Reconfigurable and Low-Complexity Accelerator for Convolutional and Generative Networks over Finite Fields

Weihong Xu, Zaichen Zhang, *Senior Member, IEEE*, Xiaohu You, *Fellow, IEEE*, and Chuan Zhang, *Member, IEEE*

***Abstract*—Convolutional neural networks (CNNs) have gained great success in various fields, such as computer vision and natural language processing. Besides, with the breakthrough in unsupervised learning, generative adversarial network (GAN) is recently utilized to generate virtual data from limited data sets. The generative model of GAN has impressive applications, such as style transfer and image super-resolution. However, the promising performance of CNN and GAN comes at the cost of prohibitive computation complexity. The convolution (CONV) in CNN and the transposed convolution (TCONV) in GAN are the two operations that dominant the overall complexity. The prior works exploit the fast algorithms, Winograd and fast Fourier transform (FFT), to reduce the complexity of spatial CONV. However, Winograd only supports fixed filter size while FFT has high transform overhead. Moreover, very few works apply fast algorithms to accelerate GAN models. In this paper, a reconfigurable and low-complexity accelerator on ASIC for both CNN and GAN is proposed to address these problems. First, by exploiting Fermat number transform (FNT), we propose two FNT-based fast algorithms to reduce the complexity of CONV and TCONV computations, respectively. Then the architectures of FNT-based accelerator are presented to implement the proposed fast algorithms. The methodology to determine the design parameters and optimize the dataflow is also described for obtaining maximum performance and optimal efficiency. Moreover, we implement the proposed accelerator on 65nm 1P9M technology and evaluate it on various CNN and GAN models. The post-layout results show that our design achieves a throughput of 288.0 GOP/s on VGG-16 with 25.11 GOP/s/mm^2 area efficiency, which is superior to the state-of-the-art CNN accelerators. Furthermore, at least $1.7\times$ speedup over the existing accelerators is obtained on GAN. The resulting energy efficiency is $275.3\times$ and $12.5\times$ of CPU and GPU.**

***Index Terms*—Fast convolution, Fermat number transform, convolutional neural network, generative network, reconfigurable architectures.**

I. Introduction

CONVOLUTIONAL neural network (CNN), as a type of feed-forward artificial neural network, has achieved great success in several fields, such as computer vision [2]–[5], game playing, and natural language processing. Besides, the other vital type of neural networks, generative adversarial network (GAN) [6], is recently developed to produce virtual data from limited training datasets with an unsupervised learning paradigm. The learned features of the generative models in GAN [7]–[9] can be used to create impressive images with various styles. CNN and GAN have attracted worldwide interest due to their powerful capabilities of feature representation. Spatial convolution (CONV) and transposed convolution (TCONV) are the two essential operations of CNN and GAN to perform feature extraction and up-sampling. Due to the existence of CONV and TCONV operations, both CNN and GAN require prohibitive computational complexity to achieve promising performance. Many accelerators [10]–[23] have been developed to speed up the inference tasks under different platforms such as ASIC and FPGA.

The typical CNN models [2]–[5] usually contain millions of parameters. Eyeriss [10] and DNA [11] are two CNN accelerators that improve the hardware utilization and energy efficiency through optimizing the dataflow of CONV computation. However, the peak performance of previous CNN accelerators is bounded by their computing resources since CONV operation takes up over 90% of the total computational complexity [10] and the spatial CONV involves substantial multiply-accumulate (MAC) operations. Although this limitation can be alleviated through increasing the number of on-chip MAC units as in [11], [19], it is impractical to implement massive MAC units for resource-constraint embedded devices such that the DSP resources of FPGA are limited.

TABLE I
EXISTING ACCELERATORS AND ALGORITHMS FOR CNN

Algorithms	Complexity Reduction	Flexibility	Transform Overhead	Ref.
Spatial	No	High	-	[10]–[13]
Winograd	Yes	Low	Low	[14], [17]
FFT	Yes	High	High	[15]–[18]

The alternative solution with higher efficiency is to exploit fast algorithms to reduce the CONV complexity thus accelerating CNN inference without increasing arithmetic units. The Winograd algorithms [24] and fast Fourier transform (FFT) are the common methods for the CONV complexity reduction. These two algorithms transform the spatial CONV computation into Winograd domain or complex domain, where the sliding window operation of spatial CONV is replaced by element-wise matrix multiplication (EWMM). Significant arithmetic reduction has been observed on the existing Winograd-based accelerators [14], [17] and FFT-based accelerators [15]–[18], [25]. However, the existing designs have the following defects as listed in Table I. The Winograd-based CNN accelerators in [14], [17] can only support the speed up for 3×3 CONV layer. But the popular CNNs and GANs, e.g. AlexNet [3] and StarGAN [9], adopt various sized filters ranging from 3×3 to 11×11. The size constraint of Winograd algorithms limits their application on different networks. Although the FFT-based algorithms [15]–[18], [25] are flexible to accelerate CONV

W. Xu, Z. Zhang, X. You, and C. Zhang are with the National Mobile Communications Research Laboratory, Southeast University, China. Email: chzhang@seu.edu.cn. *(Corresponding author: Chuan Zhang.)*

This paper was presented in part at IEEE International Conference on ASIC (ASICON), Guiyang, China, 2018 [1].

computation with different filter sizes, the complex FFT transform modules hinder the efficient hardware implementation. The high transform overhead is caused by the fact that FFT is performed in the complex domain and complex operations are required even if the input data are real numbers.

On the other side, the TCONV computation dominants the complexity of GAN and has similar computing procedure with CONV. Directly applying the traditional CNN accelerators on GAN leads to low efficiency and throughput since large amounts of redundant zeros should be inserted into the input feature map of GAN to realize the up-sampling. To this end, there is an increasing trend of designs [12], [13], [20], [21] that aim at eliminating the redundancy and accelerating GAN. The authors in [13], [20] optimize the dataflow to skip redundant computation and memory access. Instead, the zero-free method for TCONV computation is proposed in [12], [21] to improve the overall efficiency. However, the mentioned GAN accelerators are still bounded by their on-chip computing resources. None of these work exploit fast algorithms to speed up the GAN inference through reducing the complexity of TCONV computation. Therefore, a GAN accelerator taking the advantage of fast algorithms is needed for better efficiency and performance.

In this work, we address the above issues by exploiting Fermat number transform (FNT) [26], which has the similar computation flow with FFT method. Compared to FFT, the transform overhead of FNT is significantly lower since it is performed in finite field instead of complex domain. The low-complexity FNT provides more design space in terms of algorithm and hardware for better efficiency and higher performance. Therefore, we propose a reconfigurable and low-complexity accelerator that can speed up both CNN and GAN. The main contributions of this paper are listed as follows:

- Two FNT-based fast algorithms for the acceleration of both CONV and TCONV computations are proposed, respectively. The complexity analysis shows that proposed fast algorithms significantly reduce the complexity and achieves $1.8\times$ to $7.2\times$ speedup for the popular CNNs and GANs.
- Based on the proposed fast algorithms, the hardware architectures of reconfigurable accelerator are presented. The corresponding optimizations for CNNs and GANs are also introduced to improve the efficiency.
- We develop a unified framework to determine the design parameters for the proposed accelerator under different resource constraints. Besides, the dataflow and tiling parameter can also be optimized by this framework.
- We implement the accelerator on ASIC platform with 65nm 1P9M technology. The post-layout results show that our design achieves a throughput of 288.0 GOP/s on VGG-16 [4] and a throughput of 76.9 GOP/s on DCGAN [7] with 25.11 GOP/s/mm^2 peak area efficiency, which are superior to the state-of-the-art CNN and GAN accelerators.

The rest of this paper is organized as follows. The brief introduction of CNN, GAN, and their fast algorithms is given in Section II. The proposed FNT-based fast algorithms for CONV and TCONV are presented in Section III. The architecture design and dataflow analysis of proposed FNT-based accelerator are provided in Section IV. The detailed implementation and experiment results are shown in Section V. The relate works are introduced in Section VI. Section VII concludes this paper.

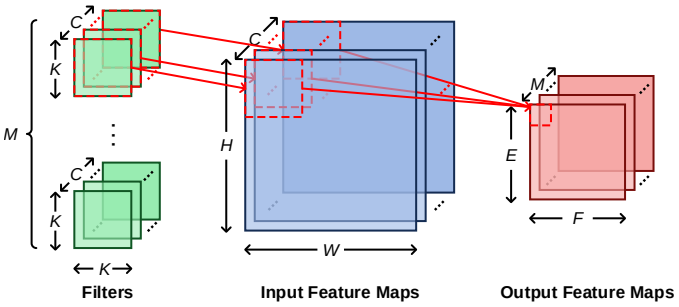
II. Preliminary

The typical CNN is constructed by stacking several types of functional layers: CONV layer, pooling layer, and fully-connected (FC) layer. Apart from these layers, the recent GANs [7]–[9] also contain TCONV layer to perform the up-sampling for images. The following introduces the computation of CONV and TCONV and the fast CONV algorithms.

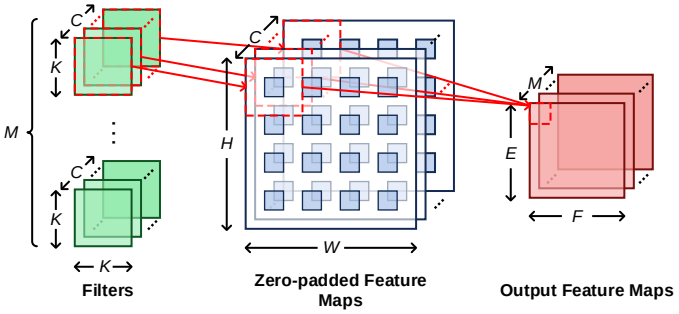
A. Convolution Layer

CONV layer extracts the hidden features from images through performing 2D convolution on the input feature map $\mathbf{I}$ with shape $C \times H \times W$ using the M trained $K \times K$ filters $\mathbf{W}$. The output feature map $\mathbf{O}$ with shape $M \times E \times F$ can be computed as the following spatial CONV (also see Fig. 1a):

$$\mathbf{O}[m][e][f] = \sum_{c=1}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} \mathbf{I}[c][e \times S + i][f \times S + j] \times \mathbf{W}[m][c][i][j]. \quad (1)$$



(a) CONV layer



(b) TCONV layer

Fig. 1. Illustrations of CONV and TCONV layers.

B. Transposed Convolution Layer

The TCONV layer, also named fractionally-strided convolution or deconvolution [27], is used to perform the up-sampling of images from low resolution to high resolution. There are two methods to compute TCONV. The straight computation of TCONV is similar to CONV as shown in Fig. 1b. For a TCONV layer with stride S, $S-1$ zeros will be first inserted between two adjacent pixels of input feature map. Then the filter kernels convolve with the expanded input feature map and

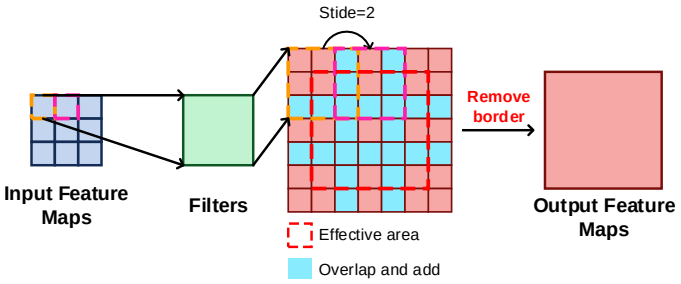


Fig. 2. Illustration of zero-free TCONV computation with $H = W = 3, K = 3, S = 2$.

the up-sampling results are produced. The straight computation of TCONV is called zero-padded TCONV.

The hardware supporting CONV layer can be reused to compute the zero-padded TCONV due to their similarities. However, the inserted zeros are redundant and will enlarge the feature maps during the computation process, which require large memory and lower the efficiency. The zero-free method is introduced in [12], [21] to improve the computation efficiency. As shown in Fig. 2, each pixel of input feature map is first element-multiplied by a $K \times K$ filter kernel. When $S < K$, overlapped area exists between two adjacent output tiles and extra additions are needed to add them up. The outer border of output pixels are invalid results, thus the final output feature maps are obtained by removing the border pixels.

C. Fast Convolution Algorithms for CNNs

The 2D convolution is computation-intensive and requires large amounts of MAC operations. To reduce the arithmetic complexity, fast convolution algorithms, such as Winograd and FFT, are utilized. The following sections briefly introduces fast algorithms based on 2D Winograd and 2D FFT, respectively.

1) Winograd Fast Convolution: The 2D Winograd fast convolution algorithms can be nested by the corresponding 1D algorithms [24]. The 2D Winograd algorithm, defined by $F(m \times m, r \times r)$, computes $m \times m$ CONV results through convolving $r \times r$ filter with the input as follows:

$$Y = A^T \left[\left(GgG^T \right) \circ \left(B^T dB \right) \right] A, \tag{2}$$

where $\circ$ denotes the element-wise multiplication. The matrices B^T, G, and A^T can be found in [17], [24].

After investigating Winograd algorithms, a part of MACs in spatial is converted into additions and the sliding window operation is replaced by element-wise multiplication. As a result, the required MAC operation is reduced from $m^2 \times r^2$ to $(m - r + 1)^2$ for $F(m \times m, r \times r)$.

2) FFT-based Fast Convolution: FFT is the other method to reduce the arithmetic complexity for CNNs, which has similar computation flow with Winograd. In Winograd fast convolution, the given feature maps and filters are transformed into Winograd domain and the EWMM is performed in real domain. For FFT-based fast convolution, the transformation is replaced by FFT while the EWMM is complex multiplication.

The complex arithmetic operation of FFT leads to higher transform overhead and more complicated element-wise multiplication in frequency domain. Two optimization strategies are introduced in [24]. First, the $n \times n$ element-wise multiplication can be realized with $n \times (\lfloor n/2 \rfloor + 1)$ complex multiplications through exploiting the Hermitian symmetry of real input signal. The second optimization is to reduce the complexity of each complex multiplication from 4 real multiplications to 3 real multiplications [24].

III. Fast Algorithms based on Fermat Number Transform

A. Fast Fermat Number Transform

The previous works [14]–[18] have demonstrated that both FFT and Winograd algorithms can effectively reduce the computational complexity of CNNs. However, there are still several defects in the mentioned CNN accelerators. The FFT-based accelerator designs [15]–[18] are flexible and can support various filter sizes by adjusting the tiling lengths. But the overhead of FFT transform and EWMM in complex domain are high. Compared to FFT, Winograd algorithms have lower transform overhead since the transform matrices B^T, G, and A^T in Eq. (2) can be simplified to addition and shift operations. However, this is true only for small m and r, e.g. $m = 2, r = 3$. It is impractical to realize the transform matrices just by shift and addition when m and r grow. Therefore, existing Winograd-based CNN accelerators [14], [17] can only enjoy the benefits of complexity reduction for fixed 3×3 filter size.

We solve the aforementioned problems by adopting FNT [26], [28], which has the same transform formula as FFT. Instead of the complex transform kernel $\omega = e^{-2\pi i/N}$ in FFT, FNT uses the real transform kernel $\alpha = 2$ in finite field [28]. Computations in finite field bring several advantages. First, compared to FFT, the intermediate memory requirement of FNT is reduced by half because only unsigned integers need to be stored. Second, the transform process is greatly simplified since $\alpha = 2$ eases the computation by using shifts and additions rather than complex multiplication in FFT. Last, the complex EWMM of FFT in frequency domain is replaced by real EWMM over finite field, greatly saving the required multiplication operations.

Without loss of generality, the FNT of a real sequence x with length N is defined as:

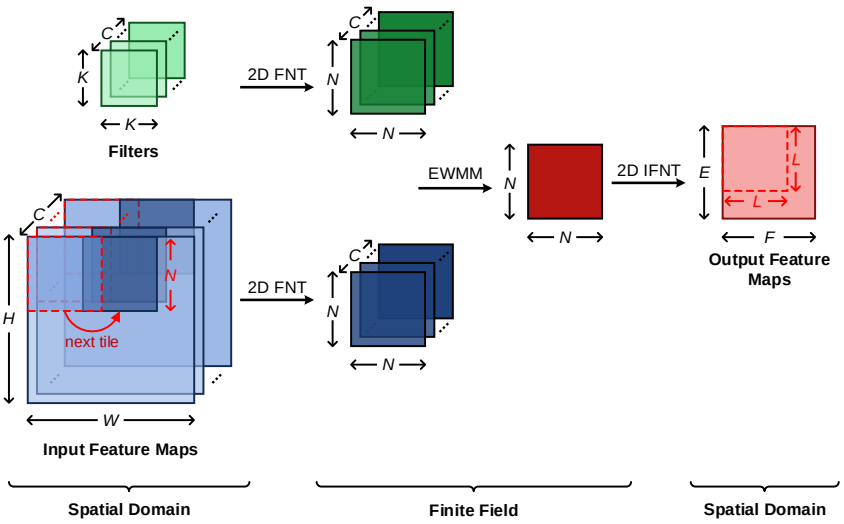
$$X_k = \sum_{n=0}^{N-1} \left\langle x_n \cdot \alpha^{nk} \right\rangle_{F_t}, k = 0, 1, \cdots, N-1, \tag{3}$$

where F_t and $\langle \cdot \rangle_{F_t}$ denote the t-th Fermat number and modular operation, respectively. We define $X = \mathcal{F}(x)$ as the 1D FNT operation. Similarly, the inverse Fermat number transform (IFNT) is defined as:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} \left\langle X_k \cdot \alpha^{-nk} \right\rangle_{F_t}, n = 0, 1, \cdots, N-1, \tag{4}$$

where $x = \mathcal{F}^{-1}(X)$ denotes the 1D IFNT operation.

FNT has a strict relationship between the bit width of input data b, the Fermat number F_t, and the transform length N [26], [29]. These parameters should be carefully selected to obtain



(a) Illustration of 2D OaS FNT

```
for(e=0; e<E; e++)
 for(f=0; f<F; f++)
  for(m=0; m<M; m++)
   for(c=0; c<C; c++)
    O[m][e][f]+ =
  Σ_{i=1}^{K} Σ_{j=1}^{K} I[c][e*S+i][f*S+j] × W[m][c][i][j]
```

(b) Spatial CONV computation

```
for(e=0; e<E; e+=Te)
 for(f=0; f<F; f+=Tf)
  for(m=0; m<M; m+=Tm)
   for(c=0; c<C; c+=Tc)
--- Computation within accelerator ---
    for(te=e; te<min(e+Te,E); te++)
     for(tf=f; tf<min(f+Tf,F); tf++)
      for(tm=m; tn<min(m+Tm,M); tm++)
       for(tc=c; tc<min(c+Tc,C); tc++){
        O_temp+ = F2(I_tile) ∘ F2(W_tile)}
  O_tile = F2^{-1}(O_temp)
```

(c) CONV computation with 2D OaS FNT

Fig. 4. Illustration of 2D OaS FNT and pseudo code for spatial CONV and 2D OaS FNT.

the correct results. The constraints between these parameters are given by:

$$\begin{cases} b &= 2^t, \\ F_t &= 2^b + 1, \\ N &= 2^{t+1}, \end{cases} \tag{5}$$

where b is the bit width of data that will not cause overflow.

TABLE II
POSSIBLE PARAMETERS OF FNT FOR CNN INFERENCE

α	t	Bit width b	Fermat number F_t	Transform length N
2	3	8	2^8+1	16
	4	**16**	$\mathbf{2^{16}+1}$	**32**
	5	32	$2^{32}+1$	64

Several possible parameters of FNT for the inference task of CNN are listed in Table II. The selection of the parameters should consider both CONV and TCONV layers. The bit widths 8, 16, and 32 have been verified to guarantee sufficient arithmetic precision for CONV layers in the previous works [10], [12], [16]. However, according to [12], the TCONV layers in GANs require higher arithmetic precision (activation bit width > 8) to generate high-quality images. Besides, the wider bit width, leading to greater transform length N, requires larger memory to store the intermediate results. Hence, the parameters of FNT are selected as $\alpha = 2$, $t = 4$, $b = 16$, $F_t = 2^{16} + 1$, and $N = 32$ throughout this paper.

B. 2D OaS FNT for Fast Convolution

The input of CONV layers in CNN are 2D feature maps with multiple channels. 2D convolution is carried out between the input feature maps and corresponding filters. The 2D convolution can be efficiently calculated using 2D FNT and 2D IFNT as Eq. (6) since FNT has the same cyclic convolution property of FFT [26]. First, the 2D FNTs, $\mathcal{F}_2(\mathbf{I})$ for input feature map $\mathbf{I}$ and $\mathcal{F}_2(\mathbf{W})$ for filter $\mathbf{W}$, are computed, respectively. Then the convolution results are obtained by performing 2D IFNT on the EWMM of $\mathcal{F}_2(\mathbf{I})$ and $\mathcal{F}_2(\mathbf{W})$.

$$\mathbf{O} = \mathcal{F}_2^{-1}(\mathcal{F}_2(\mathbf{I}) \circ \mathcal{F}_2(\mathbf{W})), \tag{6}$$

where the input size should satisfy $H, W \leq N - K + 1 = L$.

The convolution in Eq. (6) is only feasible for small input feature maps. For those cases where the size of input feature map exceeds $N \times N$, the input feature map $\mathbf{I}$ should be partitioned into small tiles that satisfy the size constraints. Overlap-and-Add (OaA) and Overlap-and-Save (OaS) [28] are the two techniques to compute the convolution for large input feature maps. The OaA method partitions the input feature maps $\mathbf{I}$ into non-overlapping tiles with size $L \times L$ and $K - 1$ pixels of adjacent tiles in the output are overlapped and added. For the OaS method, the input feature maps $\mathbf{I}$ are split into tiles with size $N \times N$ and $K - 1$ pixels are overlapped. The first $K - 1$ pixels of each output tile are discarded. There is no overlap area in the output pixels. The OaA and OaS methods have the same computational complexity whereas the OaA requires additional modules to add up the overlapped parts of output tiles and will cause memory conflict [17]. Therefore, we choose OaS in this work.

The basic steps of 2D OaS FNT are illustrated in Fig. 4 (a). First, each tile of the input feature maps and the associated filters are transformed into finite field by 2D FNT, respectively. Then the 2D FNTs of input feature maps and filters are multiplied element-wisely within each channel. The 2D IFNT is performed on the accumulated EWMM results to produce the output tiles with shape $L \times L$. Fig. 4 (b) and Fig. 4 (c) list the pseudo code for spatial CONV and 2D OaS FNT-based CONV, respectively. For 2D OaS FNT, the partial sum (Psum) $\mathbf{O}_{\text{temp}}$ is reused and accumulated on input channel level. To save the iterative transforms, 2D IFNT will be performed after all C input channels have been calculated. Each channel contains $\lceil \frac{E}{L} \rceil \times \lceil \frac{F}{L} \rceil$ tiles with size $N \times N$. The above procedure is

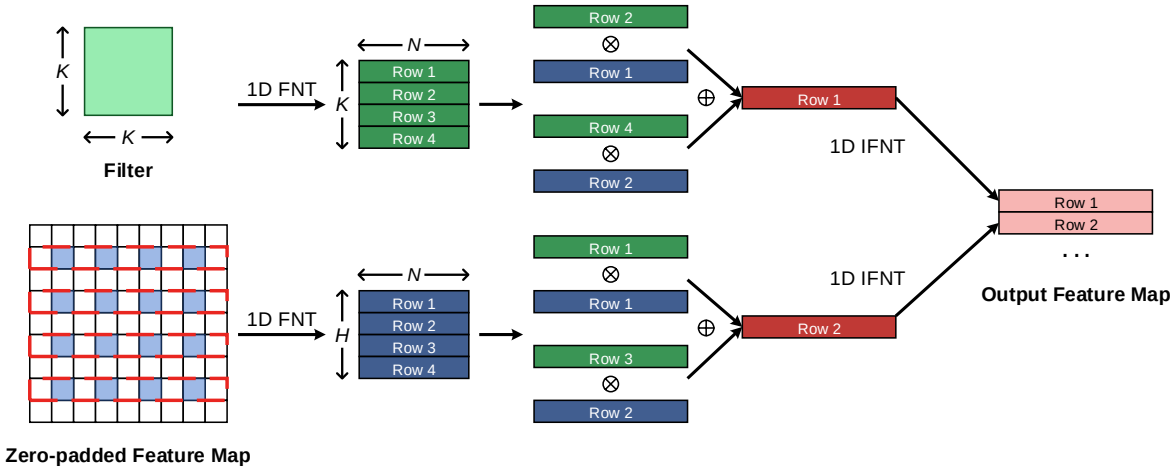


Fig. 5. Illustration of TCONV computation based on 1D OaS FNT.

repeated until all the data have been processed. Comparing the two pseudo codes, the sliding window operation of spatial CONV is parallelized by EWMM in FNT. As a result, the MAC complexity of each tile is reduced from $(H-K+1)^2 \times K^2$ to N^2 with a speedup ratio of $\frac{(H-K+1)^2 \cdot K^2}{N^2}$.

C. 1D OaS FNT for Fast Transposed Convolution

As mentioned in Section II-B, the zero-padded method for TCONV computation requires high memory bandwidth and complexity due to the substantial padded zeros. In contrast, the zero-free method [12], [21] in Fig. 2 leads to low memory bandwidth requirements and low complexity. However, additional modules are needed to process the overlapped pixels and discard redundant results, which are unfriendly for hardware implementation.

The 2D OaS FNT can be directly used to compute the TCONV layers. The process is similar to that of CONV layers in Fig. 4. The only difference is the input feature maps of TCONV layers should be padded with zeros before being transformed into finite filed. However, the drawback of using 2D OaS FNT to compute TCONV is that redundant zeros are inserted into the feature maps and transfered between off-chip memory and on-chip buffer. In this case, the limited off-chip memory bandwidth becomes the bottleneck and results in the imbalance between data loading and computation.

We propose the 1D OaS FNT method in Fig. 5 to provide a tradeoff between the computational complexity and required memory bandwidth. Assuming a TCONV layer accepts $H \times W$ input feature maps and the filter size is $K \times K$ with stride S and padding size P, $(H-1)\times(S-1)+2\times(K-P-1)$ zero rows are inserted into the input feature map, where $2\times(K-P-1)$ rows are inserted around the image border. The redundant zero rows in the padded-feature maps can be exploited to mitigate the imbalance of memory access and computation. Instead of processing the whole zero-padded feature map in 2D OaS FNT, the proposed 1D OaS FNT only performs 1D FNT on the H non-zero rows as shown in Fig. 5. Then 1D FNT sequences of the non-zero rows in zero-padded feature maps and filters are multiplied and added element-wisely. The benefits of 1D OaS FNT method lie in two aspects. First, the required DRAM memory bandwidth to fetch input feature maps is reduced by about 50%. Second, about 50% redundant MAC operations to compute output results are avoided compared to the plain 1D OaS FNT through skipping the zero rows. According to the complexity analysis in Section III-E, 1D OaS FNT results in lower complexity than zero-free method.

D. Optimization for Small Feature Maps

The 2D OaS technique improves the computing efficiency when $H \gg N$. However, the sizes of feature maps vary in different CONV layers due to the existence of pooling layer. The feature map size will gradually shrink as the CONV layer goes deeper. For example, several CONV layers have 7×7 and 14×14 input sizes in GoogLeNet [30], ResNet-50 [2], and YOLONET [5]. For the case $H + K \ll N = 32$, only $(H-K+1)^2$ pixels of the $L \times L$ output are valid while the rest of the pixels are wasted, resulting in a low utilization ratio of $(H-K+1)^2/L^2$. Zeng *et al.* [15] point out that the efficiency can be improved by choosing the appropriate transform length to fit the input size. But the constraints between transform length and bit width in Eq. (5) make it impractical for FNT.

Similar to [15], we use the batch processing technique to improve the computing efficiency. The batch processing is a widely used technique in both spatial CONV [10], [11] and FFT-based CONV [15] to process multiple independent input feature maps at a time. The filter weights are reused between different input feature maps thus the average off-chip memory access is reduced at the expense of longer processing latency. Fig. 6a shows an example of batch processing for 2D FNT with batch size $B = 4$. Specifically, the 4 input feature maps with size $H \times W$ are first grouped into the input feature maps batch with 2×2 grid. The area between feature maps is padded by zeros. After the 2D FNTs of filter and feature maps are multiplied and transformed back to real domain, the final 4 output feature maps are obtained from the output feature maps batch. The maximum batch size that can be processed within single $N \times N$ map is $\lfloor N/(H+2P) \rfloor \times \lfloor N/(W+2P) \rfloor$, where P denotes the padding size. For example, the maximum batch

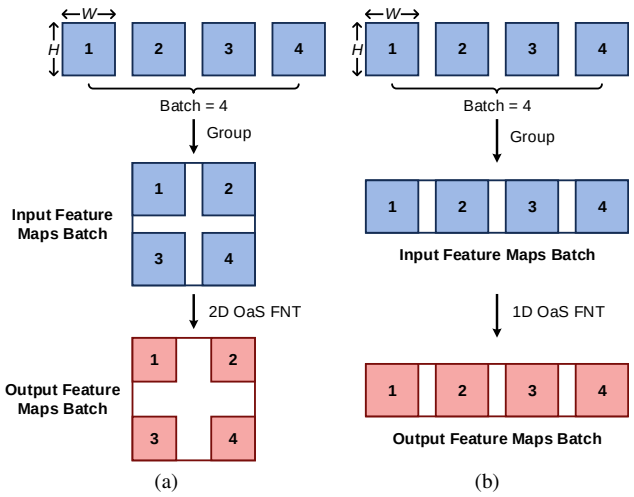


Fig. 6. An example of batch processing for (a) CONV and (b) TCONV computations with batch size $B = 4$.

size is 4 for 14×14 input feature maps. In this case, the computing efficiency is $4\times$ while the complexity of N^2 MACs stays the same compared to the original 2D OaS FNT.

Likewise, the existing GAN models also contain small feature maps with size from 1×1 to 8×8 in DiscoGAN [8] and DCGAN [7]. Fig. 6b illustrates the batch processing technique for 1D OaS FNT-based TCONV computation. In this example, the 4 input feature maps are grouped into one row of the input feature maps batch. Each row of small feature maps is concatenated and processed by 1D OaS FNT.

E. Complexity Analysis

The comparison between four CONV algorithms, namely spatial CONV, Winograd with $F(2 \times 2, 3 \times 3)$, 2D OaA-FFT with $N_{\text{FFT}} = 16$, and 2D OaS FNT with $N = 32$, is conducted to evaluate their computational complexity in terms of MACs (see Fig. 7). The batch size is $B = 16$. Five popular CNN models, AlexNet [3], VGG-16 [4], YOLONet [5], GoogLeNet [30], and ResNet-50 [2], are studied in the experiment. For Winograd, the $F(2\times2, 3\times3)$ configuration is the same as [14], [17]. We assume that the CONV layers with non-3×3 filters are computed by spatial CONV. 2D OaA-FFT with transform length $N_{\text{FFT}} = 16$ is the same as [15]. The optimizations in Section II-C are used to reduce the EWMM complexity from $4n^2$ to $3n\times(\lfloor n/2 \rfloor+1)$. We observe that the proposed 2D OaS-FNT achieves the lowest complexity for all five CNN models among the four CONV algorithms. $1.8\times$ to $7.2\times$ speedup is achieved on different models. The gain comes from the following factors. First, FNT has low transform overhead by using shift and addition. No additional multiplication is needed for FNT and IFNT. Second, all the operations and intermediate results of FNT are in real domain thus have lower complexity than FFT. Moreover, unlike Winograd for fixed 3×3 filter size, FNT-based algorithms can be applied to all CONV layers, exploiting more opportunities to reduce the complexity.

Fig. 8 shows the required MAC complexity and DRAM memory access of zero-padded, zero-free, and 2D OaS FNT

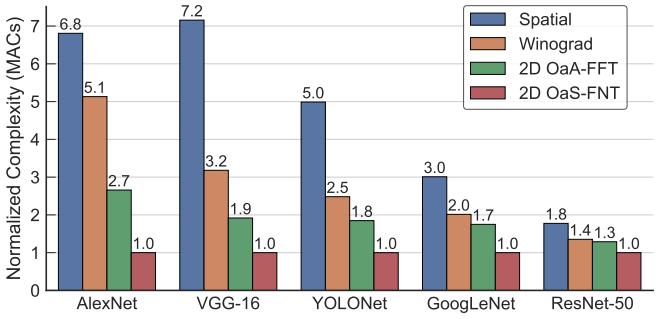
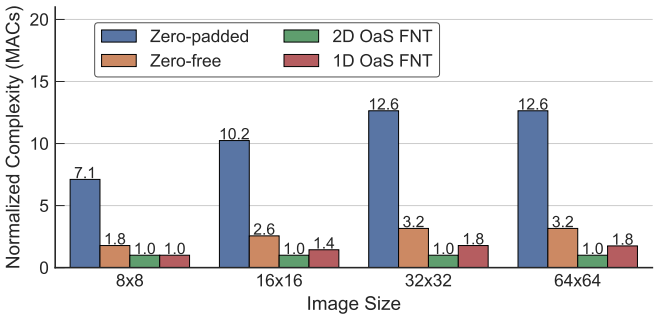
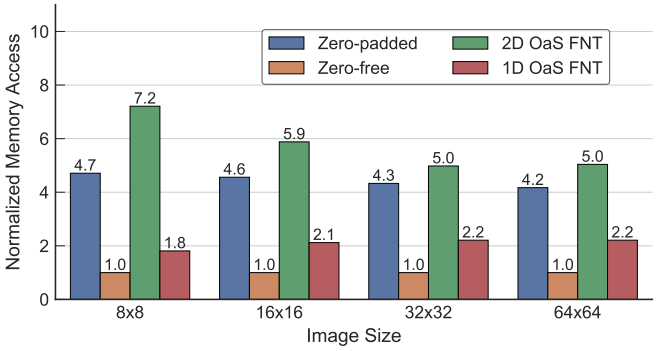


Fig. 7. Computation complexity comparison of spatial, Winograd $F(2 \times 2, 3 \times 3)$, 2D OaA-FFT ($N_{\text{FFT}} = 16$), and 2D OaS-FNT ($N = 32$) with batch size 16 on popular CNNs.

methods to compute images with different sizes. The kernel size is $K = 4$ with stride $S = 2$ and padding size $P = 1$, which is widely used in existing GANs [7]–[9]. The batch size is set to 16. 2D OaS FNT has the lowest MAC complexity under all image sizes. However, the DRAM memory access of 2D OaS FNT is $5.0\times$ to $7.2\times$ of zero-free method since the padded zeros and OaS increase the size of input feature maps.



(a) Normalized MAC complexity



(b) Normalized memory access

Fig. 8. Memory access and MAC complexity of zero-padded, zero-free, 2D OaS FNT, and 1D OaS FNT under various image sizes with $K = 4$, $S = 2$, and $P = 1$. The batch size is 16.

Compared to 2D OaS FNT, the proposed 1D OaS FNT method significantly reduces the memory access while retaining low complexity. Compared to zero-free method [12], [21], 1D OaS FNT still achieves a MAC complexity reduction of about

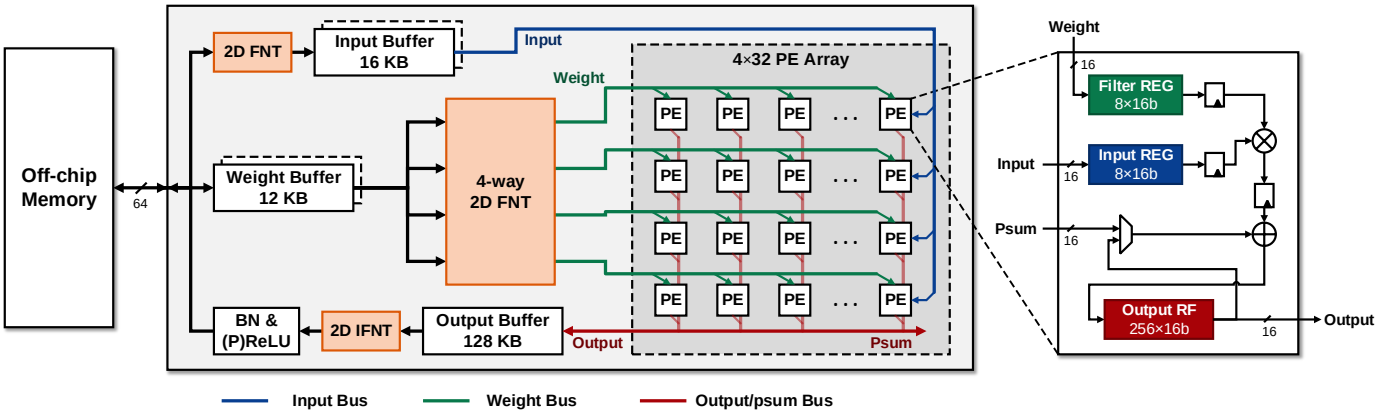


Fig. 9. Overall architectures of proposed accelerator for CNN and GAN acceleration.

44%. Meanwhile, 47.6% to 61.7% of the memory bandwidth is saved compared to zero-padded method.

IV. FNT-BASED ACCELERATOR DESIGN

In this section, we present the hardware architecture designs based on the fast algorithms in Section III. The FNT-based fast CONV and TCONV have significantly lower computational complexity compared to the original spatial CONV, thus relaxing the required number of arithmetic logic units (ALUs). However, several other challenges emerge in the hardware designs for FNT-based accelerator, which can be summarized as the following aspects:

- **Data and Memory Access**: To avoid the on-chip transform modules of filter kernels, the FFT of filters in [15], [16] is pre-calculated before loading to the on-chip buffer. The off-line transform increases the data size of filters by a factor of $(\frac{N}{K})^2$, which requires high off-chip memory bandwidth when $N = 32$. Besides, how to select the optimal dataflow is also crucial for minimizing the cost of data movement.
- **Reconfigurable Design**: Different from the previous designs [14]–[17] that only accelerate CONV layers, Section III introduces both 1D and 2D FNT-based fast algorithms for CONV and TCONV layers. Therefore, reconfigurable architectures are needed to provide the support for multiple acceleration modes.
- **Hardware Utilization**: The reduced complexity alleviates the computation-intensive CNN and GAN. In this case, the limited memory bandwidth would become the bottleneck of the system. The accelerator design should consider how to match the memory bandwidth with the computing capabilities. Besides, the performance and hardware utilization of the accelerator also need to be maximized.

A. Overview

The overall diagram of FNT-based accelerator is illustrated in Fig. 9. The proposed accelerator design consists of the on-chip global buffer, a processing element (PE) array, and six 2D FNT/IFNT modules. Each PE contains an 8×16-bit input register (REG), an 8×16-bit filter REG, a 256×16-bit output register file (RF), and a pipelined MAC.

1) On-chip Global Buffer: The on-chip global buffer is divided into three parts: input buffer (16×2 KB), weight buffer (12×2 KB), and output buffer (128 KB). The determination of the size for global buffer is based on the analysis in Section IV-B. Besides, the double buffering technique is used for input buffer and weight buffer to eliminate the clock stall caused by the data loading of input images and filter weights.

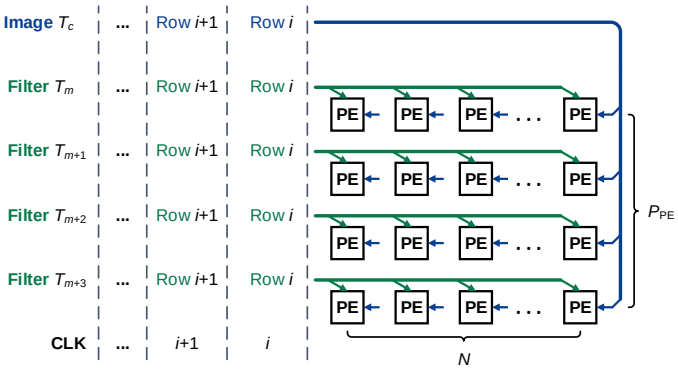


Fig. 10. Dataflow of the PE array with $P_{\text{PE}} = 4$ and $N = 32$.

2) PE Array: A total of $P_{\text{PE}} \times N$ PEs are distributed in the PE array to calculate the element-wise multiplication and accumulation of two FNT sequences. The FNTs of the input feature maps and filter weights are sent to the PE array through the input bus and weight bus, respectively. The dataflow of the PE array is shown in Fig. 10. Each row of the PE array shares the same input from the input buffer. The PE array achieves a parallelism level of P_{PE} by unrolling the output channel level loop in Fig. 4 (c) and concurrently processing P_{PE} filters. Specifically, the same rows from P_{PE} different filters are transformed in the P_{PE}-way 2D FNT module and fed into the corresponding rows of PE array, respectively. The PE array takes N clock cycles to calculate the EWMM results for P_{PE} output channels with size $N \times N$. The partial sum are accumulated in each PE or temporarily stored in the output buffer. According to the analysis in Section IV-B, the parameters are $P_{\text{PE}} = 4$ and $N = 32$ for our design.

3) FNT/IFNT: Five 2D FNT modules are implemented in the accelerator. For input feature maps, their FNT sequences have the same size with their original tiles. This means that no

extra memory is required and the FNTs of input feature maps are transformed once and saved in the input buffer. Hence, one 2D FNT module is set between the I/O interface and the input buffer. Since the 2D FNT of filters will expand by a factor of $(\frac{N}{K})^2$, we choose to transform filter weights on chip to save the off-chip memory bandwidth. The weights fetched from the weight buffer are processed by the four 2D FNT modules within the 4-way 2D FNT module.

After the computation process for each tile is finished, the output results are first transformed back to real domain by the 2D IFNT module. Then the activation and batch normalization (BN) of output are calculated in the BN & (P)ReLU module.

4) BN and (P)ReLU Module: Both ReLU and PReLU activation functions are used in GANs. Moreover, the batch normalization is the other vital functional layer in CNN and GAN. These functions can be realized by MAC unit and comparator through setting different multiplicand and bias values. Therefore, total 32 MAC units implemented in the BN & (P)ReLU module are dedicated for the calculation of the three functional layers.

B. Design Parameters and Memory Access Optimization

The existing CNN and GAN models generally contain millions of filter weights. The limited on-chip buffer resources on ASIC platform are insufficient to store the whole CONV or TCONV layer. Hence, both the input feature maps and filter weights should be partitioned into small tiles before being processed by the chip. The appropriate tiling parameters are essential for the dataflow and hardware utilization.

On the other hand, the off-chip DRAM memory access is bandwidth-limited and energy-consuming. According to [11], [31], the data reuse strategy and tiling parameters have significant impact on the total data access and energy consumption. The dataflows of previous FFT-based accelerators [15]–[17] and Winograd-based accelerators [14], [17] are selected empirically. Instead, based on the methods in [11], [31], we analyze the memory access behavior of proposed FNT algorithms to optimize the memory access and provide the guidelines for the accelerator design.

Three types of data reuse strategies introduced in [31], weight stationary (WS), input stationary (IS), and output stationary (OS), are analyzed under CONV and TCONV layers of different networks. As in Eq. (7), the first goal is to minimize the total energy consumption of memory access E_{Memory} whereas the second one is to maximize the overall performance. The total energy consumption of memory access is simplified to the sum of global buffer and off-chip DRAM access energy since the authors in [31] demonstrate that these two types of memory access dominant the overall energy consumption. The calculation of E_{Buffer} and E_{DRAM} can be found in [11]. For our design in Fig. 9, maximizing the overall performance is equivalent to maximize the parallelism P_{PE} of the PE array.

$$\begin{aligned} &\min\ E_{\text{Memory}} = E_{\text{Buffer}} + E_{\text{DRAM}} \\ &\max\ P_{\text{PE}} \end{aligned} \tag{7}$$

The optimization of Eq. (7) are constrained by the tiling parameters $\langle T_c, T_m, T_e, T_f \rangle$ and global buffer size $\langle B_{\text{input}}, B_{\text{weight}}, B_{\text{output}} \rangle$, where B_{input}, B_{weight}, and B_{output} represent the total size of input buffer, weight buffer, and output buffer, respectively. The associated optimization constraints are listed as follows:

$$\begin{cases} 1 \le T_m \le M, \\ 1 \le T_c \le C, \\ 1 \le T_e \le E, \\ 1 \le T_f \le F, \\ 2 \cdot T_c \cdot T_h \cdot T_w \le B_{\text{input}}, \\ 2 \cdot T_m \cdot T_c \cdot K^2 \le B_{\text{weight}}, \\ T_m \cdot T_e \cdot T_f \le B_{\text{output}}, \end{cases} \tag{8}$$

where the tiles of input feature map with size $T_c \times T_h \times T_w$ are stored in the input buffer while the corresponding T_m filter kernels with T_c channels are stored in the weight buffer. The T_m-channel output feature maps are stored in the output buffer.

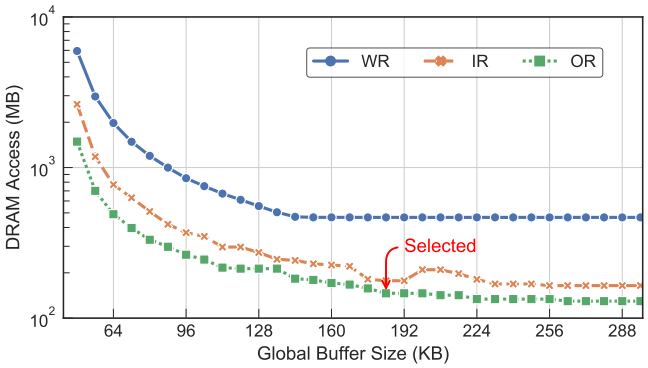


Fig. 11. Off-chip memory access under different data reuse strategies (WR, IR, OR) and different on-chip global buffer sizes on VGG-16.

Optimizing the two goals in Eq. (7) simultaneously is hard so we choose to divide them into two steps. First, we select the proper global buffer size and data reuse strategy that result in the minimum total memory access. Then the attainable maximum parallelism is determined based on the given buffer size and reuse strategy.

1) Global Buffer Size and Data Reuse Strategy: The memory access energy optimization in Eq. (7) is run for every CONV layer of VGG-16 separately. The used energy ratios for DRAM access and global buffer access are identical with [31]. Fig. 11 depicts the relationship between the total off-chip DRAM access and global buffer size for WR, IR, and OR on VGG-16. For each global buffer configuration, the weight buffer size is fixed to 12 KB and the total buffer size of PE array is set to 64 KB. The figure shows that OR consistently has the lowest off-chip DRAM access as the global buffer size increases. In comparison, WR leads to the highest DRAM access. This is because FNT parallels the CONV computation within each filter kernel by element-wise multiplication thus reducing the reuse opportunities of weights. In addition, the amount of DRAM access becomes steady when the global buffer size is greater than 160 KB, which indicates that increasing the global buffer size has little gain. Therefore, a total of 184 KB

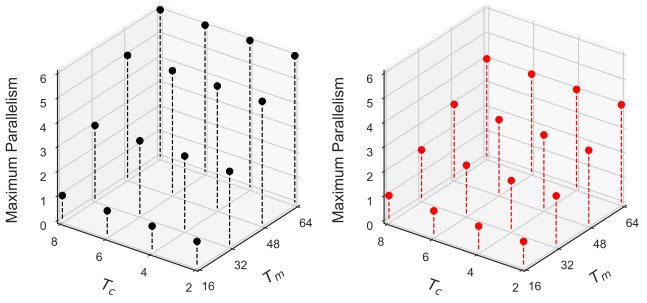
global buffer (16×2 KB for input buffer, 12×2 KB for weight buffer, and 128 KB for output buffer) is allocated and the OS is adopted as the reuse strategy for the accelerator.

2) Parallelism Maximization and Loop Unrolling: The other key design parameter is the parallelism level P_{PE} of the PE array, which is realized through unrolling the filter level loop. In this case, the PE array contains $P_{\text{PE}} \times N$ PEs and processes P_{PE} rows from P_{PE} different filters in parallel. Each PE row is responsible for each row of filter's FNT sequence. The unrolling scheme increases the parallelism and speeds up the computation by a factor of P_{PE} at the expense of more PEs. However, the higher parallelism requires higher bandwidth for data loading. There is a constraint between the parallelism P_{PE} and the off-chip memory bandwidth BW_{DRAM}. For double buffering technique, the data load time T_{load} should not exceed the processing time T_{tile} of each tile to maintain high PE utilization as follows:

$$T_{\text{tile}} \geq T_{\text{load}}. \tag{9}$$

For the CONV computation, T_{tile} and T_{load} are given in Eq. (10). T_{pip} represents the latency of computation pipeline and $Frequency$ is the operating frequency of the chip. BW_{DRAM} denotes the bandwidth of off-chip DRAM memory whereas Q_i and Q_f are the quantized bit width of input feature maps and filter kernels, respectively.

$$\begin{cases} T_{\text{tile}} = \dfrac{N \times T_c \times \lceil \frac{T_m}{P_{\text{PE}}} \rceil \times \lceil \frac{T_h}{N} \rceil \times \lceil \frac{T_w}{N} \rceil + T_{\text{pip}}}{Frequency}, \\ T_{\text{load}} = \dfrac{T_m \times T_c \times K^2 \times Q_i + T_c \times T_h \times T_w \times Q_f}{BW_{\text{DRAM}}}. \end{cases} \tag{10}$$



(a) $K = 3$ (b) $K = 5$

Fig. 12. Maximum parallelism of the PE array under different tiling configurations for $K = 3$ and $K = 5$.

Fig. 12 illustrates the maximum parallelism of the PE array under various tiling parameters T_c and T_m for filter size $K = 3$ and $K = 5$. The maximum parallelism is estimated with the parameters $N = 32$, $T_h = 32$, $T_w = 32$, $Q_i = 16$, $Q_f = 8$ under the global buffer constraints in Section a). The pipeline latency T_{pip} is neglected since it takes up a very small portion of the processing time T_{tile}. It is observed that the number of output channels T_m for each tile determines the maximum parallelism and higher parallelism can be obtained through tiling larger T_m. For the case where $K = 5$, the accelerator with parallelism $P_{\text{PE}} > 4$ is bounded by the memory access rather than the computing units. So we set the parallelism P_{PE} to 4.

C. Fast FNT/IFNT Kernels

The FNT can be efficiently implemented with the same data graph of FFT [1], reducing the transform complexity from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log_2 N)$. Compared to FFT, the butterfly (BF) module of FNT has lower complexity since it is composed of shifter and adder. The complex multiplier in FFT is replaced by simple shift and modular operations. Moreover, the intermediate results of FNT are real values instead of complex numbers, saving about 50% memory over FFT.

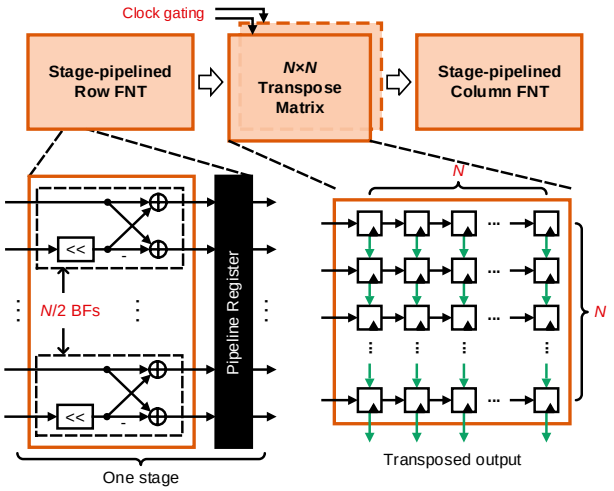
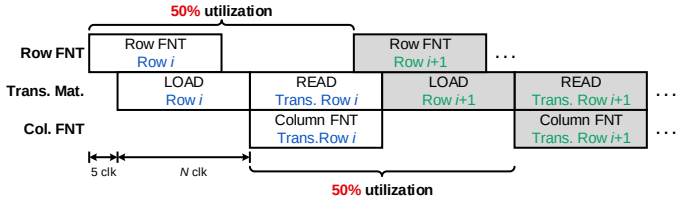


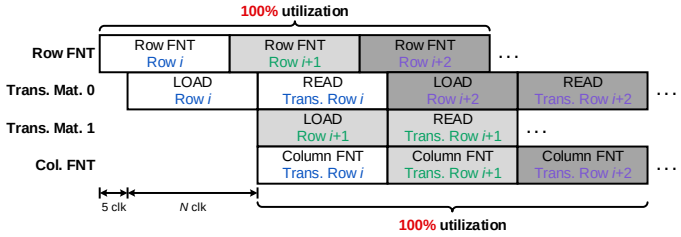
Fig. 13. Stage-pipelined fast FNT kernel with double transpose matrixes for FNT size N.

Fig. 13 illustrates the proposed stage-pipelined fast FNT kernel, which supports both 1D FNT and 2D FNT. Under 2D FNT mode, the outputs of row FNT are first fed into the $N \times N$ transpose matrix. The column FNT processes the transposed output from the transpose matrix and then produces the final 2D FNT results. Under 1D FNT mode, the desired results can be obtained directly from the output of row FNT, thus the transpose matrix and column FNT are bypassed. The row FNT and column FNT modules have $\log_2 N$ stages ($N/2$ BFs on each stage), respectively. A pipeline register is inserted between two consecutive stages to relax the critical path delay. The stage-pipelined design enables the row FNT and column FNT modules to continuously process the input data with a latency of $\log_2 N$ clock cycles. A 2D register array with total of $N \times N$ registers are implemented in the transpose matrix.

As shown in Fig. 14a, it takes N clock cycles for the transpose matrix to receive the incoming data from row FNT module and the transposed output will be sent to the column FNT module in the next N clocks. The new data from the row FNT have to wait for N clock cycles till the output of previous N rows of data in transpose matrix is complete. In this case, the utilization of row FNT and column FNT modules is 50%. To eliminate the stall caused by insufficient memory of matrix transpose, one extra transpose matrix is used to improve the utilization of row and column FNT modules to 100% such that



(a) 2D FNT with single transpose matrix



(b) 2D FNT with double transpose matrixes

Fig. 14. Timing diagrams of fast FNT kernel with and without double transpose matrixes under 2D FNT mode.

the fast FNT kernel can continuously process the input data without any stall clock.

Another two optimization opportunities are utilized to save energy dissipation and hardware overhead. First, when working under 1D FNT mode, the two transpose matrixes are not necessary so they are clock gated to save energy. The other optimization is based on the fact that the filter kernels in CNNs and GANs generally have much smaller size than $N \times N$. The largest filter size of existing popular networks is 11×11 in AlexNet [3]. The valid size of the row FNT results for a $K \times K$ filter is $K \times N$, which means the rest of $N - K$ rows are all zeros. The redundancy is eliminated by implementing 32×11 register arrays in the transpose matrix of 4-way 2D FNT module in Fig. 9. As a result, 65.6% memory of the transpose matrixes is saved for the 4-way 2D FNT module.

D. Performance Modeling

The processing time for each layer consists of the initial data loading time T_{load} and the computation time as follows:

$$T_{\text{proc.}} = T_{\text{load}} + Tiles \times \lceil \frac{C}{T_c} \rceil \times \lceil \frac{M}{T_m} \rceil \times T_{\text{tile}}, \tag{11}$$

where $Tiles$ denotes the total number of tiles in each channel of the input feature map batch as shown in Eq. (12).

$$Tiles = \begin{cases} \lceil \frac{E}{T_e} \rceil \times \lceil \frac{F}{T_f} \rceil, \text{for CONV}, \\ H \times \lceil \frac{F}{T_f} \rceil, \text{for TCONV}. \end{cases} \tag{12}$$

The total operation OP is used to measure the computational complexity of spatial CONV and zero-free TCONV as the following expression:

$$OP = \begin{cases} 2 \times E \times F \times K^2 \times M \times C, \text{for CONV}, \\ 2 \times H \times W \times K^2 \times M \times C, \text{for TCONV}. \end{cases} \tag{13}$$

Similar to FFT and Winograd algorithms in [14], [17], FNT reduces the complexity of spatial CONV and zero-free TCONV, we use the effective performance to evaluate the throughput of accelerator as follows:

$$Performance = \frac{OP}{T_{\text{proc.}}}. \tag{14}$$

The effective performance of accelerators that adopt fast algorithms may exceed the peak performance of spatial CONV with the same number of MAC units [24].

V. Experiment Evaluation

A. Experimental Setup

The FNT-based accelerator in Fig. 9 is implemented in SMIC 65nm LL 1P9M technology. The design parameters in Section IV-B are used. The global buffer and the RF in PE array are generated by commercial Memory Compiler. We synthesize the design on Synopsys Design Compiler. Then the synthesized design is placed and routed using Synopsys IC Compiler. The post-layout simulation is conducted on Synopsys VCS to verify the functions of the chip. The annotated toggle rate of the gate-level netlist is converted into switching activity interchange format (SAIF) and we use Prime-Time PX to measure the chip-only power dissipation.

TABLE III
Network Parameters of Evaluated CNNs and GANs

Name	CONV Layer	TCONV Layer	Filter Size
AlexNet [3]	5	-	$K = 3, 5, 11$
VGG-16 [4]	13	-	$K = 3$
YOLONet [5]	24	-	$K = 1, 3, 7$
GoogLeNet [30]	21	-	$K = 1, 3, 5, 7$
ResNet-50 [2]	49	-	$K = 1, 3, 7$
DCGAN [7]	0	5	$K = 4$
DiscoGAN [8]	5	5	$K = 4$
StarGAN [9]	16	2	$K = 3, 4, 7$

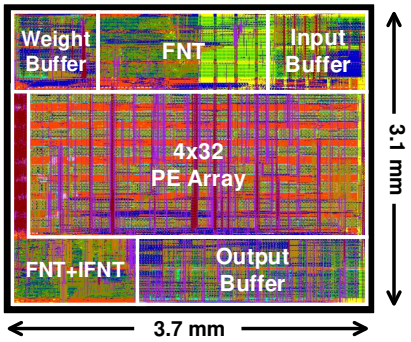
Five popular CNN models (AlexNet [3], VGG-16 [4], YOLONet [5], GoogLeNet [30], and ResNet-50 [2]) and three state-of-the-art GAN models (DCGAN [7], DiscoGAN [8], and StarGAN [9]) are adopted as the benchmarks to measure the performance. The generative models of the three GANs are evaluated. As shown in Table III, existing CNNs and GANs have variable numbers of CONV layer, TCONV layer, and filter size. DCGAN is composed of all TCONV layers while only a fraction of layers in StarGAN are TCONV layers.

The experiment steps are summarized as follows. Section V-B presents the detailed implementation results of FNT-based accelerator. Section V-C conducts an extensive comparison with existing CNN or GAN accelerators on different platforms, including ASIC, FPGA, CPU, and GPU.

B. Implementation Results

1) Detailed ASIC Results: Fig. 15 illustrates the layout, chip characteristics, and area breakdown of the FNT-based accelerator in 65nm technology. The chip with 184-KB SRAM is integrated within a core area of 3.1mm $\times$ 3.7mm. The clock

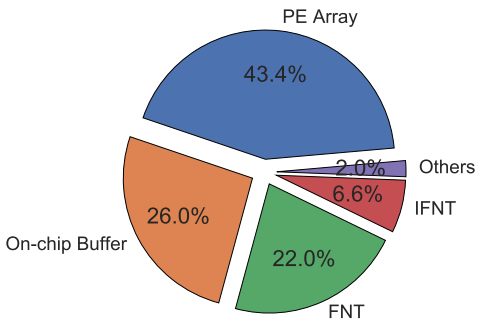
frequency is set to 200 MHz with a supply voltage of 1.2 V. A total of 160 MAC units are implemented and 128 of them are used for the FNT-based CONV and TCONV calculation. The rest 32 MACs are set in BN and PReLU module. The area breakdown analysis indicates that the PE array occupies 43.4% of the total area while the FNT and IFNT modules take up about 28.6% of the total area.



(a) Layout photo

Technology	65nm 1P9M CMOS
Core Area	3.1mm × 3.7mm
Supply Voltage	1.2 V
Clock Frequency	200 MHz
Precision	16-bit fixed point
# of MACs	128 (PE Array) + 32 (BN & PReLU)
On-chip Buffer	Input buffer: 16 KB × 2 Weight buffer: 12 KB × 2 Output buffer: 128 KB Total: 184 KB

(b) Chip specification



(c) Area breakdown

Fig. 15. Layout photo, chip specification, and area breakdown of the FNT-based accelerator in 65nm technology.

2) Results of FFT and FNT Modules: To show the advantages of FNT, the FNT and FFT modules are separately synthesized and the implementation results on 65nm technology are given in Table IV. All FFT and FNT modules are fully-parallel and stage-pipelined. The FFT modules are automatically generated by the tools described in [32]. The table shows that FNT module with size $N = 32$ consumes less area and power than 16-point FFT. The low transform overhead is resulted from the fact that FNT is computed in real domain and only requires shift-add operation, which greatly reduces the memory and arithmetic complexity.

TABLE IV
AREA AND POWER DISSIPATION OF FNT AND FFT MODULES IN 65NM TECHNOLOGY

	N	Area		Power (mW)	
		um^2	**Ratio**	**mW**	**Ratio**
FFT	8	32103.9	1.0×	7.81	1.0×
	16	115751.2	3.6×	25.71	3.3×
	32	357691.6	11.1×	74.4	9.5×
FNT	32	91968.5	2.9×	13.0	1.7×

C. Performance Comparison

1) Comparison with Spatial CONV: The accelerators without the aid of fast algorithms will reach the peak performance when the utilization of all MAC units is 100%. The speedup of our FNT-based accelerator over the spatial CONV with the same number of MAC units is depicted in Fig. 16. The spatial CONV is assumed to have 128 MAC units and work under 200 MHz frequency, which can yield a peak throughput of 51.2 GOP/s. The batch size for CNN models is set to 4 for CNNs and 64 for GANs. The throughput of our design is calculated by Eq. (14). It is observed that our design achieves 1.4× to 5.6× speedup on various CNN and GAN models. Compared to AlexNet, VGG-16, and YOLONet, the speedup on GoogLeNet and ResNet-50 is less significant since the proportion of point-wise 1×1 CONV for these two models is higher and the complexity of 1×1 CONV cannot be reduced. Compared to zero-free TCONV computation [12], [21], our design has about 1.5× speedup on DCGAN with 5 TCONV layers.

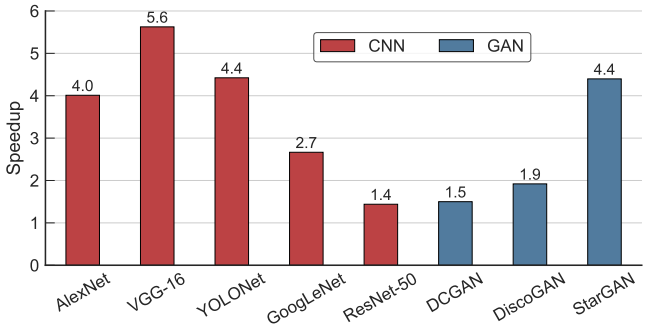


Fig. 16. Speedup of proposed accelerator for various CNN and GAN models. The batch size is 4 for CNNs and 64 for GANs.

2) Comparison with ASIC and FPGA Designs: We compare the performance of proposed FNT-based accelerator with other designs on both FPGA and ASIC platforms. Table V summarizes the comparison with existing accelerators for CNN and GAN. The two FPGA accelerators in [16] and [17] exploit FFT and Winograd algorithms to reduce the complexity, respectively. Four ASIC accelerators, namely Eyeriss [10], DNA [11], GNA [12], and GANAX [13], are taken as the baselines. Among all the baselines, GNA, GANAX, and the proposed accelerator are the three designs that natively support GAN. The throughput of all CONV or TCONV layers is measured

TABLE V
PERFORMANCE COMPARISON WITH EXISTING ASIC AND FPGA DESIGNS

Design	FPGA'2017 [16]	TCAD'2019 [17]	Eyeriss JSSC'2016 [10]	DNA TVLSI'2017 [11]	GNA TCAD'2018 [12]	GANAX ISCA'2018 [13]	**This work**
Platform	FPGA		ASIC				
	Stratix-V GXA7	Xilinx ZCU102	65nm	65nm	28nm	45nm	65nm
Clock Freq. (MHz)	200	200	200	200	200	500	200
Area (mm^2)	-	-	12.25	16.0	1.387 (7.47†)	9.07 (18.9†)	11.47
Precision	32-bit float	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed
Benchmarks	AlexNet, VGG-16	AlexNet, VGG-16	AlexNet, VGG-16	AlexNet, VGG-16	VGG-16, FCN	DCGAN, DiscoGAN	VGG-16, DCGAN
Algorithm	FFT	Winograd	Spatial	Spatial	Spatial	Spatial	FNT
Complexity Reduction	**Yes**	**Yes**	No	No	No	No	**Yes**
Support GAN	No	No	No	No	**Yes**	**Yes**	**Yes**
Throughput (GOP/s)	AlexNet: 83.00 VGG-16: 123.48	AlexNet: 1006.4 VGG-16: 2601.3	AlexNet: 51.4 VGG-16: 24.5	194.4*	VGG-16: 96.5 FCN: 44.2	-	VGG-16: 288.0 DCGAN: 76.9
Power (W)	13.18	23.6	0.278	0.479	0.142 (0.252†)	-	0.805
Area Eff. (GOP/s/mm^2)	-	-	4.20	12.15	69.6 (12.9‡)	-	25.11
Peak Energy Eff. (GOP/s/W)	9.37	105.4	184.9	405.8	679.6 (382.9‡)	-	357.8

* Average throughput. † Scaled to 65nm and 1.2 V. ‡ Efficiency under 65nm technology.

and each MAC operation equals to two operations. The area efficiency and peak energy efficiency are calculated based on the reported results. The area and supply voltage of those designs with different technologies are scaled to 65nm and 1.2 V for the fair comparison.

Our design achieves 3.4× to 38.2× energy efficiency over the Winograd-based [17] and FFT-based [16] FPGA designs. The FFT transform in [16] requires high arithmetic precision thus lowers the overall efficiency. The Winograd-based accelerator in [17] natively support fixed 3×3 CONV. In comparison, our design is more flexible and supports the acceleration for various filter sizes and different layer types.

On the CONV layers of VGG-16, our design yields a throughput of 288.0 GOP/s, which is 1.5× of DNA [11]. The resulting 357.8 GOP/s/W energy efficiency is comparable with other ASIC designs. More importantly, the area efficiency of 25.11 GOP/s/mm^2 is the highest among reported designs. On the other hand, for GAN models, a throughput of 76.9 GOP/s is achieved on DCGAN whereas the throughput of GNA [12] is 44.2 GOP/s on FCN [33]. The proposed accelerator yields about 1.7× speedup and 1.9× area efficiency over GNA under 65nm technology.

3) Case Study on AlexNet: A case study on AlexNet is presented in this section. AlexNet is composed of five CONV layers and three types of filters (11×11, 5×5, and 3×3) are used. The proposed design supports the acceleration for all CONV layers. The corresponding tiling scheme of OS dataflow derived from Eq. (7) are given in Table VI. The batch size is 4 to improve the computing efficiency.

Two CNN accelerators, namely Eyeriss [10] and DNA [11], are adopted as the baselines for performance comparison. Eyeriss contains 168 MAC units while DNA contains 512 MAC units. The layer-wise processing time with batch size 4 for Eyeriss, DNA, and our design on AlexNet is illustrated in Fig. 17. The processing time of our design is consistently

TABLE VI
TILING PARAMETERS FOR ALEXNET

Layer	B	T_c	T_m	T_h	T_w
CONV1	4	1	16	128	32
CONV2	4	6	64	32	32
CONV3	4	8	64	32	32
CONV4	4	8	64	32	32
CONV5	4	8	64	32	32

less than the other two counterparts except for the first layer. The long processing time is because the first layer of AlexNet has large stride $S = 4$. The gain of fast CONV algorithms will decrease as the stride size increases. Finally, our design obtains the smallest average processing time of 6.4 ms and a throughput of 205.4 GOP/s.

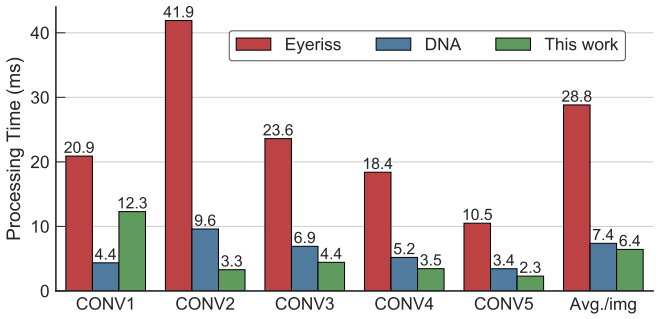


Fig. 17. Comparison of processing time on AlexNet with Eyeriss [10] and DNA [11] with batch size 4.

It should be noted that the speedup of DNA is realized through implementing more MACs on chip. Instead of increasing computing resources, our design exploits FNT algorithms to reduce the complexity and improve the overall throughput at the

expense of extra FNT and IFNT modules on chip. The results show that our FNT-based accelerator yields better performance and higher efficiency with only 25% MAC units of DNA.

TABLE VII
PERFORMANCE COMPARISON WITH OTHER PLATFORMS

Platform	CPU	GPU	ASIC
Device	Intel i7-7700	NVIDIA GTX 1080 Ti	CMOS 65nm
Clock Freq. (MHz)	3600	1923	200
Benchmark	VGG-16		
Precision	32-bit float	32-bit float	16-bit fixed
Throughput (GOP/s)	87.3	6974.2	288.0
Power (W)	65	243	0.805
Latency (ms)	351.4	4.4	106.6
Energy Efficiency (GOP/s/W)	1.3	28.7	357.8

4) Comparison with Other Platforms: The comparison with CPU and GPU platforms is also conducted using the *PyTorch* [34] framework. The CPU Intel i7-7700 and GPU NVIDIA GTX 1080 Ti are adopted as the baselines. The power consumption of GPU is obtained from the NVIDIA profiling tools. The batch size is set to 16. Table VII summarizes the comparison results on VGG-16. The GPU yields the highest throughput ($24.2\times$ of our design) at the cost of 243 W power consumption. Compared to CPU and GPU, our accelerator achieves $275.3\times$ and $12.5\times$ energy efficiency, respectively.

VI. RELATED WORK

A. CNN Accelerators

A large number of designs have been proposed to accelerate CNNs based on spatial CONV. Using the row stationary dataflow, Chen *et al.* [10] propose the energy-efficient accelerator, Eyeriss, to realize low-power CNN inference. Tu *et al.* [11] present the accelerator, DNA, which improves the hardware utilization. Zhang *et al.* [19] propose a unified representation for the FPGA acceleration of CNNs. Gong *et al.* [22] utilizes the roofline model and compression techniques to map the entire CNN model onto FPGA. Zhang *et al.* [23] conduct the analysis on the resource requirement of CNN and propose associated architectures to address the limitation. However, the mentioned designs are computation-bound and no complexity reduction technique is utilized.

On the other hand, the speedup of CNN can also be realized by exploiting fast algorithms, such 2D Winograd [24] and FFT. Liang *et al.* [17] design a line buffer for Winograd algorithms and implement them on FPGA. Shen *et al.* [14] extend the 2D Winograd to 3D cases to accelerate 3D CNNs. However, these Winograd-based designs can only accelerate CONV layers with fixed 3×3 filter size since the transform is simple only for 3×3 filter size. Compared to Winograd, FFT is flexible to support different filter sizes. Abtahi *et al.* [25] use the OaA FFT to speed up CNNs on embedded platforms. The FFT-based accelerators are also evaluated on FPGA in [16], [17]. Zeng *et al.* [15] present a framework to automatically generate high-throughput FFT-based CNN accelerators. Ding *et al.* [18] combine the circulant matrix and FFT to perform efficient inference for CNNs on ASIC. But the high transform overhead of FFT restricts the overall performance of CNN accelerators.

B. GAN Accelerators

The authors in [13] optimize the Eyeriss architecture [10] to skip the redundant zeros of TCONV computation. The resulting design, GANAX, greatly improves the efficiency of GAN acceleration. Song *et al.* [20] design the hardware architectures for both training and inference phases of GAN, which improve the data reuse and reduce redundant operation. Yan *et al.* [12] exploit the zero-free TCONV computation and propose efficient computing engine, GAN, for CONV and TCONV layers. The cross-layer connection are also optimized. Similarly, using the zero-free method, Xu *et al.* [21] present a unified accelerator to handle both CONV and TCONV computations.

VII. CONCLUSION

In this work, we present FNT-based fast algorithms for the acceleration of CNN and GAN. The proposed algorithms are flexible and can effectively reduce the complexity of CONV and TCONV computations. Moreover, the hardware architectures of FNT-based accelerator and the dataflow analysis are provided to improve the overall efficiency. Our design implemented on 65-nm technology achieves a throughput of 288 GOP/s at 200 MHz frequency with 805 mW power dissipation. The resulting area efficiency is higher than the state-of-the-art designs.

REFERENCES

[1] W. Xu, X. You, and C. Zhang, "Using fermat number transform to accelerate convolutional neural network," in *IEEE Int. Conf. ASIC*, 2017, pp. 1033–1036.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770–778.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Adv. Neural Inf. Syst.*, 2012, pp. 1097–1105.

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

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 779–788.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Adv. Neural Inf. Syst.*, 2014, pp. 2672–2680.

[7] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[8] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to discover cross-domain relations with generative adversarial networks," in *Int. Conf. Mach. Learn.*, 2017, pp. 1857–1865.

[9] Y. Choi, M. Choi, M. Kim, J.-W. Ha, S. Kim, and J. Choo, "Stargan: Unified generative adversarial networks for multi-domain image-to-image translation," in *IEEE Conf. Comput. Vision Pattern Recognit.*, 2018, pp. 8789–8797.

[10] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[11] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Trans. VLSI Syst.*, vol. 25, no. 8, pp. 2220–2233, 2017.

[12] J. Yan, S. Yin, F. Tu, L. Liu, and S. Wei, "GNA: Reconfigurable and efficient architecture for generative network acceleration," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2519–2529, 2018.
[13] A. Yazdanbakhsh, K. Samadi, N. S. Kim, and H. Esmaeilzadeh, "GANAX: A unified mimd-simd acceleration for generative adversarial networks," in *ACM/IEEE Annu. Int. Symp. Comput. Archit.*, 2018, pp. 650–661.
[14] J. Shen, Y. Huang, M. Wen, and C. Zhang, "Towards an efficient deep pipelined template-based architecture for accelerating the entire 2D and 3D CNNs on FPGA," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2019.
[15] H. Zeng, R. Chen, C. Zhang, and V. Prasanna, "A framework for generating high throughput cnn implementations on FPGAs," in *ACM/SIGDA Int. Symp. FPGA*, 2018, pp. 117–126.
[16] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *ACM/SIGDA Int. Symp. FPGA*, 2017, pp. 35–44.
[17] Y. Liang, L. Lu, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2019.
[18] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan *et al.*, "CirCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *ACM/IEEE Int. Symp. Comput. Microarchitecture*, 2017, pp. 395–408.
[19] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2018.
[20] M. Song, J. Zhang, H. Chen, and T. Li, "Towards efficient microarchitectural design for accelerating unsupervised gan-based deep learning," in *IEEE Int. Symp. High Perform. Comput. Archit.*, 2018, pp. 66–77.
[21] D. Xu, K. Tu, Y. Wang, C. Liu, B. He, and H. Li, "FCN-engine: Accelerating deconvolutional layers in classic cnn processors," in *Int. Conf. Comput.-Aided Des.*, 2018, p. 22.
[22] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "MALOC: A fully pipelined fpga accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2601–2612, 2018.
[23] J. Zhang and J. Li, "Improving the performance of opencl-based fpga accelerator for convolutional neural network," in *ACM/SIGDA Int. Symp. FPGA*, 2017, pp. 25–34.
[24] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," in *IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 4013–4021.
[25] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Trans. VLSI Syst.*, vol. 26, no. 9, pp. 1737–1749, 2018.
[26] R. Agarwal and C. Burrus, "Fast convolution using Fermat number transforms with applications to digital filtering," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 22, no. 2, pp. 87–97, 1974.
[27] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.
[28] R. E. Blahut, *Fast algorithms for signal processing*. Cambridge University Press, 2010.
[29] T. Toivonen and J. Heikkila, "Video filtering with Fermat number theoretic transforms using residue number system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 16, no. 1, pp. 92–101, 2005.
[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 1–9.
[31] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM/IEEE Annu. Int. Symp. Comput. Archit.*, vol. 44, no. 3, 2016, pp. 367–379.
[32] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer generation of hardware for linear digital signal processing transforms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 17, no. 2, p. 15, 2012.
[33] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2017.
[34] "PyTorch," https://pytorch.org/.