FPGA Implementation of an Image Classifier Using Pipelined FFT Architecture

Shafiqul Hai, *Member, IEEE*, and Tella Rajashekhar Reddy, *Student Member, IEEE*

***Abstract*—Deep neural network (DNN) belongs to an important class of machine learning algorithms generally used to classify digital data in the form of image and speech recognition. The computational complexity of a DNN-based image classifier is higher than traditional fully connected (FC) feed-forward NNs. Therefore, dedicated cloud servers and graphical processor units (GPUs) are utilized to achieve high-speed and large-capacity computation tasks in machine vision systems. However, a growing demand exists for real-time processing of complex machine-learning tasks on embedded systems. As FC layers consume the highest fraction of computational power and memory footprint, innovating novel power-efficient and low-footprint NN architecture for embedded systems is crucial. In this letter, a pipelined and parallel fast Fourier transform (FFT)-based FC-DNN architecture is implemented on Stratix-10 FPGA using VHDL. The footprint of the DNN is further reduced using a folded FFT network. The proposed algorithm is tested using two benchmark training set examples, the MNIST database of handwritten digits and the CIFAR-10 database. In both cases, we achieve > 90% accuracy, while the power consumption of the 2-parallel folded FFT-based network is around 45% less than the traditional series FFT-based architectures.**

***Index Terms*—Block circulant matrix (BCM), deep neural network (DNN), machine vision, pipelined and folded fast Fourier transform (FFT).**

I. Introduction

DEEP convolutional neural networks (CNNs) facilitate the extraction of complex high-level features of an object or image, which results in higher accuracy in prediction [1], [2]. Deep neural network (DNN) applications are built with the target to address computational complexity, processing speed, power, and memory bandwidth for software and hardware implementation.

DNNs for image classifiers include convolutional layers, max-pooling layers, and finally, the fully connected (FC) layers [3]. In an embedded system, the FC layers become a bottleneck to achieve power-efficient operation with high computation speed due to the large size of the weight matrix and input vector created from the extracted image features. Therefore, it is essential to explore new architectures which utilize the parameter redundancy techniques to reduce the size of the matrix-vector multiplication or simplify the multiplier–accumulator (MAC) layers in the FPGA, microcontroller, or ASIC-based accelerators. It is shown in [4], that fast Fourier transform (FFT)-based convolutions and overlap-add operations in the CNN layer improve the execution time by 3.36 times compared to direct convolution on the ARM A53 CPU. A block circulant matrix (BCM) vector multiplication via FFT operations is implemented in the FC layers of an optical accelerator [5], which saves 2.2 to 3.7 times area cost compared to singular value decomposition (SVD)-based architecture. The BCM-based FFT architecture was first demonstrated in [6], where the network is trained such that the weight parameters form a BCM. The BCM facilitates circular convolution or elementwise FFT multiplication between the input vector and weight matrix. The circular projection-based FFT operation improves power efficiency of an NN by reducing the space and computation complexity from $\mathcal{O}(k^2)$ to $\mathcal{O}(k)$ and time complexity from $\mathcal{O}(k^2)$ to $\mathcal{O}(k \log k)$. However, to the best of our knowledge, an FFT-based DNN has not yet been investigated with the target to reduce the power consumption and footprint by implementing pipelined, parallel, and folded FFT architecture. Therefore, to further improve the power efficiency of the NN, it is vital to investigate joint algorithms with the target to reduce the footprint of the circular projection-based FFT network.

This letter describes FPGA implementation of a novel pipelined-parallel FFT architecture for DNN by exploiting circular projection. This design is especially suitable for the power-constrained embedded systems used in autonomous vehicles, surveillance cameras, drones, and remote sensors in harsh environments. The proposed architecture reduces the number of circuit components (multiplier, adders, and comparators) on an FPGA.

II. Block Circulant Matrix Computation

A circular projection on the weight matrix is implemented in structured neural networks (SNNs) to reduce computational complexity [6]. In an SNN with n-input and m-output in any layer l, the complete weight matrix, $\boldsymbol{C}(l) \in \mathbb{R}^{m \times n}$ is partitioned into $p \times q$ numbers of submatrices, each being a $k \times k$ BCM, $\boldsymbol{W}(l)$. Therefore, the output from layer l is: $\boldsymbol{y(l)} = \boldsymbol{C(l)} \times \boldsymbol{a(l-1)} + \boldsymbol{b(l)}$, where, $\boldsymbol{a(l-1)}$ is the output from layer l-1 and $\boldsymbol{b(l)}$ is the $m \times 1$ bias vector. The output can be calculated in terms of BCM as

$$\boldsymbol{y}(l) = \begin{bmatrix} \mathbf{y}_0(l) \\ \mathbf{y}_1(l) \\ \vdots \\ \mathbf{y}_p(l) \end{bmatrix} = \begin{bmatrix} \sum_{j=0}^{q-1} \boldsymbol{W}_{0j}(l)\boldsymbol{a}_j(l-1) \\ \sum_{j=0}^{q-1} \boldsymbol{W}_{1j}(l)\boldsymbol{a}_j(l-1) \\ \vdots \\ \sum_{j=0}^{q-1} \boldsymbol{W}_{p-1j}(l)\boldsymbol{a}_j(l-1) \end{bmatrix} + \boldsymbol{b(l)}. \quad (1)$$

Received 9 September 2024; revised 3 November 2024; accepted 11 November 2024. Date of publication 18 November 2024; date of current version 13 June 2025. This work was supported by the Mathematics of Information Technology and Complex Systems (MITACS). This manuscript was recommended for publication by F. Merchant. *(Corresponding author: Shafiqul Hai.)*

The authors are with the Department of Electrical and Computer Engineering, Lakehead University, Barrie, ON L4M 3X9, Canada (e-mail: shai@lakeheadu.ca).

Digital Object Identifier 10.1109/LES.2024.3500020

In (1), each of the input-segments ($\mathbf{a}_j(l-1)$) is a $k \times 1$ vector, where, $n = qk$. The BCM ($\boldsymbol{W}(l)$) is one of the segments of the full weight matrix ($\boldsymbol{C}(l)$). A $k \times k$ BCM $\boldsymbol{W} \in \mathbb{R}^{k\times k}$ is defined by a column vector $\boldsymbol{w} = (w_0, w_1, \ldots, w_{k-1})$ which contains all independent parameters in $\boldsymbol{W}$ and the subsequent columns represent circularly shifted versions of the first column by one element, as shown in

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_0 & \boldsymbol{w}_{k-1} & \cdots & \boldsymbol{w}_1 \\ \boldsymbol{w}_1 & \boldsymbol{w}_0 & \cdots & \boldsymbol{w}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{k-1} & \boldsymbol{w}_{k-2} & \cdots & \boldsymbol{w}_0 \end{bmatrix}. \tag{2}$$

In (1), the matrix-vector multiplication can be efficiently calculated using FFT and element-wise multiplication, and therefore the forward pass computation can be performed with reduced $\mathcal{O}(k \log k)$ complexity. Next, we focus on the BCM multiplication with the input vector $\boldsymbol{a(l-1)}$ to demonstrate the algorithm for forward and backward propagation. In layer l with BCM $\boldsymbol{W}(l) \in \mathbb{R}^{k\times k}$, the forward-propagation computation can be performed using

$$\begin{aligned} \boldsymbol{a}(l) &= ReLU\big[\boldsymbol{y}(l)\big] \\ &= ReLU[\boldsymbol{W}(l) \times \boldsymbol{a}(l-1) + \boldsymbol{b}(l)]. \end{aligned} \tag{3}$$

In (3), $\boldsymbol{a(l)}$ and $\boldsymbol{a(l-1)}$ are segmented output and input vector of size $k \times 1$, respectively. We use the *ReLU* operation for the nonlinear activation function.

The matrix-vector multiplication in (3) is basically a circular convolution ($\circledast$), and can be efficiently performed by FFT ($\mathscr{F}$) element-wise multiplication ($\circ$), followed by inverse FFT ($\mathscr{F}^{-1}$) operation:

$$\begin{aligned} \boldsymbol{y}(l) &= \boldsymbol{W}(l) \times \boldsymbol{a}(l-1) + \boldsymbol{b}(l) \\ &= \boldsymbol{w}(l) \circledast \boldsymbol{a}(l-1) + \boldsymbol{b}(l) \\ &= \mathscr{F}^{-1}\big[\mathscr{F}(\boldsymbol{w}(l)) \circ \mathscr{F}(\boldsymbol{a}(l-1))\big] + \boldsymbol{b}(l). \end{aligned} \tag{4}$$

To demonstrate backpropagation, we need to update the network parameters, starting from the last layer (L) using gradient descent, where $\boldsymbol{E}$ is an error vector and α is the learning rate constant

$$\begin{aligned} \boldsymbol{w}_{ij}(L) &= \boldsymbol{w}_{ij}(L) - \alpha \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_{ij}(L)}, \text{here } i,j: 0 \text{ to } k-1 \\ &= \boldsymbol{w}_{ij}(L) - \alpha \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{a}_j(L)} \frac{\partial \boldsymbol{a}_j(L)}{\partial \boldsymbol{y}_j(L)} \frac{\partial \boldsymbol{y}_j(L)}{\partial \boldsymbol{w}_{ij}(L)} \\ &= \boldsymbol{w}_{ij}(L) - \alpha \boldsymbol{e} \circ \big[ReLU'\{\boldsymbol{y}_j(L)\} \circledast s_{\rightarrow 1}\text{flip}\{\boldsymbol{a}_j(L-1)\}\big]. \end{aligned} \tag{5}$$

In (5), $ReLU'\{\boldsymbol{y}_j(L)\} = 1$ if $\boldsymbol{y}_j(L) > 0$, otherwise the differential of the ReLU function is zero.

As mentioned before, the circular convolution operation in (5) can be achieved using FFT and inverse FFT. The weight parameters are calculated during backpropagation or training phase to form the BCM $\boldsymbol{W}(l)$ which is required in the forward propagation stage.

In both the forward and backward propagation stages the element-wise multiplication reduces the computational complexity. In addition, if a pipelined-parallel and folded FFT architecture can be designed, this will further reduce the chip footprint and increase power efficiency. We will describe this procedure in the next section.

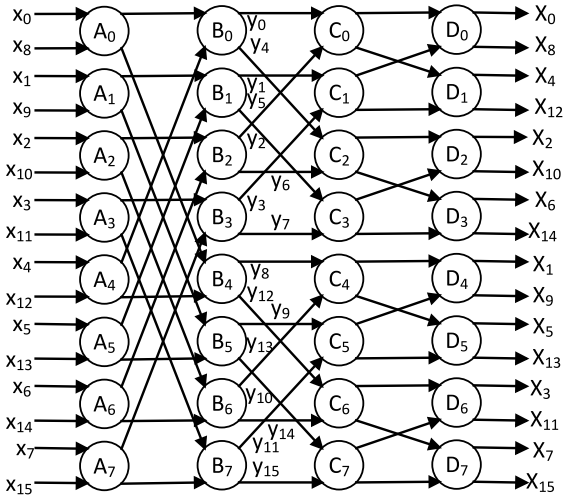

Fig. 1. DFG of a Radix-2 16-point DIF FFT before folding method.

III. Folded FFT Architecture for BCM-Based DNN

If an $m(=512) \times n(=1024)$ weight matrix $\boldsymbol{C}(l)$ of an NN is partitioned into $p\ (=32) \times\ q\ (=64)$ submatrices or BCMs, then each of the BCM, $\boldsymbol{W}(l)$, will be of dimension $k(=16) \times k(=16)$. Therefore, a 16-point FFT will be required in the forward and backward propagation stages. If a large dimension weight matrix is required in the NN, then the same design process can be extended to increase the FFT size. A 16-point FFT requires 8 butterfly units (BFUs) at each of the four stages. Fig. 1 shows the data flow graph (DFG) of a radix-2 16-point decimation in frequency (DIF) FFT. The nodes A, B, and C represent the BFU computation, where each of the BFU consists of two adders and one multiplier.

The flow graph in Fig. 1 can be pipelined or retimed, and folded to design a radix-2 multipath delay commutator (R2MDC) architecture. Pipelined and folded FFT architectures have been investigated in the past to reduce the number of BFUs [7], [8]. For example, by applying folding transformation with folding factor $= 8/2 = 4$, all BFU's on a same column can be mapped onto a single BFU. Therefore, a 2-parallel architecture can be generated using this technique. The folded 16-point FFT architecture requires 32 registers from BFU-A to BFU-B. Therefore, we apply the lifetime analysis technique, derive the register allocation table and the required control signals (s_1 to s_3), to minimize the number of registers.

The R2MDC architecture is shown in Fig. 2. The two-parallel architecture process two inputs in one clock cycle (i.e., one each from the set $y_0, y_2, y_4, y_6, y_1, y_3, y_5, y_7$ and $y_8, y_{10}, y_{12}, y_{14}, y_9, y_{11}, y_{13}$, as shown in Fig. 2). The folded circuit requires only $\log_2 N = \log_2 16 = 4$ complex BFUs, 3 complex multipliers and $3N/2 - 2 = 22$ registers (R_1 to R_{21}). The 2-parallel R2MDC architecture facilitates reducing the clock frequency (f) by half compared to traditional Radix-2 Single-path delay feedback (R2SDF) architecture [7]. Therefore, theoretically the dynamic power (CV^2f) consumption by the 16-point R2MDC is less compared to the R2SDF architecture. However, the actual power consumption can only be verified though a hardware implementation, which is described in the next section.

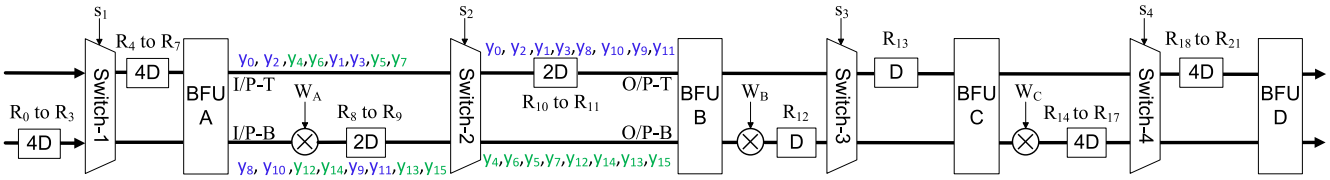

Fig. 2. Pipelined 2-parallel 16-point FFT architecture for BCM multiplication computation in the proposed image classifier.

TABLE I
COMPARISON OF INFERENCE ACCURACY

Case No.	Configuration	Accuracy
1. MINST	(28×28)-784-1024-512-10	97.55%
2. MNIST	(28×28)-784(16)-1024(16)-512(16)-10	97.07%
3. CIFAR	(32×32×3)-3072(16)-1024(16)-512(16)-10	51.4%
4. CIFAR	(32×32×3)-CNN-592-2048-1024-512-10	92.6 %
5. CIFAR	(32×32×3)-CNN-592(16)-2048(16)-1024(16)-512(16)-10	92.2%

IV. ANALYSIS AND EXPERIMENTAL RESULTS

To find the weight parameters, we train the proposed BCM-based FC-DNN, using the scripts written in MATLAB. Next, we implement the forward propagation layers on the FPGA to evaluate the pipelined-parallel and folded FFT architecture. In both the training and evaluation phase we set $Q1.15$ and $Q5.11$ fixed point number format for the weight parameters and output vector, respectively.During the training phase, we set the learning rate, $\alpha = 0.1$, and define the error function as $E = 0.5\|r-y(L)\|^2$. Table I lists the layer sizes of the NN, with different configurations and inference accuracy. For example, configuration $(28 \times 28) - 784(16) - 1024(16) - 512(16) - 10$ indicates a 2-layer FC-NN to classify a 28×28 MNIST image, where the first layer has 784 input channels, 1024 output channels with size-16 BCM partition, and so on. The first three NN models in Table I are trained for five epochs with a mini-batch size of 10. Cases 1 and 4 are set as benchmarks where no FFT-BCM partitions were applied to the weight matrices. Comparing cases 1 and 2 and 4 and 5, we observe that the BCM and pipelined-parallel FFT technique degrades the training accuracy by only 0.5%. Using other NN architectures with more convolution and FC layers, such as LeNet and AlexNet, ∼99% accuracy can be achieved. The proposed approach can improve these NNs by reducing the power consumption in the matrix-vector multiplication blocks.

Fig. 3 plots the mean square error (MSE) as a function of the number of batches for each case listed in Table I. Inference accuracy of more than 90% is achieved for MSE $<$ 0.05. For the CIFAR-10 dataset (case 3 in Table I), we observe that the MSE begins to plateau at approximately 0.3. This is expected as the CIFAR-10 colored images are significantly more complex than the MNIST datasets. Therefore, we implement two convolution layers with six kernels of size 3×3 at each of the layers, followed by pooling and FC layers to achieve $>$ 90% training accuracy (cases 4 and 5 in Table I). For these two cases, we train the NN for 20 epochs or 120000 batches (n = 5 in Fig. 3) to obtain MSE $<$ 0.05. Cases 2 and 5 require more iterations toward convergence during the first epoch due to the reduced number of weight parameters in the BCMs.

For the functional verification, we write VHDL scripts using the Modelsim software to implement the forward propagation layers of the NN models listed in Table I. Fig. 4 shows the

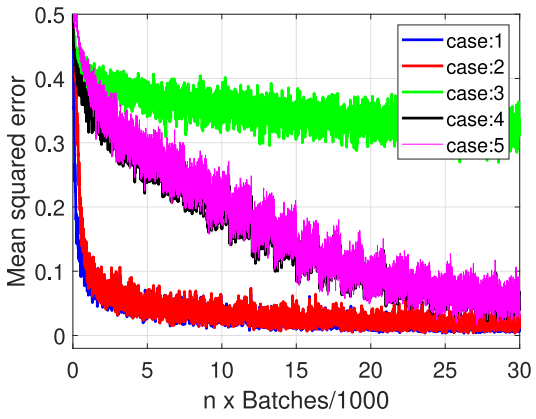

Fig. 3. Training MSE as a function of number of batches.

Fig. 4. Functional verification result. Here, the subscript i and j denote the top and bottom eight values of the parallel architecture, respectively.

final output values for one specific image (from case 2 in Table I) in hexadecimal format. Here, the output vector (y) is obtained by applying the nonlinear ReLU operation on the addition result of bias (*BIAS*) and weight-input multiplication (*sum*) values.

To confirm that the output from functional verification is correct, we convert the $Q5.11$ fixed-point number from Modelsim to a decimal floating-point value. From Fig. 4, it is seen that the only nonzero value of the output vector (y) is 0×0873. The equivalent floating-point number is 4.225. We obtain the same value from the MATLAB simulation result. As the order of this nonzero value is 1, the image is recognized as digit 0.

Next, we implement the NN models on Stratix-10 FPGA using VHDL. For example, in case 2, we implement three forward propagation layers, each containing FFT, element-wise input-weight multiplication (EWM), IFFT, and adder blocks to generate the output. Fig. 5 shows the register-transfer level (RTL) schematic of one layer generated from the Quartus Prime (version 22.3) software. The FFT and IFFT blocks were implemented based on R2MDC and R2SDF architecture to compare their power consumption. It is important to note that the FFT of the weight parameters is not evaluated using FPGA. The FFT of weight-parameters is calculated during the training phase, and they are loaded to the RAMs in FPGA through memory initialization files (MIFs). The design of R2MDC and R2SDF FFT/IFFT blocks on FPGA achieves a clock rate of

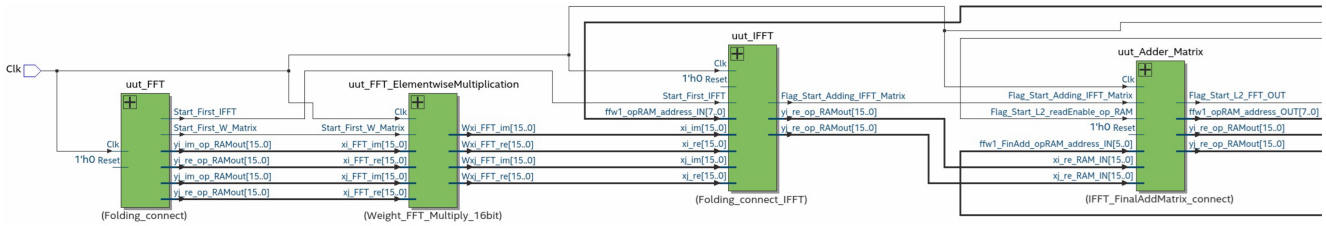

Fig. 5. RTL schematic of one layer in an NN with R2MDC architecture.

TABLE II
POWER CONSUMPTION, MEMORY, AND ACCURACY

Case no.	Power (W)	Memory (Mb)	Accuracy(%)
1	7.57	20.7	97.3
2 (Series)	2.52	5.86	96.8
2 (Parallel)	1.39	5.86	96.8
4	11.72	57.2	92.3
5 (Series)	3.41	15.6	91.9
5 (Parallel)	1.87	15.6	91.9

375 MHz. However, the throughput of R2MDC is twice that of R2SDF, as the former processes two inputs per clock cycle. Next, we measure the dynamic power consumption using the power analyzer tool in Qaurtus Prime. By keeping the same 350 Mb/s throughput rate for both architectures, the dynamic power consumption by the 16-point R2MDC and R2SDF FFT blocks is 37 and 65 mW, respectively. This result shows that the power consumption by the R2MDC architecture is not precisely 50% less than that of R2SDF. This is due to the excess capacitance from the registers (22 in R2MDC versus 15 in R2SDF).

The FPGA implementation methodology will be explained using an example. As shown in case 2 (from Table I), an MNIST image contains 784 input values. In the first NN layer, the FFT block processes two input samples in each clock cycle to sequentially implement a 16-point Fourier transform on the 784 samples. Next, the FFT of the input samples enters the EWM block, where $1024/16 \times 784 = 64 \times 784 = 50176 \times 1$ weight vector is stored in the RAM. The BCM technique reduces the size of the weight vector from $1024 \times 784 = 802816 \times 1$ to 50176×1. In this block, the weight vector (50176×1) and 64 sets of input vector (784×1) are element-wise multiplied to produce the 50176×1 output vector. Next, the IFFT block processes two input samples in each clock cycle to sequentially implement the 16-point inverse Fourier transform on the 50176×1 vector. The fourth block (*uutAdderMatrix* in Fig. 5) adds this 64 sets of 784 values in 50176×1 vector by a group of 16 to produce the $64 \times 16 = 1024 \times 1$ vector. This block also adds the bias to create the 1024×1 output vector from the first NN layer. The bias vector is also stored in the RAM of the fourth block. We repeat a similar process in the second and third layers. Table II shows the dynamic power consumption, required memory, and accuracy for cases 1, 2, 4, and 5 at a throughput rate of 350 Mb/s. For cases 2 and 5, the parallel architecture consumes 45% less dynamic power than the series architecture. The accuracy and required memory are the same due to the similar weight and bias vector size.

V. CONCLUSION AND FUTURE WORK

We propose a pipeline and parallel FFT architecture-based NN with 45% less power consumption than the series architecture. The size of the BCM is limited to 16×16 due to the 16-point folded FFT architecture. In [5], a maximum 8×8 BCM was implemented using an 8-point radix-2 optical FFT architecture. However, it is possible to further increase the BCM size with a larger FFT processor on an FPGA using techniques, such as radix-3 pipelined SDF [9]. This will further increase the power and area efficiency of a BCM-based NN.

REFERENCES

[1] B. Jena, S. Saxena, G. K. Nayak, L. Saba, N. Sharma, and J. S. Suri, "Artificial intelligence-based hybrid deep learning models for image classification: The first narrative review," *Comput. Biol. Med.*, vol. 137, Oct. 2021, Art. no. 104803.

[2] S. S. Yadav and S. M. Jadhav, "Deep convolutional neural network based medical image classification for disease diagnosis," *J. Big data*, vol. 6, no. 1, pp. 1–18, 2019.

[3] C.-Y. Chen, Y.-S. Dai, and H.-C. Hong, "A neuromorphic spiking neural network using time-to-first-spike coding scheme and analog computing in low-leakage 8T SRAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 5, pp. 848–859, May 2024.

[4] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1737–1749, Sep. 2018.

[5] J. Gu et al., "Toward hardware-efficient optical neural networks: Beyond FFT architecture via joint learnability," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 9, pp. 1796–1809, Sep. 2021.

[6] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 2857–2865.

[7] M. Ayinala, M. Brown, and K. K. Parhi, "Pipelined parallel FFT architectures via folding transformation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 6, pp. 1068–1081, Jun. 2012.

[8] M. Garrido, "A survey on pipelined FFT hardware architectures," *J. Signal Process. Syst.*, vol. 94, no. 11, pp. 1345–1364, 2022.

[9] C. Yu and M.-H. Yen, "Area-efficient 128-to 2048/1536-point pipeline FFT processor for LTE and mobile WiMAX systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 9, pp. 1793–1800, Sep. 2015.