Accelerating Fully Spectral CNNs with Adaptive Activation Functions on FPGA

Shuanglong Liu*, Hongxiang Fan†, Wayne Luk†
* Key Laboratory of Low-Dimensional Quantum Structures and Quantum Control, School of Physics and Electronics
Hunan Normal University, Changsha, China, *liu.shuanglong@hunnu.edu.cn*
† Dept. of Computing, Imperial College London, UK, {*h.fan17, w.luk*}*@imperial.ac.uk*

***Abstract*—Computing convolutional layers in frequency domain can largely reduce the computation overhead for training and inference of convolutional neural networks (CNNs). However, existing designs with such an idea require repeated spatial- and frequency-domain transforms due to the absence of nonlinear functions in the frequency domain, as such it makes the benefit less attractive for low-latency inference. This paper presents a fully spectral CNN approach by proposing a novel adaptive Rectified Linear Unit (ReLU) activation in spectral domain. The proposed design maintains the non-linearity in the network while taking into account the hardware efficiency in algorithm level. The spectral model size is further optimized by merging and fusing layers. Then, a customized hardware architecture is proposed to implement the designed spectral network on FPGA device with DSP optimizations for 8-bit fixed point multipliers. Our hardware accelerator is implemented on Intel's Arria 10 device and applied to the MNIST, SVHN, AT&T and CIFAR-10 datasets. Experimental results show a speed improvement of 6× ∼ 10× and 4× ∼ 5.7× compared to state-of-the-art spatial or FFT-based designs respectively, while achieving similar accuracy across the benchmark datasets.**

I. Introduction

Typical CNN models exhibit high computational complexity, which largely limits their applications in edge devices with resource and power constrained settings such as FPGAs. Previous research [1]–[3] has shown that the computation of convolution in the frequency domain using the Fourier transform (e.g. FFT) achieves significant speedups compared to the conventional implementation in spatial domain. However, the performance of existing frequency-domain designs suffers from repeatedly computing the Fourier transform and its inverse due to the absence of spectral implementation for sub-sampling and non-linear activations.

To address this problem, researchers [4]–[7] have recently introduced the idea of fully spectral implementation of CNNs, which aims to run the model in complete frequency domain during training or inference without the need of domain switching. Despite the success of spectral pooling introduced in [4], which proves to outperform the spatial version by preserving more information per parameter and enabling flexibility in output dimensionality, the design of non-linear activations such as the Rectified Linear Unit (ReLU) remains challenge and an open issue [5]. The main sticking point lies in that the application of non-linear function does not satisfy the linearity property of the frequency domain, and therefore must be implemented in the spatial domain [5]. The lack of appropriate representation of the activations in frequency domain has been the stumbling block for researchers working in this area [7].

In this paper, we propose a novel and adaptive design approach of ReLU activation function in frequency domain with consideration of both classification accuracy and hardware efficiency. This design maintains and approximates the non-linear performance of its spatial counterpart adaptively across different tasks. Combined with the spectral pooling in [4], a fully spectral CNN for image classification is designed, and the network structure is optimized by fusing and merging layers for model reduction. Then, we propose an efficient hardware accelerator to implement the proposed spectral CNN on FPGA, and DSP optimization for 8-bit multiplier is employed to further increase the speed of CNN inference. The main contributions of this work are:

1) An adaptive spectral ReLU activation design method, which improves the non-linear approximation of its spatial counterpart across image datasets while taking into account the efficiency of multiplications with complex numbers;

2) A fully spectral CNN approach with optimizations for model reduction in frequency domain, such as merging layers (batch normalization) and fusing layers (pooling);

3) The hardware architecture to accelerate fully spectral CNNs; when evaluated on Intel's Arria 10 SoC, it demonstrates up to 10× and 5.7× speedups compared to the spatial and FFT-based implementations on the same device, with comparable accuracy achieved across the benchmark datasets.

II. Background and Related Work

A. CNNs in Frequency Domain

CNNs are typically built of several computational operations stacked on top of each other, commonly known as layers. Frequently used layers are 2-D convolutional (Conv) layer, sub-sampling (pooling) layer or fully-connected (FC) layer. Besides, there are batch normalization layers and activations such as rectified linear unit (ReLU). Conv layer receives $C \times H_i \times W_i$ sized input feature maps, and then these inputs are convolved with a kernel having a shape of $F \times C \times K \times K$, and generates output feature maps with the size of $F \times H_o \times W_o$. On contrast, pooling layer only needs channel-wise operations, and element-wise operation is applied to ReLU layer. Therefore, the input and output feature maps of pooling and ReLU layers have the identical number of channel, i.e $F = C$.

978-3-9819263-5-4/DATE21/©2021 EDAA

FFT-based Convolution: Let X, Y and W represent the 2-D input, output and weight matrices respectively. Then Y is computed as the dot-product of the input feature maps and the corresponding weight kernel, i.e. space convolution. The powerful property of using Fourier transformation is that the heavy-duty convolution in spatial domain turns into an element-wise Hadamard product operation in the spectral domain, as shown in Equation (1).

$$\begin{aligned} Y &= \sum_{i=1} X_i \cdot W_{ij} = X \circledast W \\ &= \mathscr{F}^{-1}(\mathscr{F}(X) \odot \mathscr{F}(W)) \end{aligned} \tag{1}$$

where $\circledast$ denotes a convolution and $\odot$ denotes an element-wise product, $\mathscr{F}$ and $\mathscr{F}^{-1}$ denote Fourier transform and its inverse operation.

Other Layers: It is however challenging to compute the pooling and ReLU layers in frequency domain, since they do not have exact dual operations in spectral due to the non-linearity of these layers. Running FC layer in frequency domain leads to low compute efficiency because of the overhead of FFT and additional operations due to zero-padding when the input image size is very small (e.g. 1×1) [1]. As a result, FC layer is implemented in spatial domain [1]–[3].

B. Related Work

Computing Conv layers in the Fourier domain as point-wise products has been widely studied in machine learning community [1], [4], [5]. Mathieu *et al.* [1] introduced the use of FFTs to accelerate the training and inference of CNNs, with speed improvements of over an order of magnitude. Rippel *et al.* [4] proposed spectral pooling which performs dimensionality reduction by truncating the lower frequency representations. Francesca *et al.* [5] used Laplace transformation to implement the activation functions in frequency domain. However, Laplace transform has the numerical problem due to the exponential function in the Laplace inversion, especially when implemented in hardware with reduced arithmetic precision.

Many hardware designers [2], [3], [6], [7] have proposed FFT-based FPGA accelerators for inference and/or training of CNNs. Zhang *et al.* [2] exploited the Overlap-and-Add method to accelerate convolutional layers in frequency domain on FPGA. Then, Zeng *et al.* [3] improved this work by proposing a novel Concatenate-and-Pad technique. However, both work require Fourier transforms and the inverse at the boundary of every layer, thereby significantly limits the performance gain. Ko *et al.* [6] proposed a method to train the network entirely in the frequency domain. They used linear functions to approximate *tanh* and *sigmoid* activations. As a result, this method suffers from the lack of nonlinearity in the network, and it cannot apply to the commonly used ReLU function. Ayat *et al.* [7] proposed a spectral ReLU function to implement the fully spectral CNN. However, the design of their ReLU function depends on the evaluated dataset, and it is much more compute intensive than its spatial counterpart. In addition, the performance of their design is only evaluated in CPU instead of embedded devices or FPGAs.

III. Fully Spectral CNN with Adaptive ReLU

A. Adaptive Spectral ReLU Design

Rectified Linear Units (ReLU) are components of a network that map the input into a specific range of its output. ReLU function introduces non-linearity into the system. In this work, we consider the widely used ReLU function which is simply defined as a max operator:

$$F(x) = max(0, x) \tag{2}$$

It is straightforward to implement in the spatial domain with low computation cost. However, in the frequency domain, implementing such a non-linear function is rather difficult, since the Fourier transform can only be applied to linear operations.

As shown in Equation (1), convolution in spatial domain is equivalent to a point-wise product in the Fourier domain. Because of the duality property of the Fourier transform, Eq. (1) also works the other way around, i.e.

$$x \odot y = \mathscr{F}^{-1}(\, \mathscr{F}(x) \circledast \mathscr{F}(y)\,) \tag{3}$$

which means that convolution in spectral domain is equivalent to point-wise product in spatial domain. By substituting y with x, we have:

$$x^2 = \mathscr{F}^{-1}(\, \mathscr{F}(x) \circledast \mathscr{F}(x)\,) \tag{4}$$

Likewise, by convolving x with itself multiple times in the spectral domain, more non-linear functions $(x^2, x^3, \cdots, x^n)$ can be performed in the spatial domain:

$$x^n = \mathscr{F}^{-1}(\, \mathscr{F}(x) \circledast \mathscr{F}(x) \circledast \cdots \circledast \mathscr{F}(x)\,) \tag{5}$$

This allows for the possibility of building non-linear functions in the spectral domain using the basic building blocks as functions $(x, x^2, x^3, ...)$. Noting the linear property of Fourier transform, we can obtain a non-linear function $R(x)$ which can be computed in the spectral domain:

$$R(x) = p_0 + p_1 \cdot x + p_2 \cdot x^2 + \cdots + p_n \cdot x^n \tag{6}$$

This function can be used to approximate the max ReLU by carefully designing the coefficients p_i in each term. However, multiple convolutions are too computationally intensive, especially when complex numbers are used in spectral domain. Since batch normalization is always used before activation function which downscales the input into a specified boundary, we only need to approximate ReLU in the central region [6]. As such, only terms with low order in Equation (6) are required. Therefore, a quadratic function is used in this work to approximate the max ReLU, and it only requires one convolution in the frequency domain:

$$\mathscr{F}(R(x)) = p_0 + p_1 \cdot \mathscr{F}(x) + p_2 \cdot (\, \mathscr{F}(x) \circledast \mathscr{F}(x)\,) \tag{7}$$

Its backward propagation can be computed as below:

$$\frac{d}{dx} R(x) = p_1 + 2 \cdot p_2 \cdot x \tag{8}$$

The coefficient p_0 is the DC value of inputs and is set to zero to have the same behaviour of the original ReLU.

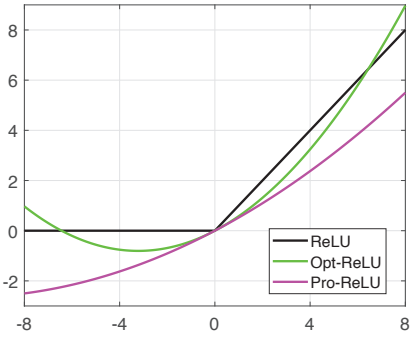

Fig. 1. Approximation of the spatial ReLU function in frequency domain using *optimal* and *hardware-friendly* (proposed) parameters.

In this work, the coefficients p_1 and p_2 are first obtained through the non-linear curve fitting tool, such as the Matlab Curve Fitting Toolbox. They are *optimal* parameters in terms of approximation quality. To improve the compute efficiency for hardware implementation, we propose the use of *hardware friendly* parameters in replace of the *optimal* parameters for approximation. The idea is to decompose the optimal coefficients by powers of two. In such a way, multiplications with p_1 or p_2 can be performed as shifting and adding, i.e.

$$p \cdot x = x >> i + x >> j \tag{9}$$

Usually, FPGAs have very limited DSP resource to implement multipliers. For example, an 8-bit complex multiplier needs two DSPs in Intel's Arria 10 device [8]. Nevertheless, operations of shifting and adding can be implemented using logic elements which is adequate in FPGA device. Figure 1 shows both approximations with *optimal* or *hardware-friendly* parameters to the original ReLU function in frequency domain.

In CNNs, the data range of inputs of ReLU layer can vary largely across layers or image datasets, as shown in Figure 2. As a result, a single spectral ReLU with fixed coefficients for every layer will have variations in approximation quality and thus lead to low classification accuracy. To address this problem, we propose to use an adaptive spectral ReLU with which the coefficients can be adaptively changed across layers based on the input range for different datasets. The information of the dynamic range of inputs is obtained from the training stage which isn't involved in the classification speed. Then the *hardware friendly* coefficients are decided by choosing the combination of shifting values (i, j) in Eq. (9). This can be implemented with a multiplexer for a number of shifting values of the input data, without any additional computational operations. Therefore, in our design both p_1 and p_2 are configured dynamically during runtime in order to adapt to the image datasets.

B. Spectral Pooling Function

The spectral pooling proposed by Rippel *et al.* [4] is used in our design. This function is to crop the frequency representation by only keeping a sub-matrix of frequencies that belongs to the lower frequency spectrum. The forward and backward propagation of this function is shown in Code 1. It

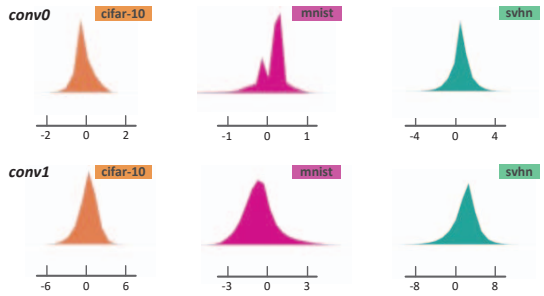

Fig. 2. The range of inputs of ReLU layers in LeNet-5 of different datasets.

has proven to be more beneficial than the spatial-domain max or average pooling, since it maintains more information and also allows any arbitrary output dimensionality.

Code 1 Spectral Pooling Function [4]

► **Forward propagation**:
Input: $X \in C^{M \times N}$
Output: $Y \in C^{H \times W}$
1: $Y \leftarrow \text{CROPSPECTRUM}(X, H \times W)$
► **Backward propagation:**
Input: $\frac{\partial R}{\partial Y} \in C^{H \times W}$
Output: $\frac{\partial R}{\partial X} \in C^{M \times N}$
1: $\frac{\partial R}{\partial X} \leftarrow \text{PADSPECTRUM}(\frac{\partial R}{\partial Y}, M \times N)$

C. Fully Spectral Network Design & Optimization

The initial LeNet-5 CNN architecture [9] consists of 7 layers, i.e. 3 Conv layers, 2 pooling layers and 2 FC layers. To improve the performance of classification accuracy, batch normalization (BN) and ReLU activation are applied to each convolutional layer. FC layer is memory bound and not suitable for spectral implementation as its input size is only 1×1. For this reason, the last two FC layers are merged into one layer with increased filters.

To save computational resources, batch normalization can be merged with the preceding convolutional layer. Let x be the data within the network to be normalized. BN computes x as follows:

$$\hat{x} = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{10}$$

where μ and σ^2 are the mean and variance computed over a batch, ϵ is a small constant included for numerical stability, γ is the scaling factor and β the shift factor. Let W_{conv} and b_{conv} denote the weight matrix and bias of the convolutional layer that precedes batch normalization. We merge these two layers by a single convolutional layer with the following parameters:

$$\begin{aligned} f &= \text{BN}(W_{conv} \cdot X + b_{conv}) = W_{new} \cdot X + b_{new} \\ W_{new} &= \frac{\gamma W_{conv}}{\sqrt{\sigma^2 + \epsilon}}, \; b_{new} = \gamma \frac{b_{conv} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \end{aligned} \tag{11}$$

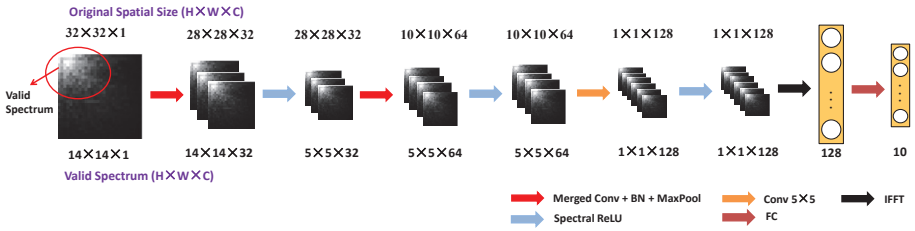

Fig. 3. Proposed fully spectral CNN architecture with model reduction.

Model Reduction: For spatial CNN, ReLU layer is performed right after the preceding convolutional layer, then followed by the pooling layer. Notice that the spectral pooling presented in Code 1 is merely cropping the sub-matrix of higher frequency spectrum. When executed in the order of CONV → RELU → MAXPOOL, many operations performed by the spectral ReLU layer are being wasted, i.e. cropped in the succeeding pooling layer, especially considering that now the spectral ReLU requires a convolution operation which has the complexity of $O(n^2)$ in complex numbers. This means a large number of computations are wasted to generate the unnecessary data which are going to be filtered out in the succeeding pooling layer.

To overcome this issue, we propose to apply the pooling layer before ReLU in spectral CNN model, i.e. in the order of CONV → MAXPOOL → RELU. We prove that this order has identical results as the spatial order. The following formula holds for any non-decreasing function f:

$$max(\, f(x_1), \cdots, f(x_n)\,) = f(\, max(x_1, \cdots, x_n)\,) \quad (12)$$

Remembering that the proposed spectral ReLU is an element-wise operation and a non-decreasing function, we have:

$$\text{MAXPOOL}(\text{RELU}(X)) = \text{RELU}(\text{MAXPOOL}(X)) \quad (13)$$

for any input X. Note that it does not work for average pooling, and it has negligible impact on the computation complexity of spatial CNNs. Nevertheless, in spectral domain, by placing ReLU after pooling, we can avoid the unnecessary Conv and ReLU operations which are performed to produce useless data, i.e. not used by the succeeding layer.

Besides, spectral pooling can be merged into Conv layers since it doesn't perform any mathematical operation in spectral domain. In this way, the ReLU layer only receives the cropped feature map which has a reduced size compared to the spatial size. For example, in MNIST dataset, the first ReLU layer has a valid input size of 14×14 and a valid output size of 5×5 (Figure 3), while the input and output size of spatial model are both 28×28. That means around $125\times$ computation reduction[1] for the first spectral ReLU layer.

With the above optimizations, the proposed spectral CNN architecture is shown in Figure 3. To get an impression on the benefit of our model reduction method, the original spatial size and the valid spectral size of MNIST in each computational layer are compared in the figure. The valid spectrum of the input image has been reduced from 32×32 to 14×14 which focus on the lower frequency representations. The reason to do that is that the power spectrum is heavily concentrated in the lower frequencies while higher frequencies tend to encode noise. As such, the elimination of higher frequencies has only minimal damage to the information in the input image and can even be viewed as a type of denoising [4].

Training and Inference: Since this work is focused on the inference stage, training is performed offline. To keep most of the backward propagation algorithm unchanged, training is mainly run in spatial domain, with the spatial ReLU suing forward propagation (Eq. 6) and backward propagation (Eq. 8). The only layer to run in spectral during training is spectral pooling, since it doesn't have a spatial counterpart. During inference, all computational layers except FC are executed in spectral domain with hardware acceleration, in order to achieve a high classification speed. FC layer is to run with spatial implementation in the host CPU. In the real-time inference stage, there are one domain conversion from spatial to spectral, i.e. FFTs of the input image, and one from spectral to spatial, which is the IFFT that succeeds the last Conv layer to generate the input of FC, as shown in Figure 3. Fourier transforms for the weight kernels of Conv can be prepared in advance and computed offline. As such, it eliminates the computation overhead of these repeated domain transformations, and also enables the simplicity and efficiency of the FFT-base Conv block in hardware without the need to consider the FFT size, compared to previous work of [2], [3].

IV. ACCELERATOR ARCHITECTURE DESIGN

A. Spectral Conv Module

The hardware architecture of the Spectral Conv module is presented in Figure 4(a). Since the spectral Conv only requires point-wise product, the implementation is quite simple which consists of a Multiply-and-Accumulate (MAC) unit for product and accumulation along channel dimension. The MAC unit needs to support complex numbers. The spectral Conv module also makes effective use of data parallelism by implementing an array of MAC units, in order to improve the throughput.

B. Adaptive ReLU Module

The architecture of the Spectral ReLU to compute Eq. (7) is presented in Figure 4(b). It consists of a dot-product block to perform convolution first. Then shifting and adding is followed to perform the multiplication with the coefficients for each term in Eq. (7). The coefficients are configured

[1] $(28^2 \times 28^2)/((14^2 \times 5^2)$

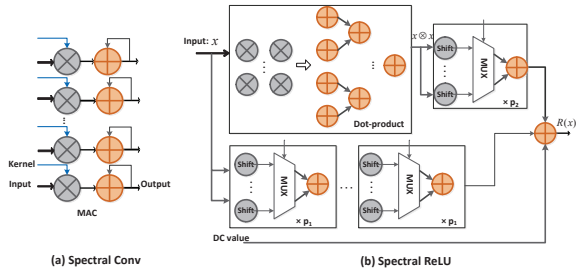

Fig. 4. Diagram of **(a)** spectral Conv and **(b)** spectral ReLU.

adaptively with controlled signals to the multiplexer to select the corresponding combination of the shifting values of the input data. This module also makes use of data parallelism to allow parallel processing.

C. Overall Architecture

Figure 5 shows the overall system design. Owning to the proposed fully spectral CNN, Fourier transforms are only required for input image and weight kernels. The transforms of weights are prepared and computed off-line, then stored in DDR memories in advance, to be cached in on-chip buffers through DMA during the execution of inference. This saves large computation cost and memory transfer overhead compared to the FFT-based approach in [2], [3]. The host CPU is responsible for FFT of the input image, IFFT of the results of the last Conv layer and the fully-connected layer. These light-weight operations are well suited for CPU and can be overlapped with the FPGA's execution time. The hardware accelerator on FPGA mainly consists of the spectral Conv and ReLU modules. Both computation modules receive inputs from on-chip buffers and their outputs are stored back to the buffers. Double buffer technique [10] with a flow control from the host is used to support inter-layer and intra-layer pipeline in order to keep both modules working concurrently, which overlaps the computation time between modules as much as possible and thus improves the compute efficiency.

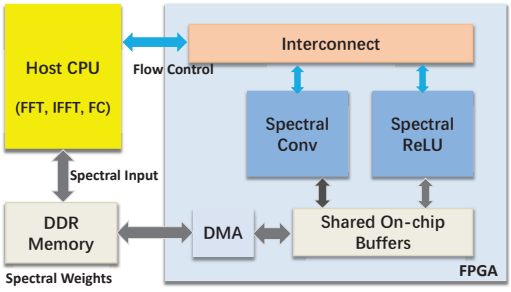

Fig. 5. The overall system architecture.

D. DSP Optimization for INT8 Complex Multiplier

The variable-precision digital signal processing (DSP) block in Intel Arria 10 devices includes two 18×19 multipliers with variable arithmetic precision support. Without any optimization, one complex multiplier with 8-bit fixed point (INT8) precision requires two DSP blocks [8]. One is responsible for the imaginary part and the other for the real part:

$$(a+jb)\times(c+jd) = [(a\times c)-(b\times d)]+j[(a\times d)+(b\times c)] \quad (14)$$

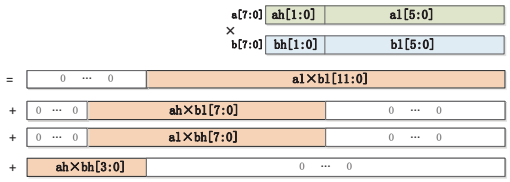

Fig. 6. The INT8 multiplication is decomposed into one 6-bit multiplication and other simple operations.

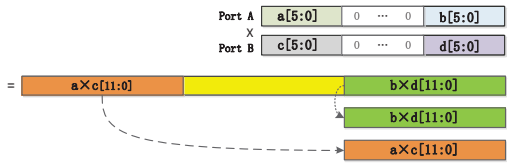

Fig. 7. The 18×19 multiplier in Arria 10 DSP block is optimized to implement two 6×6 multipliers, in order to improve the performance of INT8 multiplication.

In this work, we propose an INT8 optimization method, targeted at Intel's DSP block, which efficiently maps two 8-bit multiply into one 18×19 multiplier, i.e. 2 : 1 DSP multiplier to INT8 MAC ratio. Since the multiplier's input width is only 18 bits, we first separate the inputs ($a[7:0]$) into two parts: the higher 2-bit ($ah = a[7:6]$) and the lower 6-bit ($al = a[5:0]$). Then, the multiply $a\times b$ is decomposed into one 6-bit multiply, three multiply with very small input bits, and one addition of the four multiply results, as shown in Figure 6. Now we can pack 6-bit inputs a and b in the higher and lower 6-bit of the multiplier's 18-bit input port A, c and d in port B in the same manner, as shown in Figure 7. The 36-bit product result has $a\times c$ in higher 12-bit and $b\times d$ in lower 12-bit. As a result, two multiplication results can be separated from the 36-bit product, and the other three simple operations required to generate the 8-bit multiply result are implemented with logic resource.

V. Evaluation and Experiments

A. Benchmark Datasets

To verify and compare the accuracy and performance of the adaptive ReLU function to the fixed ReLU design, the proposed spectral CNN is applied to four image datasets: (1) MNIST, the handwritten digits; (2) SVHN, the street view house numbers; (3) AT&T, grey-scale face images and (4) CIFAR-10, color images of 10 objects. All datasets are trained using the proposed spectral CNN topology shown in Figure 3.

B. Implementation Detail

Our accelerator was implemented on Intel's Arria 10 SoC which contains an SX160 FPGA (20nm), 1.5 GHz dual-core ARM-based CPU and 2GB DDR4 memory. The CPU was responsible for one FFT, IFFT and FC operations. All other layers are run in the FPGA device. The hardware system was developed using Verilog HDL, and synthesized and placed-and-routed with Quartus Prime Pro 18.1. The designed accelerator achieved a working clock rate of 250 MHz.

C. Resource Utilization

Table I shows the resource utilization of our accelerator evaluated on the Arria 10 SX160 device. With the proposed

DSP optimization technique, we achieve a relatively high resource efficiency and compute density, as such the inference speed is largely improved with parallel processing.

TABLE I
RESOURCE UTILIZATION OF THE PROPOSED ACCELERATOR ON SX160

Resources	ALMs	Registers	DSPs	M20K
Used	43,466	95,834	140	360
Total	61,510	246,040	156	440
Utilization	70.7%	40%	90%	81.8%

D. Accuracy Comparison

We trained four versions of CNNs for comparison: the first one is the spatial CNN with original ReLU function, the other three are the spectral CNNs introduced above using the spectral pooling and ReLU with the fixed parameters, fixed and optimal parameters, adaptive and hardware-friendly parameters respectively. Table II shows the comparison of the classification accuracy of these designs on the benchmarks.

TABLE II
CLASSIFICATION ACCURACY (%) COMPARISON OF SPECTRAL CNNS.

	Spatial	Fixed-ReLU	Opt-ReLU	Adaptive-ReLU	diff.
MNIST	99.6	99.12	99	99.72	+0.6
SVHN	92.07	90.13	90.25	91.8	+1.67
AT&T	95	93.7	93.65	94.92	+1.22
CIFAR-10	79.4	75.2	75.41	78.56	+3.36

With the adaptive ReLU, the spectral CNNs achieve comparable accuracy to that of the original spatial version (Spatial), within only 0.8% accuracy loss across the datasets. However, the fixed ReLU design with either optimal or hardware friendly parameters leads to 4% accuracy loss on CIFAR-10. The accuracy difference of the adaptive and fixed ReLU designs is also listed in Table II, which confirms that the adaptive ReLU activation largely improves the accuracy of spectral CNNs.

E. Performance Comparison

The proposed hardware accelerator is implemented on Intel's Arria 10 device, together with two existing state-of-the-art implementations, i.e. spatial design and FFT-based design on the same device for performance comparison. The spatial design is built from our prior work in [10], which is used as a baseline in our experiment. The FFT-based hardware design is based on the work of [2].

Table III shows the performance of our accelerator against the other two methods (batch size = 32). The proposed design achieves 5.96× $\sim$ 9.97× speed improvement compared to the baseline and 4.0× $\sim$ 5.69× speedup compared to FFT-based design, while the FFT-based method only achieves around 1.5× speedup against the baseline. The results confirm that the computation overhead of Fourier transforms largely limits the benefit of computing Conv in frequency domain, while our method allows running CNN inference entirely in the frequency domain which avoids domain transforms. Besides, the adaptive design has the same speed as the fixed ReLU design, since it doesn't introduce any compute overhead when the ReLU is adaptively changed at runtime configuration.

TABLE III
CLASSIFICATION SPEED OF THE PROPOSED ACCELERATOR COMPARED TO THE STATE-OF-THE-ART SPATIAL AND FFT-BASED ACCELERATORS.

Methods	Datasets	time (ms)	Speedup
Spatial	MNIST/SVHN	2.70	-
	AT&T	7.38	
	CIFAR-10	3.10	
FFT-based	MNIST/SVHN	1.87	1.45×
	AT&T	4.21	1.75×
	CIFAR-10	2.07	1.50×
Fully spectral	MNIST/SVHN	0.45	6.0×
	AT&T	0.74	9.97×
	CIFAR-10	0.52	5.96×

VI. CONCLUSION

This paper proposes a fully spectral CNN approach to remove the repeated Fourier transforms when accelerating convolutions in frequency domain. This is achieved by proposing an adaptive and hardware-efficient spectral ReLU to approximate its spatial counterpart while keeping non-linearity in the network. Model reduction is performed in spectral domain by fusing and merging layers. The proposed hardware accelerator with DSP optimization shows significant speedups compared to the spatial and FFT-based implementations on the same device. Future work will apply the fully spectral approach to other CNNs, such as 3D CNNs for more complex tasks.

ACKNOWLEDGEMENT

The support of the National Natural Science Foundation of China (No. 62001165), United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1 and EP/L00058X/1), Corerain, Maxeler, Intel and Xilinx is gratefully acknowledged.

REFERENCES

[1] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," in *ICLR*, 2014.

[2] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on CPU-FPGA shared memory system," in *FPGA*, 2017, pp. 35–44.

[3] H. Zeng *et al.*, "A framework for generating high throughput CNN implementations on FPGAs," in *FPGA*, 2018, pp. 117–126.

[4] O. Rippel, J. Snoek, and R. P. Adams, "Spectral representations for convolutional neural networks," in *Advances in neural information processing systems*, 2015, pp. 2449–2457.

[5] M. Francesca, A. Hughes, and D. Gregg, "Spectral convolution networks," 2016. [Online]. Available: http://arxiv.org/abs/1611.05378

[6] J. H. Ko, B. Mudassar, T. Na, and S. Mukhopadhyay, "Design of an energy-efficient accelerator for training of convolutional neural networks using frequency-domain computation," in *DAC*, 2017, pp. 1–6.

[7] S. O. Ayat *et al.*, "Spectral-based convolutional neural network without multiple spatial-frequency domain switchings," *Neurocomputing*, vol. 364, pp. 152–167, 2019.

[8] Intel®, "Intel Stratix 10 Variable Precision DSP Blocks User Guide," 2020. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/documentation/kly1436148709581.html

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

[10] S. Liu and W. Luk, "Towards an Efficient Accelerator for DNN-Based Remote Sensing Image Segmentation on FPGAs," in *FPL*, 2019, pp. 187–193.