# Accelerating Convolutional Neural Network with FFT on Tiny Cores

Tahmid Abtahi, Amey Kulkarni, and Tinoosh Mohsenin
Department of Computer Science & Electrical Engineering
University of Maryland, Baltimore County

*Abstract*—**Fueled by ILSVRC and COCO competitions, Convolutional Neural Network (CNN) has become important in computer vision, and natural language processing. However state-of-the-art CNNs are computationally and memory intensive, thus energy efficient implementation on embedded platform is challenging. Recently VGGNet and ResNet showed that deep neural networks with more convolution layers (CV) and few fully connected layer (FC) can achieve lower error rates, thus reducing the complexity of convolution layers is of utmost importance. To reduce computations and shared memory usage in convolution layers, in this paper we evaluate the performance of direct convolution (Direct-Conv), Fast Fourier Transform (FFT) based convolution (FFT-Conv), and Overlap and Add FFT convolution (FFT-OVA-Conv) in embedded architecture including a low power domain specific many-core architecture called Power Efficient Nano Clusters (PENC) and ARM Cortex A53 CPU. To demonstrate the efficiency of FFT-Conv and FFT-OVA-Conv, we map ResNet-20 for the CIFAR-10 dataset on PENC as well as in ARM Cortex A53 CPU. Results are evaluated and compared with respect to throughput per watt, energy delay product, and execution time for three methods. Using built-in FFT instruction in PENC, the FFT-OVA-Conv performs 2.9× and 1.65× faster and achieves 6.7× and 2.3× better throughput per watt than Direct-Conv and FFT-Conv respectively. In ARM A53 CPU, the FFT-OVA-Conv achieves 3.36× and 1.38× improvement in execution time and 2.72× and 1.32× better throughput than Direct-Conv and FFT-Conv.**

***Keywords-*** Convolutional Neural Network, Domain-Specific Many-Core Accelerator, FFT Overlap and Add, Energy Efficient

## I. Introduction

Deep CNN has become increasingly important in computer vision [1], natural language processing [2], and speech recognition tasks [3]. Public Competitions for computer vision tasks such as ILSVRC and COCO have fueled tremendous growth in deep CNN architectures. Due to increasing Internet of Things (IoT) applications embedded devices need on-board processing, however computation intensive and memory exhaustive CNNs challenge current embedded platforms and tiny cores. Therefore, energy efficient and low latency acceleration [4] of CNN is of utmost importance.

CNN consists of a variety of layers such as convolution-1D, convolution-2D, fully connected, max-pooling, batch normalization, and Rectified Linear Unit (ReLU), in which convolution and fully connected layers are called as weighted layers. Convolution layers are computationally complex due to the sliding window and enormous amounts of multiplications, whereas fully connected layers are memory intensive. Fast Fourier Transform (FFT) based convolution (FFT-Conv) is widely used to reduce computation complexity of the convolution layer [5], however, FFT-Conv is only suitable when data and filter size are similar, additionally, it builds up additional intermediate memory to compute FFT coefficients. Thus FFT-Conv is not a satisfactory option on cache limited embedded processor. In this paper, we adopt over lap and add FFT (FFT-OVA-Conv), which overcomes intermediate memory buildup problem and therefore is suitable for disproportionate data and filter sizes. Furthermore, it exploits FFT's inherent reduced computational cost than Direct-Conv execution.

The key contributions to the paper are as follows:

- Computation and memory complexity analysis on Direct-Conv CNN, FFT-Conv CNN, and FFT-OVA-Conv CNN for AlexNet and ResNet-20 networks.
- Bit-accurate implementation of Direct-Conv, FFT-Conv, and FFT-OVA-Conv CNN for ResNet-20 with CIFAR-10 dataset on a domain specific low power PENC many-core and ARM A53 CPU platforms.
- Evaluate efficiency of three convolution methods on PENC many-core with respect to execution time, power, and throughput per watt with ResNet-20 network.
- Evaluate efficiency of three convolution methods on ARM A53 CPU with respect to execution time, and throughput with ResNet-20 network.

## II. Background: Convolution Neural Network

There exists a large variety of layers in CNN, however convolution and fully connected layers are highly utilized and largely contribute to the overall complexity. In Fully Connected (FC) layers, there exists a unique edge between the inputs (or prior layers outputs) and each of the neurons. Each neuron is, therefore, performing a dot-product of its inputs with a unique weight vector. The primary issue with fully-connected layers is that for high-dimensional data, the dense connectivity and large parameter set make it difficult to learn a meaningful transformation. Convolution (CV) layer can be visioned as an FC layer with added two constraints: the first is that neurons are connected to only a limited subset of inputs that are in a local neighborhood. For 1-D convolution layers, this corresponds to temporally close inputs and for 2-D convolution layers, this corresponds to spatially close inputs. The second constraint is that extensive weight sharing is enforced between neurons. These two constraints mathematically correspond to performing a series of convolution operations between the input and set of filters. A convolution layer typically consists of multiple filter banks which we refer to as feature maps. Each feature map is fed all of the input feature channels that contain temporal/spatial data and produces a corresponding output feature channel. This is achieved by convolving each input channel with a unique filter and summing across the convolved outputs to produce the output feature channel.

## III. Algorithm

### A. Direct Convolution (Direct-Conv)

Direct convolution involves multiplication addition (MAC) between a data patch and corresponding filter. Patches are constructed by striding along the data.

$$d(n) \star f(n) = \sum_{m=-\infty}^{\infty} d(m-n) \times f(m)\, dm \qquad (1)$$

For a two dimensional data with $n$ dimension and filter of size $k$ ($n > k$), number of patches can be calculated by $(n-k+1)/stride$. For

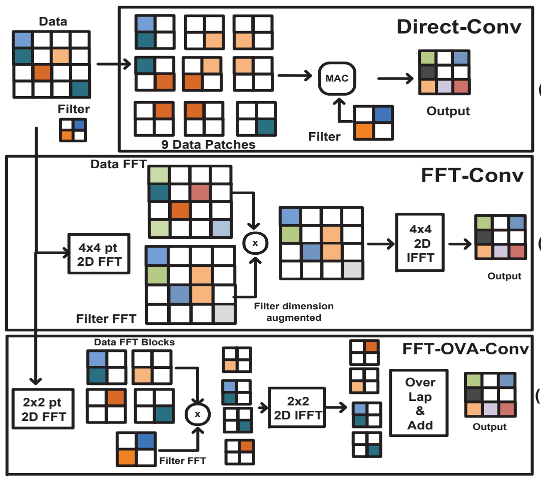


Fig. 1. Different convolution schemes (a) Direct-Conv with computational complexity of O($n^2k^2$), (b) FFT-Conv with computational complexity of O($n^2log(n)$) (c) FFT-OVA-Conv with O($n^2log(k)$) computational complexity where data dimension is n×n and filter dimension is k×k

multichannel data, MAC results between patches of data channels and filter channels are summed to obtain a pixel of the output. Fig. 1A shows convolution between a single channel 4x4 data and a 2x2 filter. 9 patches undergo MAC operation to obtain 9 pixels of the 3x3 output.

## B. *FFT Convolution (FFT-Conv)*

Convolution in time domain transforms into multiplication in frequency domain.

$$d(n) \star f(n) = \mathcal{F}^{-1}\Big\{ \mathcal{F}(d(n)) \times \mathcal{F}(f(n)) \Big\} \tag{2}$$

In this method, first both filter and data are transformed into frequency domain by FFT of the same length. Therefore filter FFT coefficients require additional intermediate memory. Then data FFT and filter FFT are element wise multiplied and fed to a Inverse FFT to obtain output. Fig. 1B shows filter dimension of 2x2 is augmented to 4x4 due to initial FFT transform.

## C. *OverLap and Add FFT Convolution (FFT-OVA-Conv)*

Overlap and add FFT method takes a portion of data and treats it as an independent input for convolution with the filter. This means that data (i.e $d(n)$) can be segmented into chunks of $d(n-kL)$, where $L$ is the length of each segment. FFT Convolution is performed on these segmented data and then block outputs are aligned and added as showed in equation 3.

$$d(n) \star f(n) = \sum_k \Big( \mathcal{F}^{-1}\Big\{ \mathcal{F}(d(n-kL)) \times \mathcal{F}(f(n)) \Big\}\Big) \tag{3}$$

Fig. 1C shows a 4x4 data to be segmented into 4 blocks of 2x2 size which corresponds to the filter dimension. Each block undergoes FFT-Conv in the next stage. No additional intermediate memory storage is created in this scheme since input to FFT are of same dimension. Outputs of FFT-Conv from 4 2x2 blocks are aligned to obtain the 3x3 output.

## D. *Computational Complexity and Memory Access Analysis*

In terms of computational complexity, Direct-Conv between n×n image and k×k filter kernel requires $(n-k+1)^2 \times k^2$ multiplication operation. The order of computation complexity being O ($n^2k^2$). Depending upon stride one memory location in image is accessed $k$ times and the memory storage requirements are $n^2 + k^2$. FFT-Conv requires $6Cn^2log(n) + 4n^2$ operations where C is a constant, computation complexity being O($n^2log(n)$). However, memory storage requirements are $2n^2$ since filter undergoes higher point FFT and require additional space. In FFT-Conv computational complexity is reduced to O($n^2log(k)$), and there is no additional memory storage requirement.

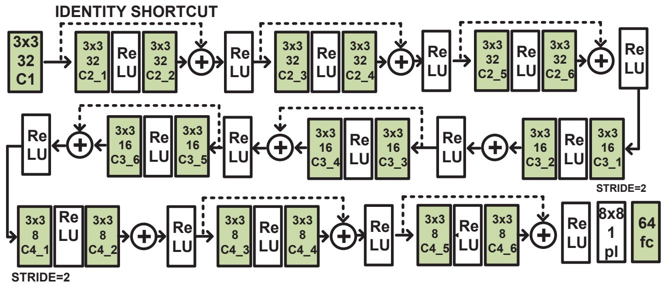


Fig. 2. ResNet-20 with CIFAR-10 data-set architecture, where C: convolutional layer with first row as filter size and second row as dimensions of output, ReLU: Rectified Linear Unit, pl: Global average pooling layer with first row as window size and second as output size, fc: fully connected layer

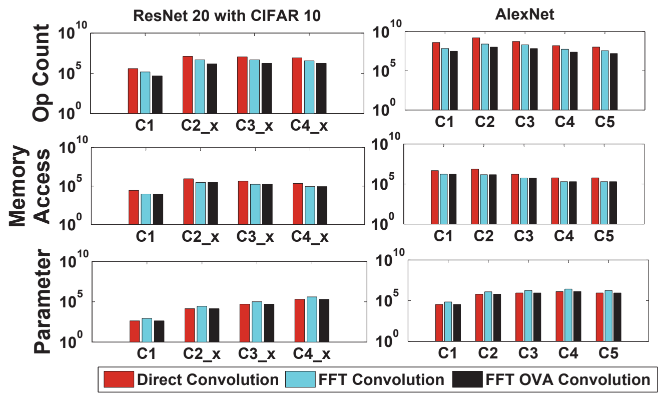


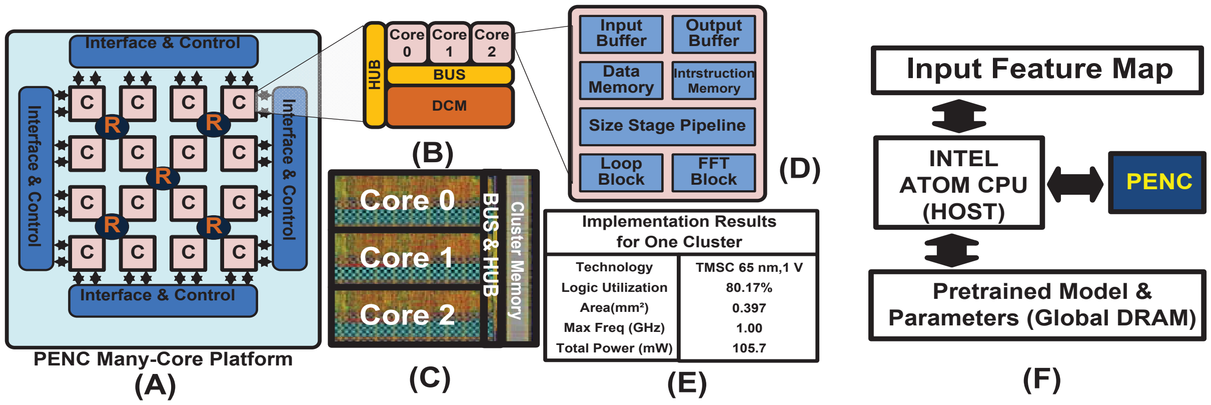
Fig. 3. Computational complexity and memory analysis between three methods of CNN convolution for ResNet-20 and AlexNet

We evaluate computational complexity and analyse memory requirements for ResNet-20 [6] architecture with CIFAR-10 and AlexNet [7]. CIFAR-10 data-set contains 50k training images and 10k testing images of 32×32 size. ResNet for CIFAR-10 can be represented by $6n+2$ stacked weighted layers with a global average pooling and softmax at the end. Only identity shortcuts were used in this network. We take $n = 3$ which results in ResNet-20 showed in Fig. 2. AlexNet consists of 5 sequential convolutional layers with three different filter sizes of 11×11, 5×5, 3×3. All filters in ResNet-20 are 3×3.

Fig. 3 shows, for both AlexNet and ResNet-20, FFT-OVA-Conv reduces computation 7× and 10× than Direct-Conv and 2.8× and 2.5× than FFT-Conv which is visually evident in initial layers where input image size is larger than filter kernel. In terms of memory access both types of FFT convolution are on average 3× better than Direct-Conv. Intermediate memory storage for FFT-Conv requires 2× more space than Direct-Conv and FFT-OVA-Conv.

## IV. PENC Many-Core & Evaluation Methodology

The PENC many-core architecture is composed of 64 processing clusters (192 cores) connected through routers in a three-level Globally Asynchronous Locally Synchronous (GALS) hierarchical tree [10], [11]. Each cluster consists of three processing tiny cores and distributed shared memory of 3072 words connected through



| Implementation Results for One Cluster | |
|---|---|
| Technology | TMSC 65 nm,1 V |
| Logic Utilization | 80.17% |
| Area(mm²) | 0.397 |
| Max Freq (GHz) | 1.00 |
| Total Power (mW) | 105.7 |

Fig. 4. (A) Power Efficient Nano Clusters (PENC), Many-core Architecture (B) Bus-based Cluster Architecture (C) Post-layout view of bus-based cluster implemented in 65nm, 1V TSMC CMOS technology (D) Block Diagram of core architecture with FFT block (E) Post Layout implementation results of optimized bus-based cluster (consisting of 3 cores + bus + cluster Memory) (F) Heterogeneous deployment of PENC with Intel Atom CPU (Host) where pretrained model and parameters (filter weights) are used for network inference

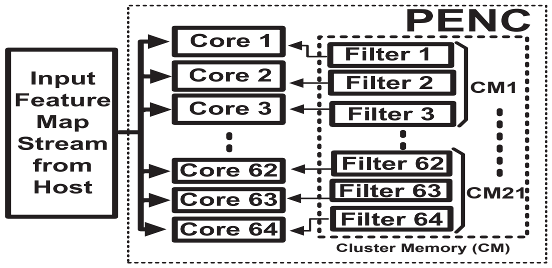


Fig. 5. Data flow between cores with host streaming input feature maps to cores with different filter weights saved in cluster memory

low latency bus architecture. Each processing tiny core has unique instructions, 16 quick-access registers, and 128 instruction and data memory. Fig 4 shows the block diagram of PENC many-core with the details of bus-based cluster and processor block diagram. It also summarizes post-layout implementation results on 65 nm, 1V TSMC CMOS technology. To evaluate PENC many-core we developed a stand-alone hardware simulator and compiler in Java. The simulator serves as a reference implementation of the architecture. Each task of algorithm is first implemented in assembly language on every processing core using many-core simulator. Binary files generated by many-core compiler are used to program each core individually. For execution time and energy consumption analysis of the algorithm, binaries obtained from many-core compiler are mapped on to VLSI hardware design of the many-core platform and simulated using Cadence NC-Verilog.

Each cluster consists of three tiny core with access to the 3072 words cluster memory via the bus. To access the memory, cores use two memory instructions: LD and ST. The LD and ST instructions enable the use of a much larger addressable space, which allows the PENC to support CNNs. PENC has 8 and 16 point built-in FFT instruction. The FFT instruction activates the hardware block that provides the FFT addresses for a complex radix-2 FFT. Upon the source and destination specifying, the address in memory containing the real and imaginary values for inputs are retrieved from the hardware block.

## V. Many-Core implementation result

ResNet-20 with CIFAR-10 is taken as a case study for evaluating convolution performance on PENC many-core. Fig. 4(F) shows the heterogeneous platform consisting of PENC as convolution accelerator and Intel Atom CPU as host performing task scheduling, data marshaling and post processing. Pre-trained model parameters are stored on global DRAM shared between host processor and PENC accelerator. Double buffering approach is adopted to hide data memory transfers between PENC and Atom. Inactive cores in PENC are shut down by power gating to reduce power consumption.

TABLE I
Comparison of ResNet-20 implementations for CIFAR-10 data-set on PENC

| Platform | Method | Execution Time (ms) | Energy (mJ) | Throughput Per Layer (MB/s) | EDP (mJ-s) |
|---|---|---|---|---|---|
| PENC (Semi-serial) | Direct-Conv | 66.5 | 38 | 5 | 2.5 |
| | FFT-Conv | 67.2 | 53 | 6 | 3.5 |
| | FFT-OVA-Conv | 31 | 28 | 18 | 0.9 |
| PENC (Most-parallel) | Direct-Conv | 36 | 65 | 10 | 2.3 |
| | FFT-Conv | 21 | 50 | 30 | 1 |
| | FFT-OVA-Conv | 12 | 31 | 60 | 0.4 |

Instruction level parallelism (ILP) is achieved by broadcasting input feature maps to active cores to perform convolution with different filter weights stored in corresponding cluster memories as showed in Fig. 5. Three different levels of parallelism are implemented based on storage and scope of ILP : semi-serial, semi-parallel and most-parallel. In semi-serial implementation, number of cores is assigned as per minimum number of cores required to store total layer filter weights, which is on average 15 cores per layer for ResNet-20. In most-parallel implementation, core number is assigned equal to the number of filters, thus achieving maximum ILP which results in 36 cores per layer on average. FFT based convolutions are implemented with PENC many-core FFT instruction. 3x3 Filters are zero padded to either 8 or 16 dimension as per size of the data. If data dimension is same or higher than 16 such as in $C1, C2-x, C3-x$ layers, 16 point FFT is used. Here, $C1$ represents first convolution layer, $C2-x$ second layer group with 32×32 data dimension and so forth. 2D FFT is performed by first finding 1D FFT of data in x direction followed by 1D FFT in y direction. Since intermediate memory is created in FFT-Conv after first FFT, we use cluster memory from a nearby idle cluster for additional storage. Therefore each cluster is surrounded by a storage cluster where cores are idle but cluster memory and corresponding routers are used. FFT-OVA used only 8-point FFT operation as data is segmented to 3×3 blocks.

FFT-OVA-Conv improves total run time by 2.9× and 1.65× than Direct-Conv and FFT-Conv. Fig. 6(A) shows the layer-wise run time breakdown for most parallel implementation. Since active clusters in FFT-Conv are surrounded by storage clusters for additional memory, data transmission cycles are longer and hence its run time exceeds both counterparts in final layers. Power increases in FFT-Conv by 1.4× than both counterparts due to additional router and storage

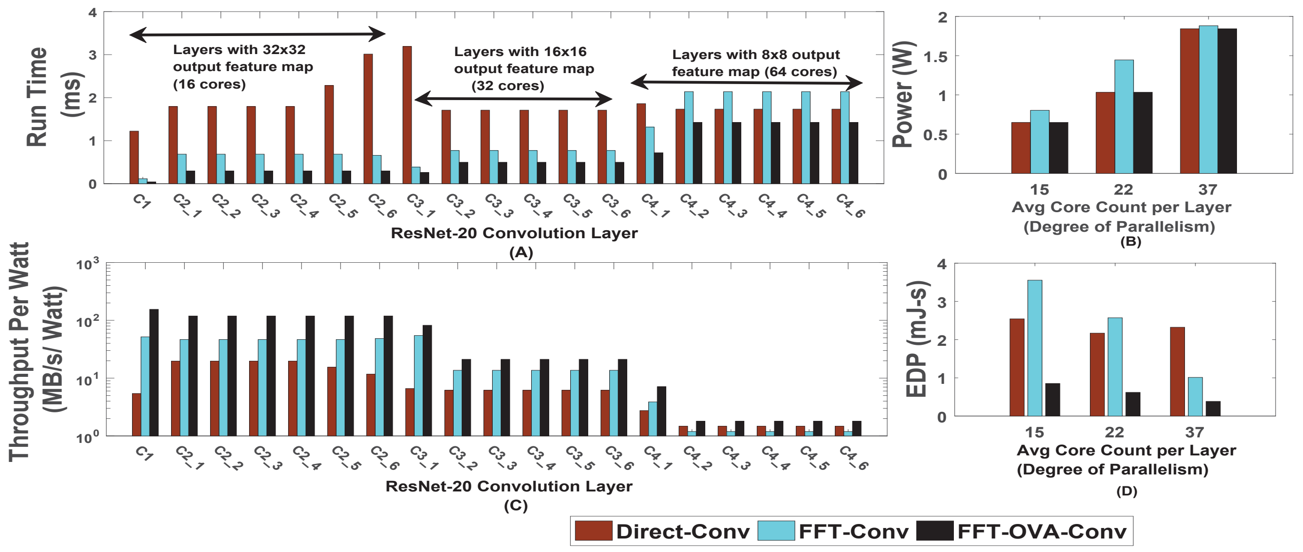


Fig. 6. (A) Layer wise run time breakdown for most-parallel implementation (B) Total Power (C) Layer wise throughput per Watt for most-parallel implementation (D) Energy Delay Product (EDP) for three different implementation: semi-serial implementation with 15 cores/layer, semi-parallel implementation with 22 cores/layer and most-parallel implementation with 37 cores/layer

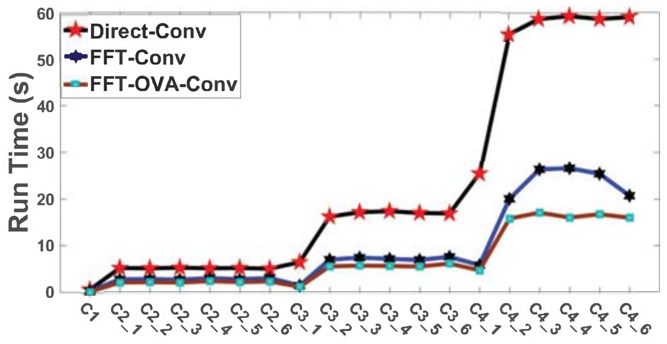


Fig. 7. Layer-wise run time for ARM Cortex-A53 CPU running at 1.2 GHz for serial implementation for different convolution methods

cluster memory power as showed in Fig. 6(B). Throughput per watt and Energy Delay Product (EDP) improve in FFT-OVA-Conv by 6.74×, 2.3× and 6×, 2.6× respectively than Direct-Conv and FFT-Conv.

## VI. CPU implementation result

We implemented ResNet-20 on low power ARM Cortex-A53 CPU running at 1.2 GHz frequency residing on Raspberry Pi 3-B using existing software libraries. GNU Octave high level programming language is used to run ResNet-20 model with user defined serial functions for convolutions, ReLU and average pooling operation. Timing results is obtained from Octave's time measurement function. Fig. 7 shows the layer-wise timing breakdown for three different convolution methods. FFT-OVA-Conv achieves 3.36× and 1.38× improvement in execution time and 2.72× and 1.32× better throughput than Direct-Conv and FFT-Conv.

## VII. Conclusion

Direct-Conv, FFT-Conv and FFT-OVA-Conv methods for CNN are explored on ARM CPU and a domain specific many-core architecture PENC using built-in FFT instruction for ResNet-20 with CIFAR 10 data-set with a variety of parallel mappings. FFT-OVA-Conv has an execution time of 12.4 ms and average per layer throughput of 60.5 MB/s with 1.84 W average per layer power consumption and is 2.9× and 1.65× faster and achieves 6.7× and 2.3× better throughput per watt than Direct-Conv and FFT-Conv respectively. In ARM A53 CPU, the FFT-OVA-Conv shows 3.36× and 1.38× improvement in execution time and achieves 2.72× and 1.32× better throughput than Direct-Conv and FFT-Conv.

## References

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

[2] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[3] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[4] A. Page, A. Jafari, C. Shea, and T. Mohsenin, "Sparcnet: A hardware accelerator for efficient deployment of sparse convolutional networks," *Journal on Emerging Technologies in Computing (JETC)*, p. 10, 2017.

[5] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

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

[8] A. Kulkarni and T. Mohsenin, "Low overhead architectures for omp compressive sensing reconstruction algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–13, 2017.

[9] A. Page and T. Mohsenin, "Fpga-based reduction techniques for efficient deep neural network deployment," in *Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on*, 2016, pp. 1–8.

[10] A. Page, N. Attaran, C. Shea, H. Homayoun, and T. Mohsenin, "Low-power manycore accelerator for personalized biomedical applications," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '16. ACM, 2016, pp. 63–68.

[11] A. Kulkarni, T. Abtahi, E. Smith, and T. Mohsenin, "Low energy sketching engines on many-core platform for big data acceleration," in *Proceedings of the 26th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '16. ACM, 2016, pp. 57–62.