictable, as it is not clear at the beginning which part of the model weights will ultimately vanish, and different input data might result in completely different sparse structures. This problem is more serious for deep neural networks, which typically comprise massive parameters and lack interpretability. Approximation method, on the other hand, involves approximating



(a) (b)

Figure 1.2: Example contour plots showing the optimal solutions for dense, sparse and binary models. These contour plots shows the optimal sparse solution (a) or the optimal binary solution (b) can locate far away from the optimal dense solution. Optimal solutions in the total space are marked as white star, while those in the subspaces are marked as white triangle. The bold solid lines in (a) are the two axes, and the bold dashed lines in (b) are the two lines $y = \pm x$.

the convolution results of complex models through sparse matrices, where sparse approximation theory usually plays key role, such as low-rank expansion and sparse decomposition. As an example, in [8], sparse model is obtained after training the original complex model, and fine-tuning is necessary after initial sparsification.

As another accelerating technique, binarization converts model parameters or even input data into binary values, replacing multiplication of real numbers with addition or even bitwise operations [12–19]. Typical training strategies for binary models are inspired by binary approximation of matrix with minimum quantization error, and binary are commonly accompanied by sparse. However, binary models suffer from significant performance reduction, and high inference accuracy demands specific strategy with excess computation complexity. As an example, [18] proposes one technique where a set of binary convolution filters are stochastically generated with untrainable weights, followed by full precision linear layer with trainable weights. Although less parameters are necessary to update in the training phase, more channels are requisite for high accuracy, and the computation is still complex with dense addition for binary convolution.

Both existing sparsification and binarization techniques make implicit assumptions to some extent that the optimal solution of sparse or binary models locates nearby the optimal solution of the original dense models in the solution space, as the training strategies proposed are usually inspired by approximation with minimum error. This methodology can be suspicious, as the solution manifold might be complicated and delicate, making optimal models with different characteristics potentially far away (examples are shown in Fig. 1.2), and thus require different strategies for training. Also, it is still unclear if performance reduction for sparse or binary models is due to their approximating nature, or comes from the algorithms finding the approximate models.

Research on discriminative fast approximation methods with specified model structures can be beneficial to partly answer the above two questions. With a specified model structure during the training phase, we are restricted in a subspace of such sparse or binary models with the given structure when searching for the optimal solution, without considering the optimal solution for the original dense model. Also, searching for the optimal solution in a specific subspace for each compression method reduces bias for comparison between different methods. In order to do this, we propose a new technique to train sparse and binary sparse models, where the model has the same specified structure during training as the final model used for inference.

The thesis is organized as follows. In Chapter 2, we will discuss a discriminative sparsification algorithm to implement convolution neural networks with only two weights in each kernel, as opposed to $k \times k$ that is commonly used. With training techniques detailed, computation complexity in terms of the number of floating point operations (FLOPs) and model size is analyzed, which demonstrates efficiency improvement of the sparse model as opposed to their dense version. Specific initialization scheme is proposed, which is important for correct convergence of training the sparse models. Experimental results on datasets of MNIST [20] and CIFAR10 [21] are provided to verify the effectiveness of the proposed technique, where sparse versions of several architectures are compared with their dense counterparts, including AlexNet, VGGNet, Inception-v3 and ResNet. Chapter 3 discusses discriminative binarization algorithm, which is extended from the sparsification algorithm to reduce the number of multiplications further. Two methods are

proposed to obtain binary sparse models, and structural augmentation technique is employed to reduce accuracy degradation. Experimental results of applying the technique to AlexNet on datasets MNIST and CIFAR10 show the effectiveness of the proposed technique. Future works are discussed in Chapter 4, including sparsification technique of fully-connected layers, transfer learning with sparse models, and hardware implementations for the proposed technique.

2. DISCRIMINATIVE SPARSIFICATION

In this chapter, we will discuss a training technique to obtain sparse convolution neural networks with fixed structures, where the number of weights in each $k \times k$ convolution kernel is restricted to 2 instead of $k \times k$. The sparse structure is fixed during training step, which has the potential advantage for power and memory management in hardware implementation. In this proposed technique, nonzero elements in each kernel are determined with specific principles, which will be discussed before the detailed description of the algorithm. To show the efficiency of improvement provided by the proposed technique, computational complexity of sparse models is compared with their original dense versions. For correct and fast convergence of training deep neural networks, initialization is critical, and the specific initialization scheme is analyzed for the sparse models. The effectiveness of the proposed technique is verified empirically by extensive experiments on two commonly adopted benchmark testbeds, MNIST [20] and CIFAR10 [21], where the proposed technique is applied to several most successful deep models, including AlexNet [1], VGGNet [2], Inception-v3 [3], and ResNet [4, 5].

2.1 Sparse Convolution Neural Networks

Convolution neural networks involve massive weight parameters, especially for deep models with tens to thousands of convolution layers. Complicated convolutions make deep models powerful on extracting informative feature representations, at the expense of enormous computation complexity, together with the potential overfitting problem. On the other hand, it is not conclusive if all the parameters are indispensable, especially when we take into account that various sparse models are proposed in the literature [6–11] for model compression and computation speedup, in order to enhance training and inference efficiency. Previous methods by sparsification can be roughly categorized as regularization or approximation. Regularization techniques take advantage of the sparsity of parameters optimized with structural regularization, such as ℓ_1 regularization and its variants, to make most parameters in the model vanish. Approximation techniques leverage

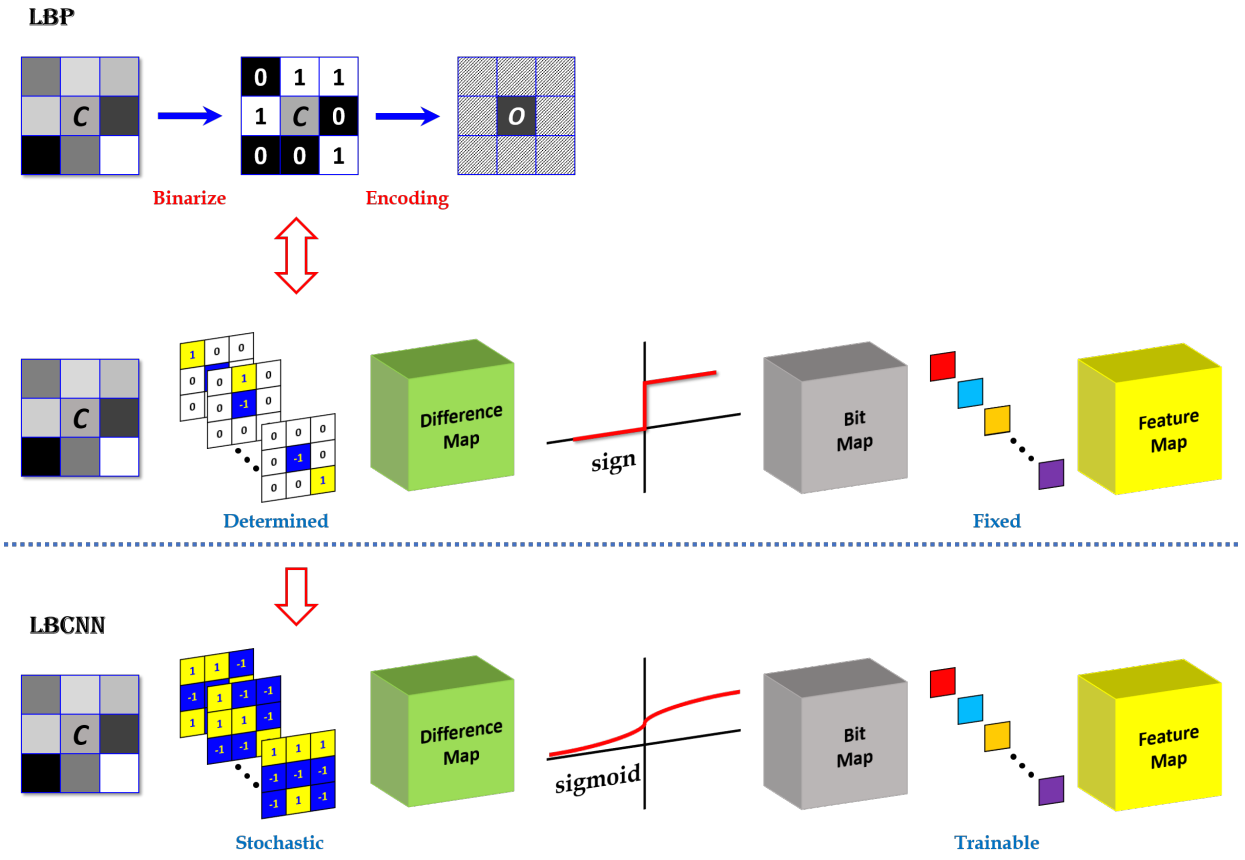


Figure 2.1: LBP viewed as convolution neural network and evolved to LBCNN. [18]

sparse approximation theory, featuring an extensive use of low-rank expansion and sparse decomposition methods. However, if it can be safely assumed *a priori* that some sparse structure is valid for the classification task, it is natural to ask whether we can restrict the model with that fixed structure, making the model to possess consistent sparsity during the training phase. In the following sections, we will try to answer this and propose one training technique for models with fixed sparse structure. Some theoretic analysis of the initialization scheme will be also provided, which is important for correct and fast convergence during training.

2.2 Algorithm Details

As mentioned in the previous section, we want to specify the structure of the final sparse model before training. This basic idea is inspired by the local binary convolution neural net-

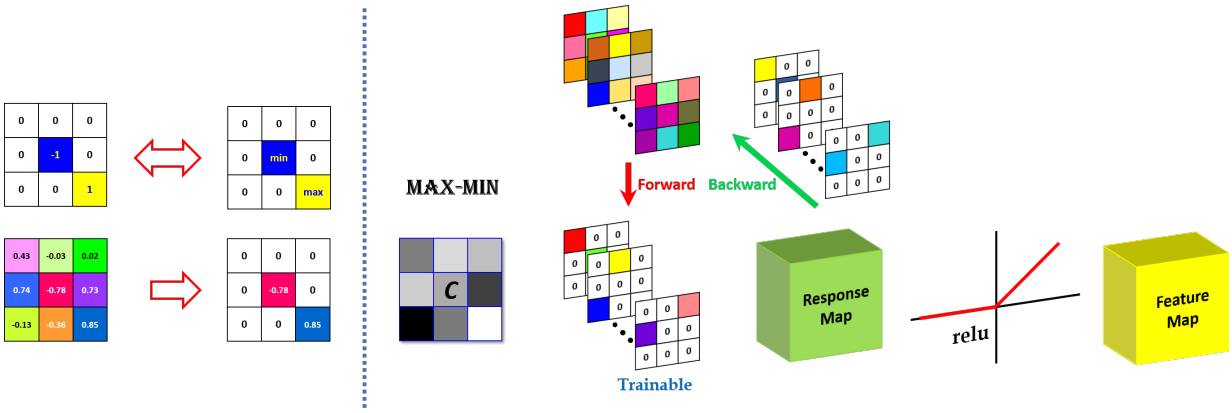


Figure 2.2: Generalize LBCNN by the kernel-wise Max-Min technique.

work (LBCNN) [18]. As illustrated in Fig. 2.1, the authors observed the similarity between the traditional local binary pattern (LBP) operator and the standard convolution operator, and further demonstrated that it is possible to recover the LBP operation through eight 2-sparse binary convolution filters followed by linear connected layers. The authors replaced the standard convolution layers with a bunch of fixed binary convolution layers, and fed their outputs to linear connected layers. Weights in the binary convolution layers only take values from $-1, 0, +1$ and are generated stochastically before training, while the linear connected layers employ trainable full precision parameters. The authors validated that the prediction performance degradation of LBCNN is small considering the significantly reduced computation. However, it is a little arbitrary to stochastically generate the fixed binary convolution layers in LBCNN as it has the risk of ignoring the data characteristics when training the deep networks. Different data might require different binary convolution filters, while tuning on different candidate filters is almost impossible in the framework of LBCNN. Using specific binary convolution layers, even though generated stochastically, will inevitably introduce prior informantion arbitrarily specified, resulting in potential bias with degenerated prediction performance. Motivated by that, for our "discriminative" sparsification, we expect to determine the nonzero kernel weights through training, where data help determine the contributing kernel elements and their corresponding weights.

In order to do this, we notice that for 2-sparse binary filters, where kernel weights only take

values from $\{+1, 0, -1\}$, the only two nonzero elements are the maximum and minimum in the kernel. Enlightened by this observation, as shown in Fig. 2.2, we propose a new training strategy, where all kernel elements are replaced by zero, except the maximum and minimum, which remain intact. However, the non-extremal elements cannot be discarded, because we would like to have the flexibility to learn the positions of final nonzero elements through training so that the final 2-sparse binary filters can better capture data characteristics than the existing LBCNN does. For that, we use sparse kernels only for forward and backward propagation, and dense kernels are recovered during parameter updates. This is the same strategy that has been adopted for training binary filters [12]. Since only extremal elements contribute to the final results, during each step of parameter updates, only the original extremal elements are modified. These parameters may retain or lose their extremal status afterwards. When the training is finished, the optimal model will need only two parameters for each convolution kernel instead of $k \times k$ as in standard CNNs so that we achieve the reduced deep models. We will call this training strategy the *Max-Min* technique with its pseudo-code provided in Algorithm 1. The obtained sparse CNN will be called kernel-wise sparse CNN, since the extremal elements (maximum and minimum) are determined for each kernel or along the kernel dimension.

2.3 Computation Complexity Analysis

Convolution neural networks typically involve enormous computation complexity, as each kernel needs to do convolution with multiple local regions from inputs of the corresponding layer. For comparison of different models, the number of parameters is a specification of model size, and the number of full-precision multiplications (FLOPs) is widely adopted to characterize computation complexity. As an example, VGG19 employs 16 convolution layers and 3 fully-connected layers. The whole model contains 143.6 million learnable parameters, with 20 million for convolution layers and 123.6 million for fully-connected layers. Although the number of parameters in fully-connected layers dominates in VGG19, the trend is reversed when investigating the number of FLOPs. For inference on one image of size 224×224, for example, from the ImageNet dataset [22], 19.5 billion FLOPs are required for convolution and 123.6 million FLOPs for fully-

Algorithm 1 SGD training of kernel-wise sparse CNN with Max-Min technique. C is the cost function for minibatch and the function $\mathrm{sparse}(w)$ retains only the maximal and minimal elements along kernel dimensions while setting all others to zero. L is the number of layers.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_{t-1}^{\mathrm{sp}} \leftarrow \mathrm{sparse}(w_{t-1})$

For $k = 1$ to L, compute h_k knowing h_{k-1}, w_{t-1}^{sp} and b_{t-1}

2. Backward propagation:

Initialize output layer's activation gradient $\frac{\partial L}{\partial h_L}$

For $k = L$ to 2, compute $\frac{\partial L}{\partial h_{k-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and w_{t-1}^{sp}

3. Parameter update:

Compute $\frac{\partial L}{\partial w_{t-1}^{\mathrm{sp}}}$ and $\frac{\partial L}{\partial b_{t-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and h_{k-1}

$w_t \leftarrow w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}^{\mathrm{sp}}}$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$

connected matrix multiplication, from which we can see the significance of making convolution layers sparse with respect to model reduction.

With the proposed technique, at each step of inference, there are only 2 multiplication for each kernel, instead of $k \times k$, so the reduction in FLOPs is around $\frac{1}{2}k^2$. The training phase can benefit more from it, since during forward and backward propagation, only two parameters in each kernel are involved in the computation and need to be updated at each step, and calculation of gradients with respect to other parameters can be saved. As an example, for sparse version of VGG19, there are 4.46 million parameters in convolution layers and 4.4 billion FLOPs will be required for inference since all convolution layers have the same kernel size 3×3. The total number of FLOPs for inference is reduced from 19.65 billion to 4.5 billion. Detailed comparison for other deep architectures will be presented in a latter section.

2.4 Initialization

Initialization is important for correct and fast convergence of training deep models, which has been well studied in the existing literature [23–30]. The basic conclusion is that parameters

need to be carefully initialized on the critical line, which separates the chaotic and ordered phases considering network training dynamics. Only in this case the model is trainable without gradient vanishing or exploding issues [26]. For standard CNNs, the relationship between variances of outputs in two adjacent layers is given by

$$\mathrm{Var}[\mathbf{y}_i^{(l)}] = \frac{1}{2} n_l \mathrm{Var}[\mathbf{W}_{ij}^{(l)}] \mathrm{Var}[\mathbf{y}_i^{(l-1)}], \tag{2.1}$$

where $\mathbf{y}^{(l)}$ is the input of the activation function at the l-th layer; $\mathbf{W}^{(l)}$ denotes kernel weight matrix; and n_l is the number of filters. Non-vanishing and non-exploding gradients for convergence are typically guaranteed by avoiding reducing or magnifying the magnitudes of input signals exponentially across layers, leading to the condition of He initialization scheme [24]

$$\frac{1}{2} n_l \mathrm{Var}[\mathbf{W}_{ij}^{(l)}] = 1. \tag{2.2}$$

For the proposed Max-Min sparsification technique, the sparse weights only contain the two extremal elements in each kernel. For the k^2 i.i.d. random variables in $\mathbf{W}$ where each follows the same cumulative distribution function (CDF) $F(w)$, the joint distribution of the minimum and maximum elements is determined by the CDF as

$$\begin{aligned} \Psi(x, y) &\equiv P(w_{\min} \leq x, w_{\max} \leq y) \\ &= P(w_{\max} \leq y) - P(w_{\min} > x, w_{\max} \leq y) \\ &= F^{k^2}(y) - \{\mathrm{relu}(F(y) - F(x))\}^{k^2} \end{aligned} \tag{2.3}$$

where relu is the rectified linear unit function defined as

$$\mathrm{relu}(x) = \max(0, x) \tag{2.4}$$

Each element has a probability of $1/k^2$ to be the maximum or minimum element, so the m-th

moment of the sparse weights $\widetilde{\mathbf{W}}_{ij}^{(l)}$ is accordingly given by

$$\begin{aligned}\mathbb{E}[\{\widetilde{\mathbf{W}}_{ij}^{(l)}\}^m] &= \frac{1}{k^2}\iint_{(x,y)\in\mathbb{R}^2}(x^m+y^m)\frac{\partial^2\Psi}{\partial x\partial y}\mathrm{d}x\mathrm{d}y \\ &= \int_{x=-\infty}^{x=+\infty} x^m(1-F(x))^{k^2-1}\mathrm{d}F(x) + \int_{y=-\infty}^{y=+\infty} y^m(F(y))^{k^2-1}\mathrm{d}F(y) \\ &= \int_{x=-\infty}^{x=+\infty} x^m[F^{k^2-1}(x)+(1-F(x))^{k^2-1}]\mathrm{d}F(x)\end{aligned} \tag{2.5}$$

With the common assumption of Gaussian weight distribution with zero mean for the standard CNNs, the above expression is simplified to

$$\mathbb{E}[\{\widetilde{\mathbf{W}}_{ij}^{(l)}\}^m] = \int_{x=-\infty}^{x=+\infty}[x^m+(-x)^m]F^{k^2-1}(x)\mathrm{d}F(x) \tag{2.6}$$

From the above derivation we can see that the variance of sparse weights $\widetilde{\mathbf{W}}_{ij}^{(l)}$ is given by

$$\mathrm{Var}[\widetilde{\mathbf{W}}_{ij}^{(l)}] = \mathrm{Var}[\mathbf{W}_{ij}^{(l)}]\cdot 2\int_{x\in\mathbb{R}} x^2\Phi^{k^2-1}(x)\mathrm{d}\Phi(x) \tag{2.7}$$

where k is the kernel size, $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution. The initialization scheme proposed in [24] should be adjusted accordingly as

$$\frac{1}{2}n_l\mathrm{Var}[\mathbf{W}_{ij}^{(l)}]\cdot 2\int_{x\in\mathbb{R}} x^2\Phi^{k^2-1}(x)\mathrm{d}\Phi(x) = 1 \tag{2.8}$$

For kernel size $k=3$, the scaling factor is 0.569. Experiments show that this scaling for initialization is important for fast and correct convergence.

2.5 Experiments

To verify the proposed technique, we apply it to several most successful deep models, including AlexNet, VGGNet, Inception-v3, and ResNet. For 1×1 convolution layers employed in Inception-v3 and ResNet50, there is only one element in each kernel, thus sparsification is not necessary. For models using He initialization, including VGGNet, Inception-v3, and ResNet, the correct scaling

factor is taken into account during initialization, while for AlexNet, no special initialization scheme is adopted as usual.

Model size and computation complexity for each model is compared with its sparse version in Table 2.1, where computation complexity is for inference on one sample of size 224×224, as described previously.

Model Name	Model Type	# params	FLOPs
AlexNet	dense	61.1M	715M
	sparse	59.1M	163M
VGG19	dense	143.7M	19.6B
	sparse	128.1M	4.5B
Inception-v3	dense	23.8M	6.4B
	sparse	13.6M	2.4B
ResNet18	dense	11.5M	1.8B
	sparse	2.96M	403M
ResNet50	dense	22.7M	3.8B
	sparse	13.9M	2.2B

Table 2.1: Comparison of model size and computation complexity.

The performance of the proposed technique is evaluated on MNIST and CIFAR10, whose example images are illustrated in Fig. 2.3. MNIST is a benchmark for handwritten digit number recognition and CIFAR10 is for object classification. Test accuracy for different models on both datasets is summarized in Table 2.2, where we can see that the proposed technique is effective for different deep architectures, since the performance of our sparse model is comparable to the dense model for each of the four different architectures. The plot of training and test accuracy for both dense and sparse models are illustrated in Fig. 2.4 and Fig. 2.5 for MNIST and CIFAR10, respectively. It can be seen that in each plot, the test accuracy of the sparse model converges to a similar value as its dense version does.

Figure 2.3: Example images from the datasets MNIST and CIFAR10. MNIST is from [20]. CIFAR10 is from [21].

Model Name	Model Type	Test Accuracy (%)	
		MNIST	CIFAR10
AlexNet	dense	99.49	89.93
	sparse	99.47	89.54
VGG19	dense	99.62	93.91
	sparse	99.57	93.55
Inception-v3	dense	99.38	89.80
	sparse	99.05	89.43
ResNet18	dense	99.36	92.72
	sparse	98.43	90.18
ResNet50	dense	99.01	92.90
	sparse	98.96	91.49

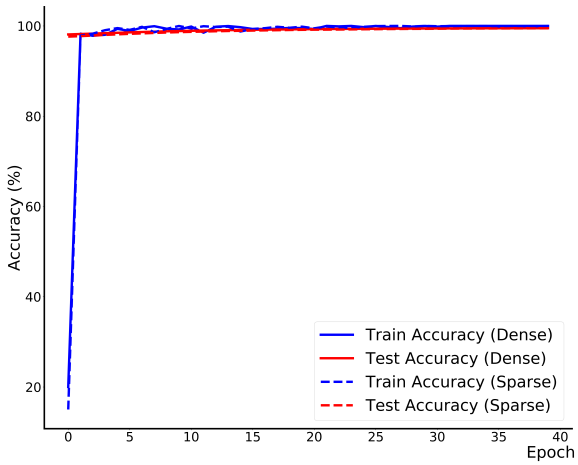
Table 2.2: Prediction performance comparison on MNIST and CIFAR10.

2.6 Discussion and Summary

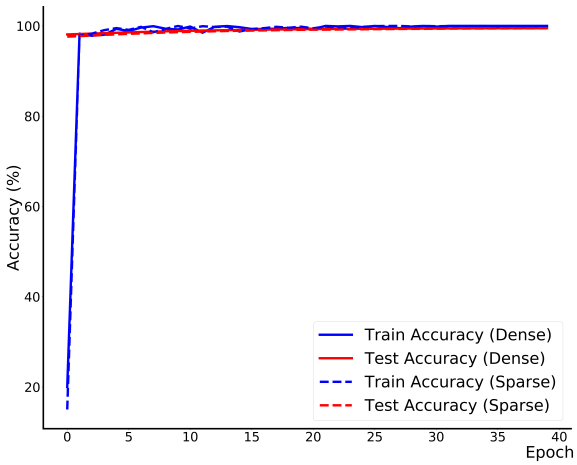
Sparsification is one common technique to reduce model size and save computation complexity, and searching optimal solution for sparse models with fixed structure. In this chapter, we propose a Max-Min technique to train sparse convolution neural networks. The sparse models obtained with our technique have fixed structures, which have the potential advantage of power and mem-

ory management for hardware implementation. Detailed algorithm for training with the proposed technique is described, together with specific techniques employed in the algorithm. Initialization scheme for the proposed Max-Min technique is discussed, which is important for

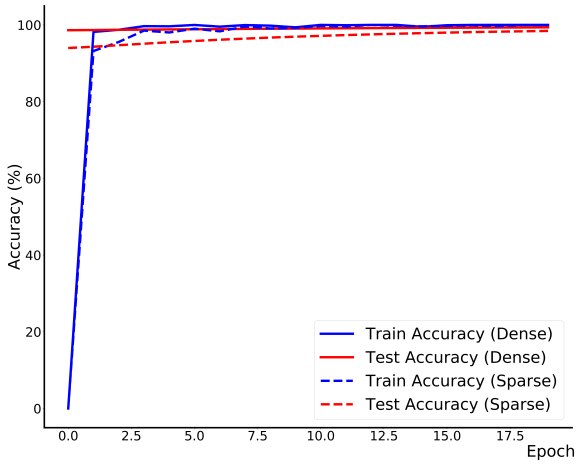
Detailed algorithm is described, together with specific techniques and initialization scheme, which is important for correct and fast convergence during the network training phase. Analysis of computation complexity shows significant improvement of model efficiency resulted from our sparse models over the dense versions, especially in terms of FLOPs. Extensive experiments on several most successful deep models with two benchmark datasets, MNIST and CIFAR10, demonstrate comparable prediction performance from the sparse models with significant computation reduction compared to dense models, which verifies the effectiveness of the proposed technique for different deep architectures.



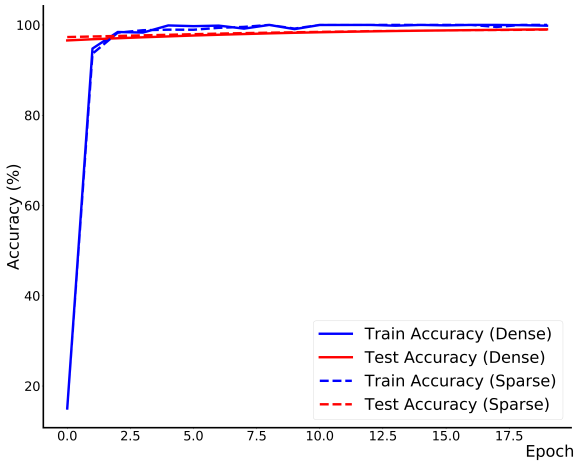
(a) AlexNet



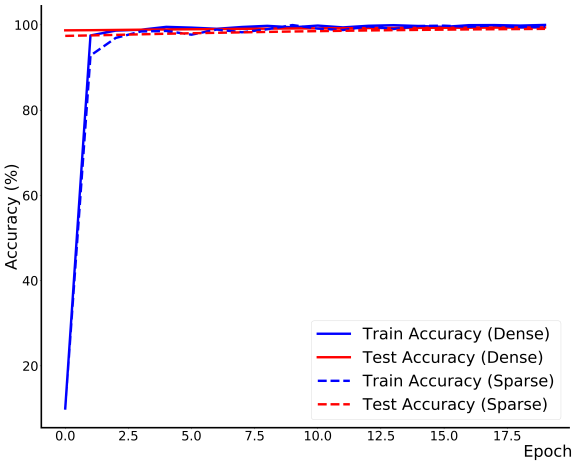
(b) VGG19



(c) ResNet18

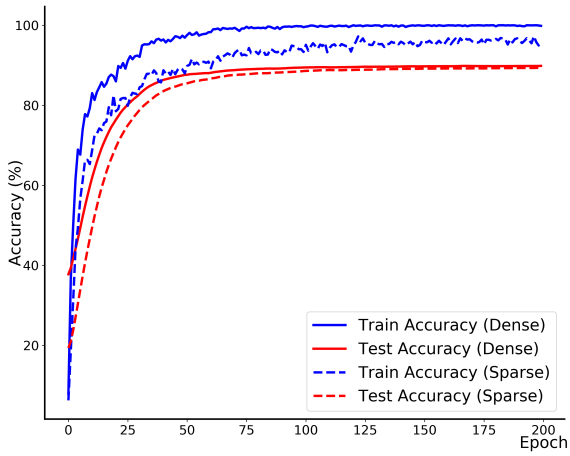


(d) ResNet50

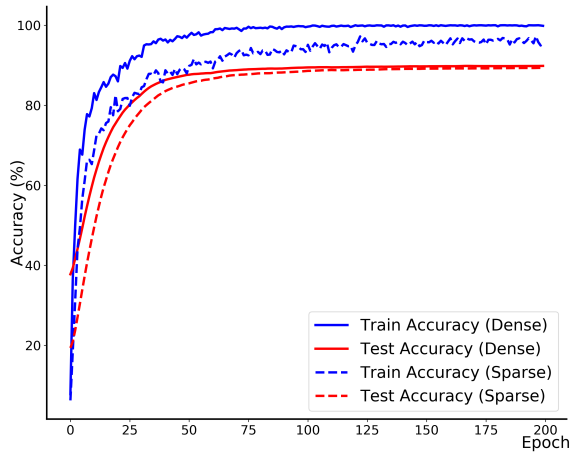


(e) Inception-v3

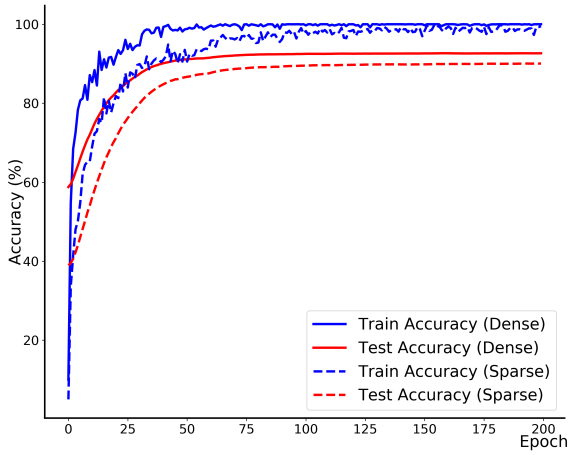
Figure 2.4: Accuracy of dense and sparse models on MNIST.



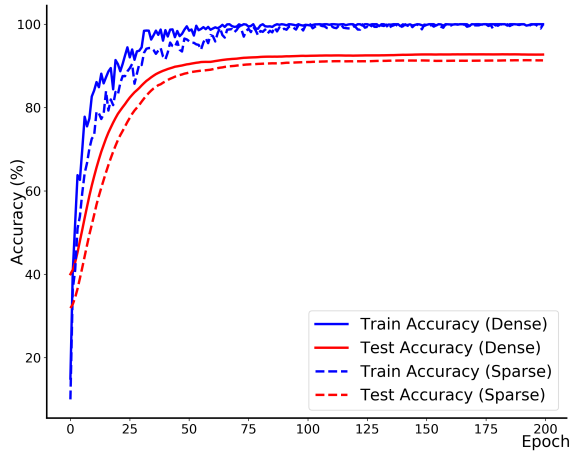
(a) AlexNet



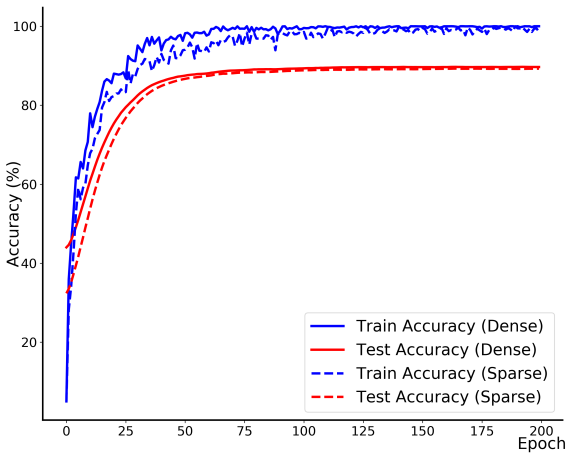
(b) VGG19



(c) ResNet18



(d) ResNet50



(e) Inception-v3

Figure 2.5: Accuracy of dense and sparse models on CIFAR10.

3. DISCRIMINATIVE BINARIZATION

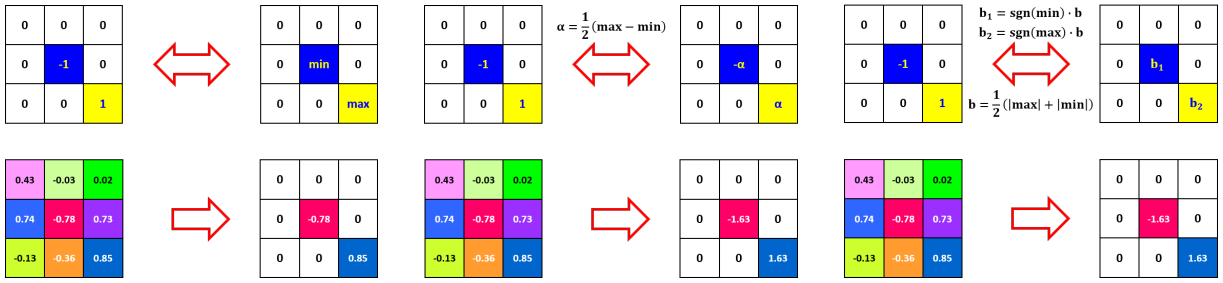
Sparsification can reduce model size and computation complexity, as the number of real-value parameters in convolution kernel is reduced. However, extensive full-precision multiplications are still involved, limiting the compression ratio of sparse models. In this chapter, we will further extend the previous technique to train binary sparse models, where multiplication is replaced by addition in convolution filters, and thus reduce model size and computation complexity to more extent. To determine these binary sparse weights, two methods with detailed algorithms are discusses. For correct and fast convergence of training, specific initialization schemes for the binary convolution filter are described. Experiments show significant accuracy reduction due to binarization, which partly answer the question about which model compression technique between sparsification and binarization is more harmful to performance. To alleviate the performance degradation, we adapt the technique proposed in local binary convolution neural networks (LBCNN) [18], where a wider internal layer is inserted to enhance representation capability of binary models. In order to further reduce computation complexity of LBCNN, a new technique, called channel-wise Max-Min sparsification, is introduced to reduce the number of parameters and floating-point operations (FLOPs) for 1×1 convolution filters, where the kernel-wise sparsification becomes ineffective. At last, for comprehensive comparison between different models including dense, sparse, and binary sparse versions, computation complexity is estimated for each of them. Experiments on CIFAR10 with different techniques applied on AlexNet are conducted for performance comparison, which shows the effectiveness of the proposed methods of training binary sparse and channel-wise sparse models.

3.1 Binary Convolution Neural Networks

Sparse models reduce the computation complexity through vanishing a certain portion of weights in convolution filters. However, they still require real number multiplications for convolution, which is slow and energy consuming. On the contrary, binary models have convolution filters

with binary weights, where nonzero elements only take values from $\{+1, -1\}$. This will reduce the computation complexity considerably, as only summation (floating-point addition) is needed for convolution. During the network training phase, binary weights are usually generated by taking the sign of full precision weights [12]. Although efficient, binary CNNs are prone to severe prediction accuracy degradation, especially for large datasets such as ImageNet. Various techniques have been proposed to tackle this problem. The authors in [14] introduced a scale factor for minimizing quantization error and proposed special training techniques for binary convolution layers. In [15] and [16], multi-bit convolution was developed to alleviate information loss introduced by aggressive quantization. The authors in [15] made the observation that quantization of the first and last layers will lead to significant accuracy reduction. Replacing ReLU with PReLU as the activation function with low learning rate for stable training has shown improved prediction performance, and a new regularization is proposed specifically for binary convolution networks [17]. [19] adopts multiple binary weight bases and multiple binary activations to alleviate the accuracy degradation. [18] makes analogy between binary convolution and local binary pattern method, and replaces standard convolution with local binary convolution, where binary convolution filters, which are generated stochastically and not trainable, are followed by linear fully connected layers with trainable full precision weights. Both the last two techniques achieve comparable accuracy as standard CNNs.

As we have successfully trained sparse models with only two parameters in each kernel and slight accuracy degradation, it is a natural question to ask whether we can take one further step to use only one full precision parameter, and thus only employ addition in each convolution filter. In the following section, we will introduce two methods for training binary sparse models, where the structure of the binary sparse models is fixed as before, and positions of nonzero elements in the binary kernels are determined during training. The two methods for training binary sparse models are illustrated in Fig. 3.1, together with the method for training sparse model introduced in the previous chapter. We will also apply the technique proposed in [18] to alleviate accuracy degradation, where a wider internal layer is inserted without introducing more full-precision multiplications.



(a) Sparse (b) Heuristic binary (c) Approximation binary

Figure 3.1: Comparison between the methods to get sparse and binary kernels. (a) is for sparse, (b) and (c) are for binary. The latter two will be discussed in the following two sections.

However, our binary filters are not generated stochastically, but trainable using data.

3.2 Heuristic Method

In order to achieve binary operation, one heuristic method is to assume a priori that each binary convolution kernel is composed of one $+1$ and one -1, with all other elements vanishing. Denoting the maximum and minimum elements in a given kernel $\mathbf{W}$ as $W_{\max}$ and $W_{\min}$, we replace these two elements with $(W_{\max} - W_{\min})$ and $(W_{\min} - W_{\max})$, respectively. The remaining part is the same as previous Max-Min technique for sparse models. With this strategy, effectively each kernel only needs one full precision parameter for forward and backward propagation. This heuristic method is summarized in Algorithm 2.

Initialization of this model is a little different, as the variance of the new binary kernel weights is different from previous results. Denote the scaled binary kernel with $(W^{(l)}_{\max} - W^{(l)}_{\min})$ and $(W^{(l)}_{\min} - W^{(l)}_{\max})$ as the only two nonzero elements by $\alpha^{(l)}\widetilde{\mathbf{B}}^{(l)}$, where $\alpha^{(l)} = W^{(l)}_{\max} - W^{(l)}_{\min}$, $\widetilde{\mathbf{B}}^{(l)}$ is a binary kernel with one $+1$ and one -1.

Based on the same result for the distribution of the two extremal elements derived in the previous chapter,

$$\Psi(x, y) = F^{k^2}(y) - \{\text{relu}(F(y) - F(x))\}^{k^2} \tag{3.1}$$

Algorithm 2 SGD training of binary sparse CNNs with heuristic Max-Min technique. C is the cost function for minibatch and the function $\mathrm{binarysparse}(w)$ replaces the maximal and minimal elements along kernel dimensions by $+1$ and -1, respectively, while sets all others zero. Δ is the difference between maximum and minimum. L is the number of layers. This algorithm is illustrated as the case of one kernel, but generalization to multiple kernels is straightforward.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_{t-1}^{\mathrm{bnsp}} \leftarrow \Delta \cdot \mathrm{binarysparse}(w_{t-1})$

For $k = 1$ to L, compute h_k knowing h_{k-1}, w_{t-1}^{bnsp} and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial L}{\partial h_L}$

For $k = L$ to 2, compute $\frac{\partial L}{\partial h_{k-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and w_{t-1}^{bnsp}

3. Parameter update:

Compute $\frac{\partial L}{\partial \Delta}$ and $\frac{\partial L}{\partial b_{t-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and h_{k-1}

$w_t \leftarrow w_{t-1} - \eta \frac{\partial L}{\partial \Delta} \cdot \mathrm{binarysparse}(w_{t-1})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$

we have the m-th moment of the binary sparse weights $\alpha^{(l)}\widetilde{\mathbf{B}}_{ij}^{(l)}$ with an even m as

$$\begin{aligned}\mathbb{E}[\{\alpha^{(l)}\widetilde{\mathbf{B}}_{ij}^{(l)}\}^m] &= \frac{1}{k^2}\iint_{(x,y)\in\mathbb{R}^2}[(x-y)^m + (y-x)^m]\frac{\partial^2\Psi}{\partial x \partial y}\mathrm{d}x\mathrm{d}y \\ &= \iint_{y>x} 2(x-y)^m(k^2-1)[F(y)-F(x)]^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\ &= \iint_{(x,y)\in\mathbb{R}^2}(k^2-1)|x-y|^m|F(x)-F(y)|^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y)\end{aligned} \tag{3.2}$$

while for odd m it is zero. With the same assumption of Gaussian random weights as before, the variance of the scaled binary weights is given by

$$\mathrm{Var}[\alpha^{(l)}\widetilde{\mathbf{B}}_{ij}^{(l)}] = \mathrm{Var}[\mathbf{W}_{ij}^{(l)}]\cdot(k^2-1)\iint_{(x,y)\in\mathbb{R}^2}|x-y|^2|\Phi(x)-\Phi(y)|^{k^2-2}\mathrm{d}\Phi(x)\mathrm{d}\Phi(y) \tag{3.3}$$

where k is the kernel size, $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution. The initialization scheme for the proposed sparse filter is thus modified from He's [24]

to

$$\frac{1}{2}n_l\mathrm{Var}[\mathbf{W}_{ij}^{(l)}]\cdot(k^2-1)\iint_{(x,y)\in\mathbb{R}^2}|x-y|^2|\Phi(x)-\Phi(y)|^{k^2-2}\mathrm{d}\Phi(x)\mathrm{d}\Phi(y)=1 \tag{3.4}$$

For kernel size $k=3$, the scaling factor is 2.105. Experiments show that this scaling for initialization is important for fast and correct convergence.

3.3 Approximation Method

A priori assumption of binary filter is heuristic, as it is not guaranteed that each sparse filter in the optimal model contains both positive and negative weights, and it is possible that some filters in the optimal binary 2-sparse model is composed of nonzero weights with the same sign. To take into account this situation, we consider to use optimal binary approximation of the sparse convolution, where the binary kernel $\widetilde{\mathbf{B}}^*$ and scale factor α^* are given by [14]

$$\widetilde{\mathbf{B}}^* = \mathrm{sign}(\widetilde{\mathbf{W}}) \tag{3.5a}$$

$$\alpha^* = \frac{1}{2}\|\widetilde{\mathbf{W}}\|_1 \tag{3.5b}$$

respectively. Here, $\widetilde{\mathbf{W}}$ denotes the 2-sparse version of the dense kernel $\mathbf{W}$. The averaging factor is $\frac{1}{2}$ instead of $\frac{1}{n}$, because there are only two non-vanishing elements in each sparse kernel. However, this scale factor can be absorbed into the training parameters, and this extra multiplication is not necessary. This approximation method is summarized in Algorithm 3.

The m-th moment of the optimal scaled binary sparse weights $\alpha^{*(l)}\widetilde{\mathbf{B}}_{ij}^{*(l)}$ with an even m is

$$\begin{aligned}\mathbb{E}[\{\alpha^{*(l)}\widetilde{\mathbf{B}}_{ij}^{*(l)}\}^m] = &\iint_{x<y<0}\frac{1}{2^{m-1}}(x+y)^m(k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y)\\ &+\iint_{0<x<y}\frac{1}{2^{m-1}}(x+y)^m(k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y)\\ &+\iint_{x<0<y}\frac{1}{2^m}[(y-x)^m+(x-y)^m](k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y)\end{aligned} \tag{3.6}$$

while for odd m it is zero.

Algorithm 3 SGD training of binary sparse CNNs with approximation Max-Min technique. C is the cost function for minibatch and the function $\mathrm{binarysparse}(w)$ replaces the maximal and minimal elements along kernel dimensions by their signs, while sets all others zero. α is the sum of the absolute value of maximum and minimum. L is the number of layers. This algorithm is illustrated as the case of one kernel, but generalization to multiple kernels is straightforward.

Require: a minibatch of (inputs, targets), previous parameters w_{t-1} (weights) and b_{t-1} (biases), and learning rate η.

Ensure: updated parameters w_t and b_t.

1. Forward propagation:

$w_{t-1}^{\mathrm{bnsp}} \leftarrow \alpha \cdot \mathrm{binarysparse}(w_{t-1})$

For $k = 1$ to L, compute h_k knowing h_{k-1}, w_{t-1}^{bnsp} and b_{t-1}

2. Backward propagation:

Initialize output layer's activations gradient $\frac{\partial L}{\partial h_L}$

For $k = L$ to 2, compute $\frac{\partial L}{\partial h_{k-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and w_{t-1}^{bnsp}

3. Parameter update:

Compute $\frac{\partial L}{\partial \alpha}$ and $\frac{\partial L}{\partial b_{t-1}}$ knowing $\frac{\partial L}{\partial h_k}$ and h_{k-1}

$w_t \leftarrow w_{t-1} - \eta \frac{\partial L}{\partial \alpha} \cdot \mathrm{binarysparse}(w_{t-1})$

$b_t \leftarrow b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$

For Gaussian random weights with zero mean, the above expression can be simplified as

$$
\begin{aligned}
\mathbb{E}[\{\alpha^{*(l)}\widetilde{\mathbf{B}}_{ij}^{*(l)}\}^m] = & \iint_{0<y<x} \frac{1}{2^{m-1}}(-x-y)^m(k^2-1)(F(x)-F(y))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
& + \iint_{0<x<y} \frac{1}{2^{m-1}}(x+y)^m(k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
& + \iint_{x<0<y} \frac{1}{2^{m}}[(y-x)^m+(x-y)^m](k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
= & \iint_{x,\,y\in\mathbb{R}^+} \frac{1}{2^{m-1}}(x+y)^m(k^2-1)|F(x)-F(y)|^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
& + \iint_{x<0<y} \frac{1}{2^{m-1}}|y-x|^m(k^2-1)(F(y)-F(x))^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
= & \iint_{y>0} \frac{1}{2^{m-1}}(|x|+y)^m(k^2-1)|F(x)-F(y)|^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y) \\
= & \iint_{(x,y)\in\mathbb{R}^2} \frac{1}{2^{m}}(k^2-1)(|x|+|y|)^m|F(x)-F(y)|^{k^2-2}\mathrm{d}F(x)\mathrm{d}F(y)
\end{aligned}
\tag{3.7}
$$

The variance of the optimal scaled binary weights is given by

$$\mathrm{Var}[\alpha^{*(l)}\widetilde{\mathbf{B}}_{ij}^{*(l)}] = \mathrm{Var}[\mathbf{W}_{ij}^{(l)}] \cdot \frac{1}{4}(k^2-1) \iint_{(x,y)\in\mathbb{R}^2} (|x|+|y|)^2 |\Phi(x)-\Phi(y)|^{k^2-2} \mathrm{d}\Phi(x)\mathrm{d}\Phi(y) \quad (3.8)$$

and the initialization scheme should be modified to

$$\frac{1}{2} n_l \mathrm{Var}[\mathbf{W}_{ij}^{(l)}] \cdot \frac{1}{4}(k^2-1) \iint_{(x,y)\in\mathbb{R}^2} (|x|+|y|)^2 |\Phi(x)-\Phi(y)|^{k^2-2} \mathrm{d}\Phi(x)\mathrm{d}\Phi(y) = 1 \quad (3.9)$$

For kernel size $k = 3$, the scaling factor is 0.527. Experiments show that this scaling for initialization is important for fast and correct convergence.

3.4 Insertion of Wider Internal Layers

The above methods can be applied to train binary sparse convolution neural networks. However, as shown in Fig. 3.2, experiments show significant accuracy degradation for both methods, meaning that too much information is lost during binarization. Also, there is some strange behavior of the approximation method, making its performance worse than the other method. The undesired behavior might be due to the information loss on the gradient, as we have not taken into account the gradient of sign and absolute value function.

To solve the problem of performance degradation, we employ the method proposed in [18], where wider internal filters are introduced, to enhance the representation capacity of the convolution filter. Wider internal filters can be viewed as introducing more quantization bits, which has the potential to improve the prediction accuracy with such enhanced representations, in addition to the full-precision values used in the following 1×1 convolution filters. The original input is processed by the binary sparse convolution layer we proposed, and then by full-precision 1×1 convolution layer. For simplicity, we will call this method local binary sparse convolution, to differ it from the dense local binary convolution in the original paper. We use the same number of channels (256) inside the internal layer as in [18] for AlexNet. The accuracy plots in Fig. 3.3 demonstrate that this technique facilitates the recovering of accuracy. Fig. 3.4 compares the effect of this technique on

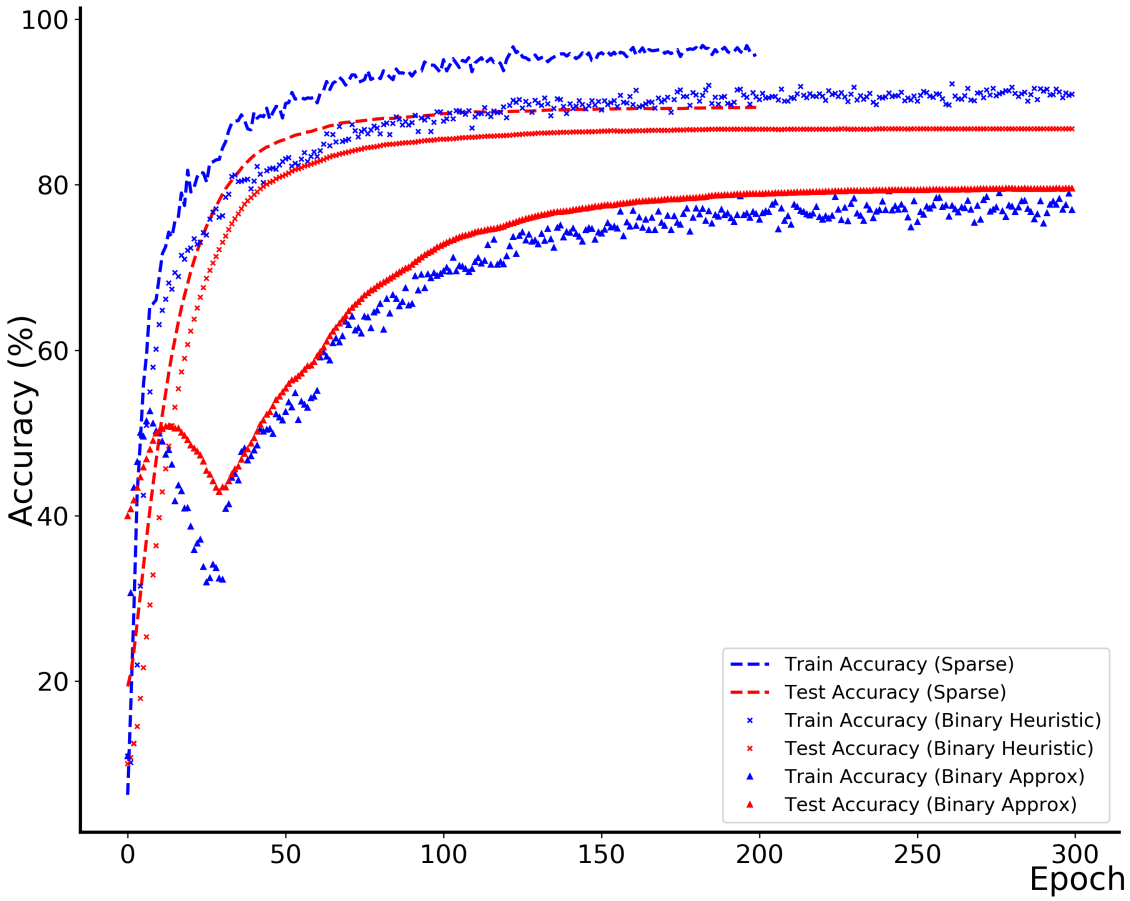
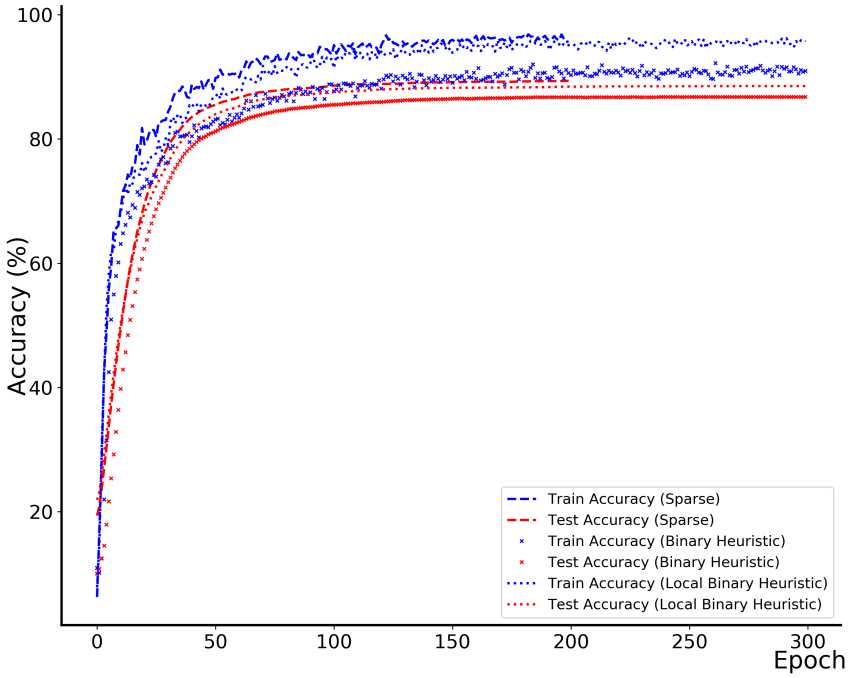


Figure 3.2: Accuracy degradation of binary sparse AlexNet on CIFAR10.

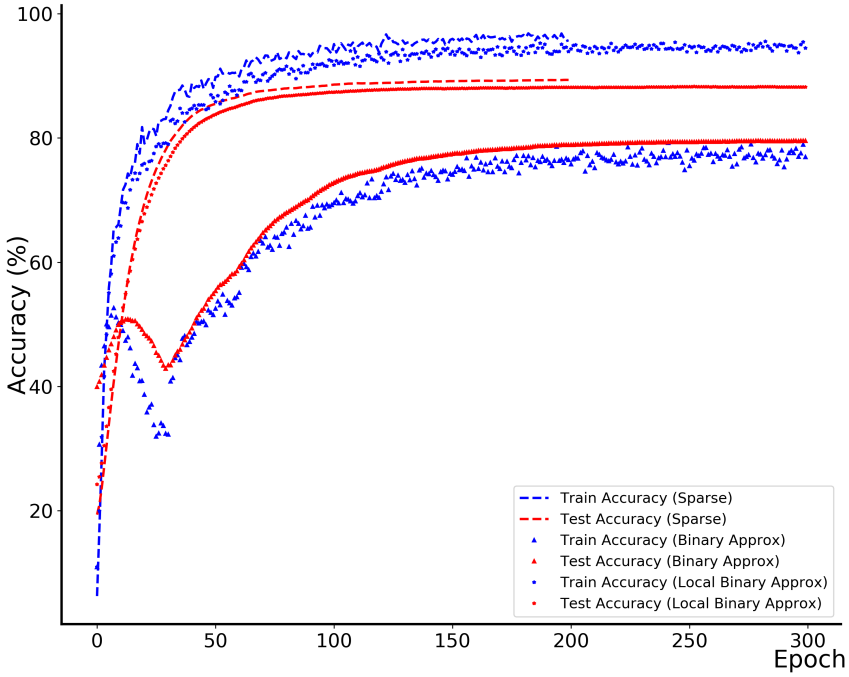
binary sparse models using heuristic and approximation methods, where we find the approximation method becomes useful, and the two methods show little difference.

3.5 Channel-wise Sparse CNN

Insertion of wide internal layers is beneficial for accuracy recovering, as we have already discussed. However, dense convolution layer is still involved, and we can not use previous Max-Min technique to make it sparse. Actually, since each kernel have size 1×1, kernel-wise Max-Min sparsification is inapplicable. On the other hand, for accuracy recovering, we employed a large number of redundant filters. Therefore, to reduce the computation complexity further, we can make the convolution layer sparse along the input channel dimension, using similar method to train sparse model as we discussed in previous chapter. This is called channel-wise Max-Min technique, as il-



(a) Accuracy of local binary sparse AlexNet on CIFAR10 using heuristic method.



(b) Accuracy of local binary sparse AlexNet on CIFAR10 using approximation method.

Figure 3.3: Accuracy recovering by insertion of wide internal layers.

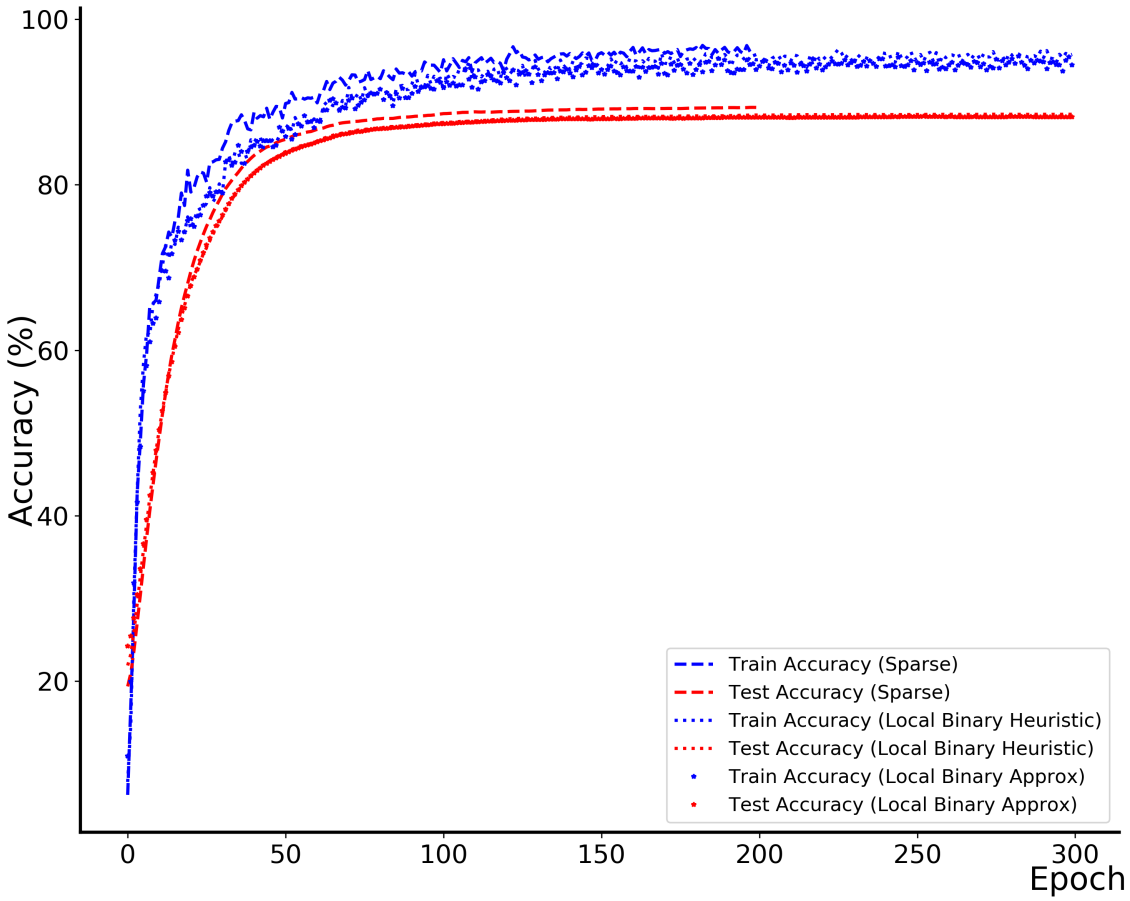


Figure 3.4: Accuracy comparison of local binary sparse AlexNet on CIFAR10 using heuristic and approximation methods.

lustrated in Fig. 3.5. The algorithm is essentially the same as Algorithm 1, except the weight tensor is made sparse along the dimension corresponding to input channel, instead of the two dimensions of kernel.

Fig. 3.6 shows the performance of the proposed channel-wise sparse local binary sparse AlexNet on CIFAR10. As can be seen, using a internal layer of size 256 as dense version will make the performance reduce. However, as we only use two instead of 256 elements, we can increase it to larger value (2048 as an example), without introducing significant computation. The analysis and comparison of computation complexity for different sparse and binary sparse version will be shown in the next section. Table 3.1 summarizes the performance of different techniques mentioned before.

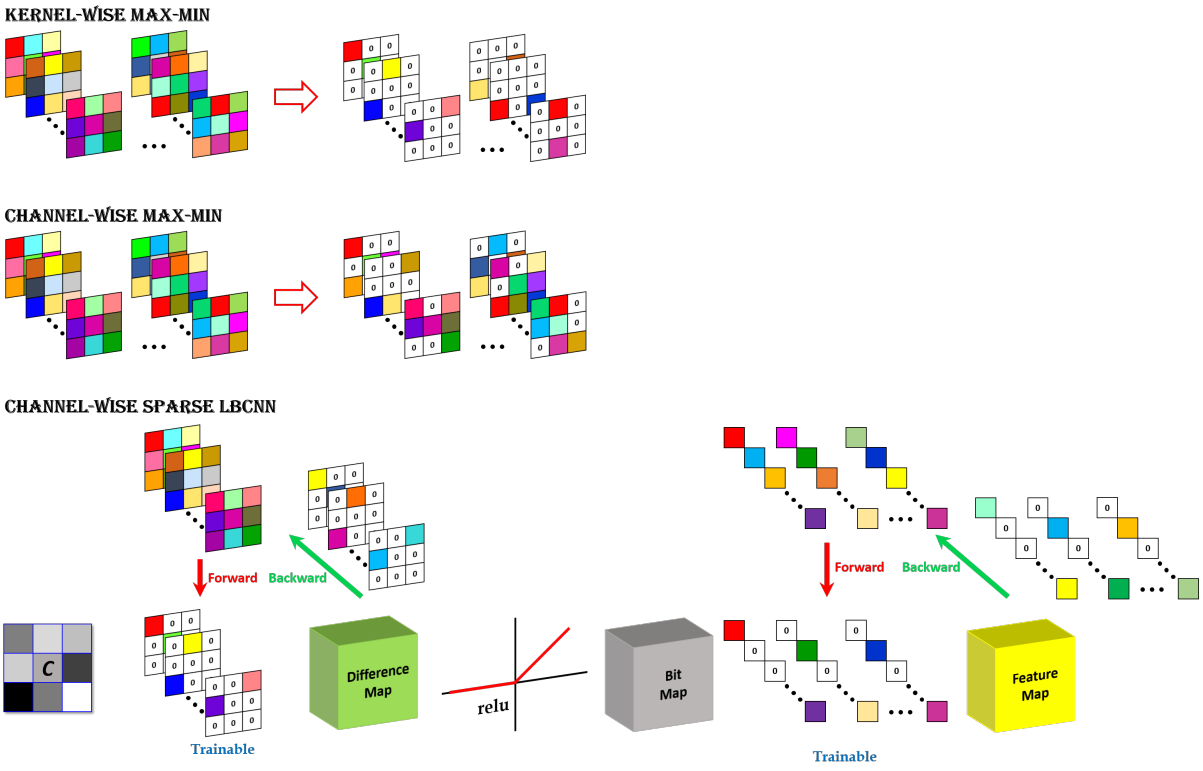


Figure 3.5: Kernel-wise vs Channel-wise Max-Min and Channel-wise LBCNN.

Model Name	Model Type	Test Accuracy (%)
AlexNet	dense	89.93
	sparse	89.54
	binary sparse (heuristic)	86.83
	binary sparse (approximation)	79.69
	local binary sparse (heuristic)	88.61
	local binary sparse (approximation)	88.35
	channel sparse local binary sparse (256)	87.14
	channel sparse local binary sparse (512)	87.55
	channel sparse local binary sparse (1024)	88.22
	channel sparse local binary sparse (2048)	88.60

Table 3.1: Comparison of performance on CIFAR10.

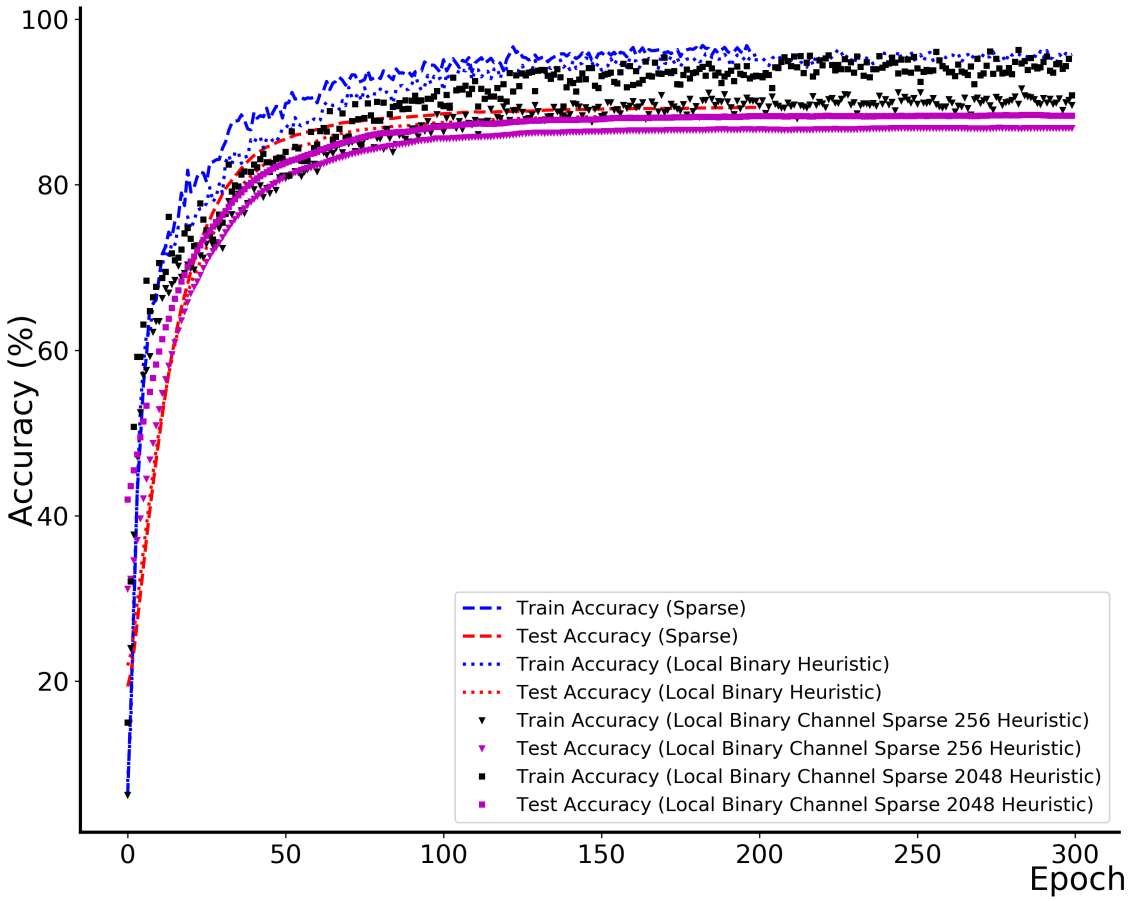


Figure 3.6: Accuracy comparison of different local binary sparse AlexNet on CIFAR10 using heuristic method.

3.6 Computation Complexity Analysis

Computation complexity of local binary CNNs is dominated by the full-precision multiplications in the 1×1 convolution layers. As summarized in Table 3.2, making the channel sparse reduces computation complexity considerably. Increasing the width of the added internal layers has little impact. Although increasing the channel number to 2048 will inflate the number of additions in binary convolution layers by 8, the final number of parameters is still the same as dense local binary convolution networks proposed in [18] because we are using binary sparse convolution here. On the other hand, the number of multiplications (FLOPs in 3.2) shrinks by 3 with channel-wise sparsification, from 183M in local binary CNNs (both dense and sparse) to around

65M in channel sparse local binary CNNs.

Model Name	Model Type	# params	FLOPs
AlexNet	dense	61.1M	715M
	sparse	59.1M	163M
	binary sparse	58.9M	114M
	local binary (256)	58.9M	183M
	channel sparse local binary sparse (256)	58.6M	60.7M
	channel sparse local binary sparse (512)	58.6M	61.3M
	channel sparse local binary sparse (1024)	58.6M	62.5M
	channel sparse local binary sparse (2048)	58.6M	64.8M

Table 3.2: Comparison of model size and computation complexity.

3.7 Summary

In this chapter, we extend the Max-Min technique for training sparse models to deriving binary sparse models. Two algorithms are introduced, namely heuristic and approximation methods, together with corresponding initialization schemes. Experiments show that binarization has more harmful impact on the model prediction performance. A wider internal layer is inserted to alleviate the degradation of prediction accuracy. In addition, channel-wise Max-Min sparsification technique is proposed to further reduce the computation complexity without adverse effect on prediction performance. Experiments with AlexNet on CIFAR10 verifies the effectiveness of the proposed techniques.

4. SUMMARY AND FUTURE WORK

In this thesis, we propose two model compression techniques with training algorithms to reduce model size and computation complexity of convolution neural networks. The proposed Max-Min technique is an effective method to train sparse models with fixed convolution kernel structures. The accuracy of sparse models derived by the Max-Min training technique is comparable with that of corresponding dense deep models. Applied to binarization, on the other hand, requires inserting more internal filters to maintain the prediction performance after model reduction. The channel-wise Max-Min sparsification is proposed to further reduce the computation complexity, combined with binarization. Two methods are introduced to determine binary weights, namely heuristic and approximation methods, which show similar performance with more internal filters inserted. Detailed algorithms and specific training techniques, together with initialization schemes are illustrated. Computation complexity of dense, sparse and binary sparse models are analyzed and compared.

Applying these techniques to various deep models, experiments on different datasets, including MNIST and CIFAR10, show the effectiveness of both methods. Comparing Max-Min sparsification and binarization, the performance degradation from binarization that we empirically observe in our experiments indicates that binarization is more harmful than sparsification regarding prediction accuracy.

There are several interesting problems that remain unsolved, from both software and hardware aspects. More effort can be made on these problems in the future. These include the following.

1) We only talk about sparsification and binarizaiton of convolution layers. Although computation complexity is mainly determined by convolution layers, model size might be dominated by fully-connected layers. Applying the proposed Max-Min technique on such layers will also be beneficial, especially for those models where the model size is dominated by the last fully-connected layers, such as AlexNet and VGGNets. One candidate method is to apply it along the input feature dimension.

2) After training, we only need the nonzero parameters to shrink the model size. However, if we want to use the trained sparse models for transfer learning, we should start from a dense model to do Max-Min training on new data. It might be better for this dense model to be as close to the last dense model during training as possible. Improper initialization might make the corresponding transfer learning slow and incorrect. Therefore, correct initial model restoring for transfer learning is an important issue to tackle.

3) During the training phase, we only use two parameters for forward and backward propagation, but we need to store all parameters during parameter updating. To reduce the memory requirement and thus improve training speed, hardware implementation should be designed specifically. As an example, we may store the redundant parameters in the disk, and read and write them only when we need to update parameters. This will potentially make the I/O speed for reading and writing to become the bottleneck of deep network training.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*, pp. 630–645, Springer, 2016.

[6] B. Graham, "Spatially-sparse convolutional neural networks," *arXiv preprint arXiv:1409.6070*, 2014.

[7] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 806–814, 2015.

[8] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

[9] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," *arXiv preprint arXiv:1706.01307*, 2017.

[10] B. Graham, M. Engelcke, and L. van der Maaten, "3d semantic segmentation with submanifold sparse convolutional networks," *arXiv preprint arXiv:1711.10275*, 2017.

[11] H. Kim, J. Yoon, B. Jeong, and S. Lee, "Rank-1 convolutional neural network," *arXiv preprint arXiv:1808.04303*, 2018.

[12] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, pp. 3123–3131, 2015.

[13] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[14] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[15] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations.," *Journal of Machine Learning Research*, vol. 18, pp. 187–1, 2017.

[17] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?," in *AAAI*, pp. 2625–2631, 2017.

[18] F. Juefei-Xu, V. N. Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, vol. 1, IEEE, 2017.

[19] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, pp. 345–353, 2017.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," tech. rep., Citeseer, 2009.

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[23] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[25] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[26] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein, "Deep information propagation," *arXiv preprint arXiv:1611.01232*, 2016.

[27] G. Yang and S. Schoenholz, "Mean field residual networks: On the edge of chaos," in *Advances in neural information processing systems*, pp. 7103–7114, 2017.

[28] J. Pennington, S. Schoenholz, and S. Ganguli, "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice," in *Advances in neural information processing systems*, pp. 4785–4795, 2017.

[29] S. Hayou, A. Doucet, and J. Rousseau, "On the selection of initialization and activation function for deep neural networks," *arXiv preprint arXiv:1805.08266*, 2018.

[30] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. S. Schoenholz, and J. Pennington, "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks," *arXiv preprint arXiv:1806.05393*, 2018.