COMPRESSING DEEP NEURAL NETWORKS USING TOEPLITZ MATRIX: ALGORITHM DESIGN AND FPGA IMPLEMENTATION

Siyu Liao[†] *Ashkan Samiee*[⋆] *Chunhua Deng*[†] *Yu Bai*[⋆] *Bo Yuan* [†]

[†] Department of Electrical & Computer Engineering, Rutgers University
[⋆] College of Engineering and Computer Science, California State University, Fullerton

ABSTRACT

Deep neural networks (DNNs) have emerged as an important artificial intelligence technique. However, the computation-intensive and storage-intensive DNNs pose severe challenges on efficient execution over the underlying hardware platform. In this paper we propose to impose Toeplitz structure on DNN models to achieve high compression ratio with negligible performance loss. Accordingly, the hardware performance can be significantly improved after performing model compression. We evaluate the proposed approach on speech recognition and implement the corresponding compressed model on FPGA. Experimental results show that our approach enables high hardware performance while retaining high task performance.

***Index Terms*—** Toeplitz matrix, DNN Compression, FPGA

1. INTRODUCTION

Deep neural networks (DNNs) have gained great success in various applications such as object detection [1], machine translation [2], speculative execution [3] and etc. Based on their strong feature extraction and representation capability, DNNs can achieve very high accuracy, and achieve even beyond human-level performance in certain tasks [4].

The current unprecedented success of DNNs highly relies on the adoption of large-size network models. Both theoretical analysis [5] and experimental results [6][7] show that increasing the depth or width of networks can significantly improve the learning capability of models. Motivated by this discovery, nowadays both academia and industry are continuing to propose deeper or wider DNNs to pursue even better task performance.

However, simply scaling up DNNs is not free. Because DNNs are computation intensive and storage intensive, the large model sizes directly cause huge computational and memory cost, thereby imposing severe challenge on the performance and efficiency of the underlying hardware platform that executes DNNs. In particular, such challenge is very realistic for embedded platforms since these types of hardware are resource constraint and energy constraint.

To address this challenge, many efforts have been proposed on both algorithm and hardware communities. Among numerous strategies, *model compression* is viewed as an important technique since it can efficiently reduce model size with negligible performance loss, thereby improving the hardware performance for network execution while retaining high task performance. The essence of network compression is based on the observation that many well-trained DNN models contain redundancy: for instance, part of neurons or connections of networks can be removed without affecting accuracy [8]. Encouraged by this phenomenon, researchers are actively exploring efficient network compression methods at different levels. To date, various types of compression approaches, ranging from pruning [9], regularization [10], network decomposition [11], low bit-width quantization [12], have been proposed in numerous works.

In this paper, we propose a new approach to perform model compression on DNNs. By utilizing the elegant mathematical property of Toeplitz matrix [13][14], we propose to impose Toeplitz structure on the topology of DNN models, thereby leading to simultaneous reduction in both computational and storage requirement. To accommodate this structure-imposing approach, Toeplitz matrix-based forward and backward propagation schemes are developed. To validate the proposed method, we evaluate its task performance on long short-term memory (LSTM) for speech recognition. Experimental results show that our approach can enable 28.7 times reduction in LSTM model size with negligible task performance loss. Further, we develop hardware architecture of block Toeplitz matrix-based long short-term memory (LSTM) and implement it on FPGA board. With 200MHz clock frequency and 20W power consumption, the FPGA design takes 7.35 μs for processing one data, which corresponds to over 130000 frame/s throughput.

2. PRELIMINARIES

2.1. Deep Neural Network

The pipeline of DNNs typically contains DNN training and inference. During the training phase, DNNs learn from input data by optimizing an objective function. For instance, in the image classification task the objective function of DNNs is usually minimizing the cross-entropy loss [6]. To achieve that, gradient-based optimizer is typically used to perform optimization process. After being well trained in training dataset, in the inference phase DNN models receive the test data and perform classification or regression according to different applications.

Specifically, both training and inference procedure require the computation flow called *forward propagation*. In general, for a fully connected layer in the DNN, forward propagation can be described as follows:

$$\mathbf{y} = f(\mathbf{a}) = f(\mathbf{W}\mathbf{x} + \mathbf{b}), \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^m$, $\mathbf{W} \in \mathbb{R}^{m\times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ are the output, weight matrix, input of the current layer and bias, respectively. Also, $\mathbf{a}$ is the inner product of $\mathbf{W}$ and $\mathbf{x}$ and $f(\cdot)$ is an activation function such as sigmoid.

Besides forward propagation, DNN training also requires *backward propagation* to update model weights after each iteration. The general principle of such update is as follows:

$$\mathbf{W} \longleftarrow \mathbf{W} - \alpha\frac{\partial L}{\partial \mathbf{W}}, \tag{2}$$

where L is the objective function to optimize, $\frac{\partial L}{\partial \mathbf{W}}$ is the gradient of objective function with respect to the weight matrix and α is usually a small constant number.

2.2. Toeplitz Matrix

In this paper the proposed approach is to utilize Toeplitz matrix to represent weight matrices in DNN models. Mathematically, a Toeplitz matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ can be defined by a vector $\mathbf{w} = (w_{1-n}, w_{2-n}, \ldots, w_0, \ldots, w_{n-1})$, where w_i is a scalar for $1-n \le i \le n-1$:

$$\mathbf{W} = \begin{bmatrix} w_0 & w_{-1} & \cdots & w_{1-n} \\ w_1 & w_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & w_{-1} \\ w_{n-1} & \cdots & w_1 & w_0 \end{bmatrix}. \tag{3}$$

It should be noted that there are $2n-1$ parameters when defining a square Toeplitz matrix; while the conventional unstructured matrix with the same size contains n^2 parameters.

3. PROPOSED APPROACH

3.1. Impose Toeplitz Structured on DNNs

Section 2.2 shows that the Toeplitz matrix has much lower space complexity ($O(n)$) than conventional matrix ($O(n^2)$). Encouraged by this characteristics, we propose to impose Toeplitz structure on the construction of DNN models. In other words, the weight matrices of layers of DNNs are now enforced to be Toeplitz matrices. To accommodate this change, both the forward propagation and backward propagation schemes need to be reformulated as follows.

Forward propagation: To perform Toeplitz matrix-based forward propagation, a straightforward method is to simply replace $\mathbf{W}$ in Eqn. 1 with Toeplitz format and conduct matrix-vector multiplication. Although this simple change works, it is not optimal in efficiency. This is because as a type of structured matrix, Toeplitz matrix is inherently affiliated with fast matrix-vector multiplication. Specifically, as pointed out in [14], the multiplication between Toeplitz matrix and vector can be performed using Fast Fourier Transform (FFT) and its inverse (IFFT) as follows:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \text{IFFT}(\text{FFT}(\mathbf{w}') \circ \text{FFT}(\mathbf{x}'))_{1:n}, \tag{4}$$

where $\mathbf{w}' = (w_0, \ldots, w_{1-n}, 0, w_{n-1}, \ldots, w_1) \in \mathbb{R}^{2n}$, $\mathbf{x}' = (\mathbf{x}, 0, \ldots, 0) \in \mathbb{R}^{2n}$, $\mathbf{x} \in \mathbb{R}^n$ is the original input, and $\circ$ means the element-wise product. The subscript $1:n$ means that we take the first n elements from IFFT result as vector $\mathbf{a} \in \mathbb{R}^n$. Notice that because the computational complexity of FFT/IFFT is $O(n \log n)$, the overall computational complexity is reduced from $O(n^2)$ to $O(n \log n)$ and the space complexity is reduced from $O(n^2)$ to $O(n)$ too. This means that imposing Toeplitz structure on DNN models can save the storage and accelerate the execution simultaneously.

Backward propagation: The essence of backward propagation is to calculate gradients. Similar to the procedure proposed in [15], we derive the gradient computation given objective function L with respect to $\mathbf{w}'$ as follows:

$$\frac{\partial L}{\partial \mathbf{w}'} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'}) \circ \text{FFT}(\mathbf{x}'')), \tag{5}$$

where $\mathbf{x}'' = (x_1, 0, \ldots, 0, x_n, \ldots, x_2) \in \mathbb{R}^{2n}$, and $\frac{\partial L}{\partial \mathbf{a}'} = (\frac{\partial L}{\partial \mathbf{a}}, 0, \ldots, 0) \in \mathbb{R}^{2n}$. Moreover, the gradients of input $\mathbf{x}$ can be also calculated as:

$$\frac{\partial L}{\partial \mathbf{x}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'}) \circ \text{FFT}(\mathbf{w}''))_{1:n}, \tag{6}$$

where $\mathbf{w}'' = (w_0, w_1, \ldots, w_{n-1}, 0, w_{1-n}, \ldots, w_{-1}) \in \mathbb{R}^{2n}$, and the first n elements of IFFT result will be vector $\frac{\partial L}{\partial \mathbf{x}}$.

3.2. Impose Block-Toeplitz Structure on DNNs

From the perspective of deployment, simply imposing square Toeplitz structure on DNN models is challenging. This is because using Toeplitz matrix renders a fixed compression ratio while in practice it always requires flexibility for compression effect. To address this challenge, we propose to impose block-Toeplitz structure on the construction of DNNs. In general, a block-Toeplitz matrix consists of multiple square Toeplitz matrices, and it can fit weight matrices in any shape [1]. Accordingly, the forward and backward propagation schemes need to be re-investigated in more general scenarios.

Forward propagation: Let $\mathbf{W} \in \mathbb{R}^{m\times n}$ be the weight matrix, which is divided into multiple square blocks in size $b \times b$. There are $m/b \times n/b$ blocks and each is a Toeplitz matrix $\mathbf{w}_{ij}$

[1]Zero-padding may be required If one dimension of matrix is not the multiple of another one. This will not cause storage or computation overhead.

that can be defined using $2b-1$ weight parameters, where $i\in\{1,\ldots,m/b\}$, $j\in\{1,\ldots,n/b\}$. Similarly we can divide the input vector $\mathbf{x}$ into different $\mathbf{x}_j\in\mathbb{R}^b$, and output vector $\mathbf{a}$ into different $\mathbf{a}_i\in\mathbb{R}^b$. Then the product of matrix and vector can be re-written as follows:

$$\mathbf{a}=\mathbf{W}\mathbf{x}=\begin{bmatrix}\mathbf{a}_1\\ \vdots\\ \mathbf{a}_{m/b}\end{bmatrix}=\begin{bmatrix}\sum_{j=1}^{n/b}\mathbf{w}_{1j}\mathbf{x}_j\\ \vdots\\ \sum_{j=1}^{n/b}\mathbf{w}_{m/b,j}\mathbf{x}_j\end{bmatrix}, \tag{7}$$

where the calculation of $\mathbf{a}_i$ can be accelerated with FFT/IFFT as in Eqn. 4. Algorithm 1 summarizes the scheme of block-Toeplitz matrix-based forward propagation.

Backward propagation: In the scenario of using block Toeplitz matrix, similar to the derivation in Section 3.1, the corresponding calculation of gradient descent can also be written as:

$$\frac{\partial L}{\partial \mathbf{w}'_{ij}}=\text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'_i})\circ\text{FFT}(\mathbf{x}''_j)), \tag{8}$$

$$\frac{\partial L}{\partial \mathbf{x}_j}=\sum_{i=1}^{m/b}\text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'_i}\circ\text{FFT}(\mathbf{w}''_{ij}))_{1:b}, \tag{9}$$

where $\mathbf{a}'_i$, $\mathbf{x}''_j$ and $\mathbf{w}'_{ij}$ are defined similarly as in Eqn. 5, and $\mathbf{w}''_{ij}$ is similar to $\mathbf{w}''$ as in Eqn. 6, respectively. Algorithm 2 summarizes the scheme of block-Toeplitz matrix-based backward propagation.

Notice that similar to the case of Toeplitz matrix, block-Toeplitz matrix also achieves simultaneous reduction in space and computational complexity. Specifically, the space complexity is reduced from $O(n^2)$ to $O(n^2/b)$ and computational complexity is reduced from $O(n^2)$ to $O(\frac{n^2}{b}\log b)$. Hence such reduction can be precisely controlled by adjusting the block size b.

Algorithm 1: Block-Toeplitz Matrix-based Forward Propagation

```
Input: $\mathbf{w}'_{11},...,\mathbf{w}'_{m/b,j},\mathbf{x},b$
Output: $\mathbf{a}$
Partition $\mathbf{x}\in\mathbb{R}^n$ into $n/b$ vectors, $\mathbf{x}_1,\ldots,\mathbf{x}_{n/b}$;
for $i\leftarrow 1$ until $m/b$ do
    $\mathbf{a}_i\leftarrow 0$;
    for $j\leftarrow$ until $n/b$ do
        $\mathbf{a}_i\leftarrow\mathbf{a}_i+\text{IFFT}(\text{FFT}(\mathbf{w}'_{ij})\circ\text{FFT}(\mathbf{x}_j))_{1:b}$;
    end
end
return $\mathbf{a}$;
```

3.3. Empirical Experiments and Compression Ratio

To evaluate the proposed approach, we perform experiment on long short-term memory (LSTM) for a speech recognition task. LSTM is a type of widely used DNN for various sequence-involved tasks. In general, a LSTM takes a sequence of input

Algorithm 2: Block-Toeplitz Matrix-based Backward Propagation

```
Input: $\frac{L}{\mathbf{a}'_1},\ldots,\frac{L}{\mathbf{a}'_{m/b}},\mathbf{x}''_1,\ldots,\mathbf{x}''_{n/b},b$
Output: $\frac{L}{\mathbf{x}},\frac{L}{\mathbf{w}''_{11}},\ldots,\frac{L}{\mathbf{w}''_{m/b,n/b}}$
for $j\leftarrow 1$ until $n/b$ do
    $\frac{L}{\mathbf{x}_j}\leftarrow 0$;
    for $i\leftarrow$ until $m/b$ do
        $\frac{L}{\mathbf{w}'_{ij}}\leftarrow\text{IFFT}(\text{FFT}(\frac{L}{\mathbf{a}'_i})\circ\text{FFT}(\mathbf{x}''_j))$;
        $\frac{L}{\mathbf{x}_j}\leftarrow\frac{L}{\mathbf{x}_j}+\text{IFFT}(\text{FFT}(\frac{L}{\mathbf{a}'_i})\circ\text{FFT}(\mathbf{w}''_{ij}))_{1:b}$;
    end
end
return $\frac{L}{\mathbf{x}},\frac{L}{\mathbf{w}''_{11}},\ldots,\frac{L}{\mathbf{w}''_{m/b,n/b}}$;
```

Table 1: Task performance with different compression ratio

Model	Block Size	WER	Overall Comp. Ratio
Uncompressed	-	18.08 [16]	1.00
Compressed-1	32	17.02	15.34
Compressed-2	64	18.12	28.76
Compressed-3	128	21.20	51.56

$\mathbf{x}_1,\ldots,\mathbf{x}_T$ and generates a sequence of output $\mathbf{y}_1,\ldots,\mathbf{y}_T$, where T is number of time steps. The computation in LSTM can be formulated for each time step $1\le t\le T$ as follows:

$$\begin{aligned}\mathbf{i}_t&=\sigma(\mathbf{W}_{ix}\mathbf{x}_t+\mathbf{U}_{ir}\mathbf{y}_{t-1}+\mathbf{b}_i)\\ \mathbf{f}_t&=\sigma(\mathbf{W}_{fx}\mathbf{x}_t+\mathbf{U}_{fr}\mathbf{y}_{t-1}+\mathbf{b}_f)\\ \mathbf{g}_t&=h(\mathbf{W}_{gx}\mathbf{x}_t+\mathbf{U}_{gr}\mathbf{y}_{t-1}+\mathbf{b}_g)\\ \mathbf{c}_t&=\mathbf{f}_t\circ\mathbf{c}_{t-1}+\mathbf{g}_t\circ\mathbf{i}_t\\ \mathbf{o}_t&=\sigma(\mathbf{W}_{ox}\mathbf{x}_t+\mathbf{U}_{or}\mathbf{y}_{t-1}+\mathbf{b}_o)\\ \mathbf{y}_t&=\mathbf{o}_t\circ h(\mathbf{c}_t)\end{aligned}, \tag{10}$$

where $\mathbf{W}$ and $\mathbf{U}$ indicate weight matrices and $\mathbf{b}$ indicates bias parameters for each gate in LSTM, respectively. The $\circ$ is the element-wise product. $\sigma(\cdot)$ and $h(\cdot)$ are sigmoid and hypertangent functions, respectively.

From the above equation it is seen that LSTM consists of multiple weight matrices. Therefore, we impose block-Toeplitz structure on all weight matrices to compress model size. Specifically, the compression is performed on the model structure in the LSTM layers of model in [16]. The dataset is the AN4 audio data for speech recognition, where training utterances are 948, testing utterances are 130, and there are in total 29 unique spoken characters. During train process, we train 150 epochs using batch size 32, learning rate 0.0003 with annealing rate 1.01. The task performance is measured in word error rate (WER).

Table 1 summarizes the test results with different compression ratio. It is seen that the proposed approach leads to high compression ratio with negligible performance loss. Notice that here the block size does not equal to overall compression ratio. This is because after high compression on weight matrix the uncompressed bias vectors dominates the model size.

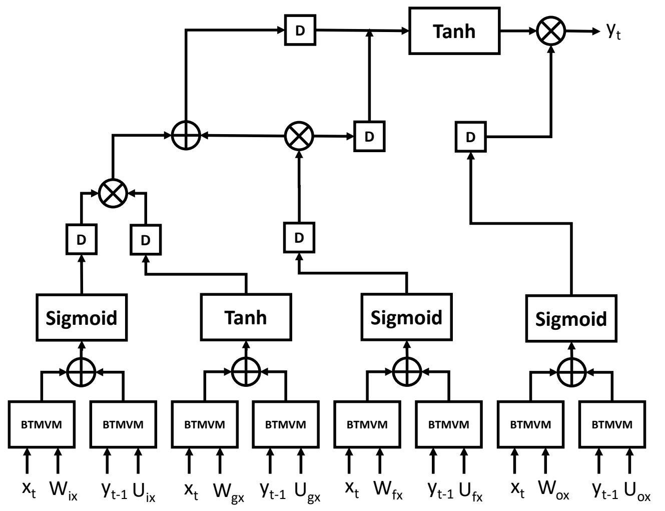
D
Tanh
y_t
D
D
D
D
D
D
Sigmoid
Tanh
Sigmoid
Sigmoid
BTMVM
x_t W_{ix} y_{t-1} U_{ix} x_t W_{gx} y_{t-1} U_{gx} x_t W_{fx} y_{t-1} U_{fx} x_t W_{ox} y_{t-1} U_{ox}

Fig. 1: Architecture of block-Toeplitz matrix-based LSTM. Bias vector is included in weight matrix. BTMVM represents block Toeplitz matrix vector multiplication.

4. FPGA DESIGN AND PERFORMANCE

To demonstrate the advantage of the proposed approach for hardware design, we implement the example block-Toeplitz matrix-based LSTM in Section 3 and evaluate its hardware performance for inference on FPGA. Fig. 1 shows the hardware architecture of the example LSTM. Here the key module is the block Toeplitz matrix vector multiplication (BTMVM) module to perform matrix-vector multiplication, which is conducted using FFT/IFFT as described in Algorithm 1. Notice that according to the characteristics of forward propagation and FFT/IFFT, three optimization approaches can be used to improve the hardware performance:

1) Due to the linearity of IFFT, the IFFT operation can be done after summing over all the FFT results, thereby number of IFFT operations reduced greatly;

2) Since weight matrices have been determined after training, their FFT transformation can be pre-calculated and saved in BRAMs buffers of FPGA to reduce computational cost;

3) Considering real-input FFT has conjugate symmetric property, we only need to store half of the pre-calculated frequency domain weight matrices, thereby further saving memory cost.

Experiment Setup: The FPGA platform used in the experiment is Alpha Datas ADM-7V3 board consisting off a Xilinx Virtex-7 (XC7VX690T-2) FPGA chip and a 16 GB DDR3 memory. The FPGA is plugged into the PCI-e 3.0 X8 of= host motherboard that include an Intel Core i7-8700 processor. We develop the C++ model of LSTM and use Xilinx Vivado HLS 2017.4 to generate RTL model. After that, the FPGA implementation is done by connecting the LSTM IP core to the PCIe IP core in the FPGA to handle the DMA communications.

Design Consideration: For the target LSTM model, its component weight matrices **W** and **U** are of 512×512 size. According to Table 1, We choose the block size as 64 to achieve good balance between compression ratio and task performance. Therefore, each weight matrix consists of 64 smaller 64×64 Toeplitz matrices. Regarding FFT, notice that Eqn. 4 shows that the number of FFT points for fast operation on Toeplitz matrix is the double of matrix dimension. Therefore, 128-point FFT and IFFT is used for Toeplitz structure calculation. In our FPGA implementation, such fixed-point 128-point FFT is implemented by Xilinx FFT IP core that requires 12 DSP48E1 slices, one 32k BRAM and consumes 5 clock cycles. Also notice that due to the BRAM budget, unlike the suggestion in the first paragraph of this section, in this design we do not pre-calculate FFT of weight matrices or save them in BRAM since BRAM resource is limited. Instead, those FFT calculations are performed online during inference time. We suggest that such pre-calculation can be performed when the underlying FPGA board has abundant BRAM resource.

Hardware Performance: Table 2 summarizes the hardware performance of the example LSTM FPGA design. Operated on 200MHz, the FPGA design consumes 1470 cycles to process one input data, which corresponds to 7.35 μs latency. Therefore, the throughput of our design is over 130000 frame/s. Meanwhile, the overall power consumption of the entire FPGA board is 30W.

Table 2: Hardware performance of FPGA design.

FPGA board	FPGA:Virtex-7(690t)(28nm)
Quantization scheme	16 bits
Clock frequency	200MHz
Number of clock cycles	1470
DSP resource usage	3,600 (72%)
BRAM resource usage	1470 (87%)
LUT resource usage	859,200 (67%)
FF resource usage	429,600 (70%)
Latency for data processing	7.35 μs per frame
Power consumption	30 Watt

5. ACKNOWLEDGEMENT

This work is supported by the National Science Foundation Awards CCF-1854737 and CCF-1854742.

6. CONCLUSION

In this paper we propose to impose Toeplitz structure on the DNN models. By simultaneously reduce storage cost and accelerate computation procedure, the proposed approach can enable high hardware performance of DNN accelerator with retaining high task performance.

7. REFERENCES

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[3] Maotong Xu, Sultan Alamro, Tian Lan, and Suresh Subramaniam, "Laser: A deep learning approach for speculative execution and replication of deadline-critical jobs in cloud," in *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 2017, pp. 1–8.

[4] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] Gaurav Pandey and Ambedkar Dukkipati, "To go deep or wide in learning?," *arXiv preprint arXiv:1402.5634*, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] Sergey Zagoruyko and Nikos Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[8] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al., "Predicting parameters in deep learning," in *Advances in neural information processing systems*, 2013, pp. 2148–2156.

[9] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[10] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2074–2082.

[11] Jian Xue, Jinyu Li, and Yifan Gong, "Restructuring of deep neural network acoustic models with singular value decomposition.," in *Interspeech*, 2013, pp. 2365–2369.

[12] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[13] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, Victor Pan, and Bo Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," *arXiv preprint arXiv:1703.00144*, 2017.

[14] Victor Y Pan, *Structured matrices and polynomials: unified superfast algorithms*, Springer Science & Business Media, 2012.

[15] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2857–2865.

[16] Zhisheng Wang, Jun Lin, and Zhongfeng Wang, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2763–2775, 2017.