CNNs on FPGAs for Track Reconstruction

Thomas Boser[a], Jason Nielsen[a], Ian Johnson[b], Paolo Calafiura[b]

[a] *University of California, Santa Cruz*
[b] *Lawrence Berkeley National Laboratory*

Abstract

With recent increases in the luminosity of High Luminosity Large Hadron Collider (HL-LHC) collisions creating more tracking data, an efficient track reconstruction solution has become necessary. As it currently stands, during the level 1 trigger, it is necessary to identify 50 million particle tracks per second with latency lower than 10μs per track. Current algorithms are implemented on ASIC chips or FPGAs and scale O(N^2)or worse. Thus the track trigger system at HL-LHC will have to process O(10x) more tracks than they currently do while providing the same efficiency and purity and operating with the same bandwidth and latency requirements. Simultaneously, deep learning has become a standard technique in computer vision. The HEP.TrkkX project[1], among others, is exploring the applicability of Deep Neural Networks (DNNs) to the tracking problem. Current popular deep learning libraries are all heavily reliant on Graphics Processing Units (GPUs) to shoulder the bulk of heavy computation. Unfortunately, current and projected GPU architectures do not meet the stringent latency requirements (10μs) of LHC L1 triggers. To address this problem, we have started investigating the feasibility and performance of DNN inference implemented on FPGAs. We demonstrated a pipelined CNN inference in firmware which

can be scaled to maximize FPGA resource usage, along with an OpenCL implementation of the same network. We found that Digital Signal Processors (DSPs), a resource on the FPGA which can be used as a multiplier, are a limiting resource. Our firmware implementation allows larger FPGAs with more DSPs available to maximize resource usage in order to further parallelize individual convolution and dense layers. We implemented and benchmarked LeNet[2] on an Altera Cyclone V, a small FPGA with 112 DSPs, as compared to over 5000 in high end FPGAs. Finally, we analyzed the feasibility of DNN architectures used for tracking with very stringent latency requirements.
Keywords: Deep Learning, Tracking, CNNs, FPGAs, Physics

1. Introduction

The Large Hadron Collider is the world's most powerful particle accelerator[3]. Its purpose is to cause protons to collide at near-light speed in order to study theories in particle physics. The collision of two protons causes many smaller particles to scatter from the point of collision. A single instance of two protons colliding is called an *event*. Large detectors are used to capture the trajectories of particles produced by the collision. When a particle is detected a *hit* is produced in the form of the coordinates where the particle was detected. Figure 1 depicts a simplified

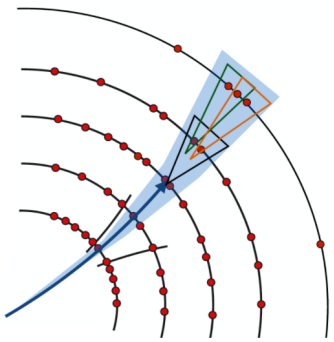

Figure 1: A simplified depiction of an event. Points represent hits, circles represent detectors[4].

two dimensional view of what an event looks like. Because of the extremely high speeds particles travel, timing information is unavailable, so a particle's trajectory must be inferred given only a set of hits.

1.1. The Tracking Problem

The first step in analyzing data produced by LHC experiments, such as the ATLAS[5] experiment, is to infer *tracks* from the set of hits output by the detectors. A track is the set of hits which belong to a single particle, meaning the trajectory of this particle. Because the detectors can only produce a set of hits as output, accurately producing tracks is not a trivial task. This initial step is a component of the Level 1 Trigger (L1), a step in which events only events which have "interesting" particles to study are kept. The L1 trigger must occur in real time because upwards of 100MB of data is produced per 100 nanoseconds, which would be too costly to store[6].

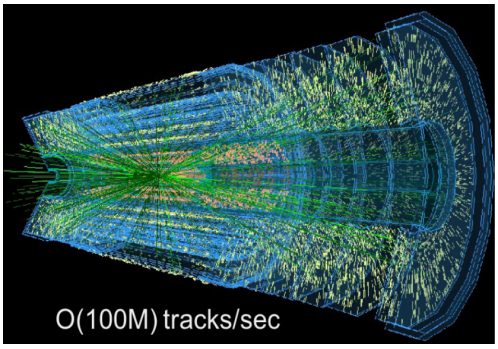

Figure 2: A realistic depiction of an event with hits scattered in three dimensions[4].

Current algorithmic implementations solving the tracking problem divide it into four steps: hit clustering, track seed finding, track building, and track fitting[1]. Although the algorithms currently in use are sufficiently accurate, the need for improved efficiency demands a solution which will scale with increased event luminosity. Because datasets are complex, as is demonstrated in Figure 2, finding such a solution has not

been trivial. We estimate that events will need to be processed with sub 10μs latency[7]. Because DNNs are being explored as a possible solution to the tracking problem we will examine whether achieving sub 10μs latency is possible with DNNs.

1.2. Low Latency CNNs

GPUs have become a standard tool for working with DNNs because of their ability to perform massive amounts of floating point operations. While the throughput achievable using GPUs is extremely high and continues increasing every year[8], extremely low latency inference can not be achieved using GPUs. Data batching is an efficient solution to maximizing throughput and GPU usage and has become necessary in order to make use of massively powerful GPUs. Unfortunately, this solution is very unsuitable to highly latency sensitive applications, which would need to make use of small batches[9]. We have found that GPUs provide a highly parallel approach to floating point multiplication which enables high throughput applications to strive but does not support extremely low latency implantations. FPGAs have been explored as a low power CNN accelerator, but no published solution has explored extremely low latency inference. Work by Qui et al. compares multiple CNN accelerators, all of which have latencies in the millisecond range, multiple orders of magnitude off of our goal[10].

2. Background

2.1. Convolutional Neural Networks and Deep Learning

Deep learning[11] is a class of "black box" machine learning algorithms which allow for learning to be done on data with multiple levels of abstraction.

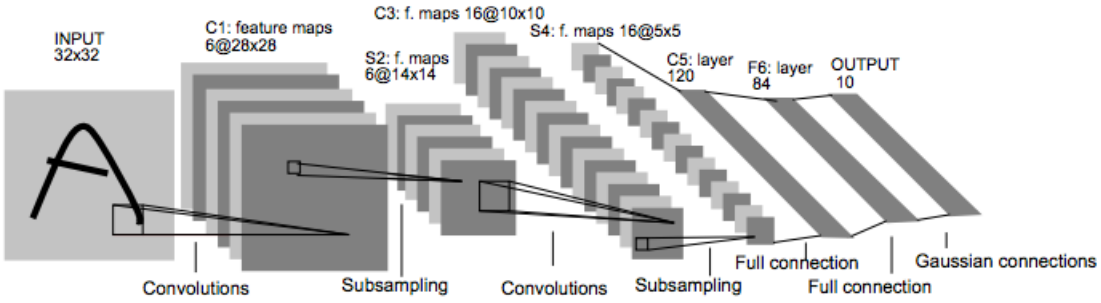

Figure 3: A visual representation of the LeNet V architecture[2].

Deep learning algorithms excel because rather than learn from predefined features, they find the features which most heavily impact predictions. Our work focuses specifically on Convolutional Neural Networks, a deep learning algorithm that is most commonly applied to image classification problems. CNNs are feed forward networks, meaning multiple transformations are applied to inputs one after another (transformation steps are often called layers). Training is done backwards using backpropagation, and can be treated as a one time cost so does not have a latency requirement for our purposes.

In order to demonstrate the functionality of our implementation, we implemented LeNet V, a classical CNN architecture that is still often used today. Figure 3 shows a diagram of LeNet V with each layer labeled. The functionalities of each layer and component are as follows, with each bold element signifying a layer we have implemented in firmware:

- Input: CNNs take matrices of varying dimension as input. In the case of the original implementation of LeNet V the input was a 32x32 grayscale image of a digit or letter.

- **Convolutions**: Convolutions are the foundation of CNNs. In the case of CNNs during a convolutional layer, a filter matrix is convolved over the input matrix. The filter matrix is a weights matrix of varying dimensionality which is updated during backpropagation. In the graphic, filters are labeled as "feature maps".

- **Subsampling**: Also known as downsampling and pooling. Pooling combines the output of multiple neurons, reducing the size of input of the next layer. Max pooling is most commonly used where only the maximum value from a set of neurons is selected.

- **Full connection**: Also known as a dense layer, these layers are called fully connected because neurons in a fully connected layer are connected to each previous activation. This means that rather than filters convolving over the input, dense layers simply consist of a single matrix multiplication.

- **Activations**: These aren't visible in the LeNet V diagram but are a key component of CNNs. Activations provide a nonlinearity which are often found after most convolution layers. An example of this is the rectified linear activation where negative values are set to 0 and positive values are unchanged.

2.2. Field Programmable Gate Arrays

FPGAs are re-programmable integrated circuits consisting of logic blocks which can be configured to perform low level operations such as bit masking, shifting, and addition. This allows FPGAs emulate the functionality of

Application Specific Integrated Circuits (ASICs) - fabricated chips - while maintaining flexibility and significantly lower cost of development.

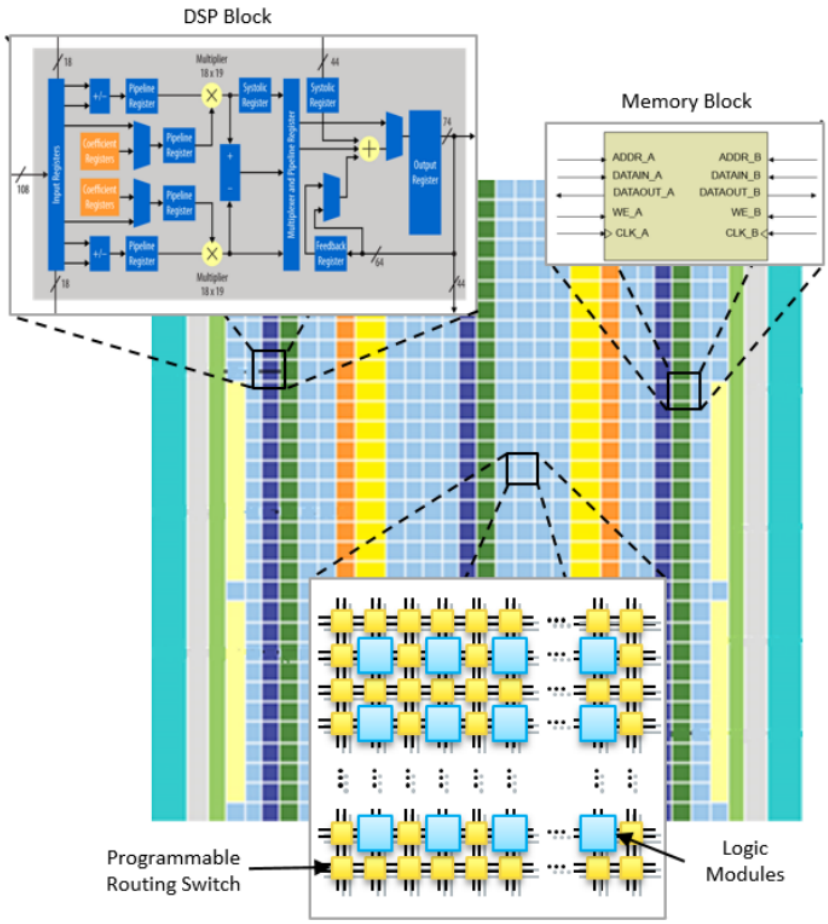

Figure 4: A diagram showing the logic layout of an Altera FPGA along with other components present on their FPGAs.[12].

Figure 4 shows the general architecture of Altera FPGAs. Our work uses a Terasic DE10-Standard SoC which houses an Altera Cyclone V FPGA. The figure shows that an FPGA primarily consists of logic modules and routing switches, which are the FPGA's re-programmable logic. In addition

to the standard logic modules, Altera FPGAs have memory blocks and DSP Blocks. Memory blocks can be used to store information which cannot be encoded in the logic modules. In order to store the trained CNN on the FPGA, memory blocks must be used. Frequent memory accesses can cause the FPGA to run at a lower clock frequency because of the time required for memory to travel through the FPGA, and preemptive fetches may be necessary in a pipelined solution. Digital Signal Processor (DSP) blocks can serve as multipliers on the FPGA. Multiplication is a costly operation when implemented using logic modules, making it most effective to use DSPs as multipliers and use logic modules for other operations. Altera DSPs can perform two multiplications per clock cycle (because they are pipelined), so with a fully pipelined implementation you can perform 2*number of DSPs multiplications per clock cycle[13].

Programming FPGAs is classically done using a hardware description language (HDL), a programming language which describes electronic circuits. This gives programmers the ability to fully control the FPGA's layout in the same way possible when designing an ASIC. We demonstrate our implementation using VHDL, a popular HDL. FPGAs can also be programmed using High Level Synthesis (HLS), a practice in which HDL code is synthesized from a higher level language such

Figure 5: In order to use OpenCL on an FPGA a CPU co-processor is necessary.

as C or C++. Finally, a more modern option for programming FPGAs is OpenCL[14], a computing framework which allows for significantly higher level programming of FPGAs. Figure 5 shows how OpenCL is used in order to program FPGAs. OpenCL requires a CPU co-processor to handle data transfer to the FPGA, as well as send the FPGA instructions. This is done using an FPGA-specific API provided by vendors. In addition to our firmware solution, we have created an implementation of LeNet V using OpenCL.

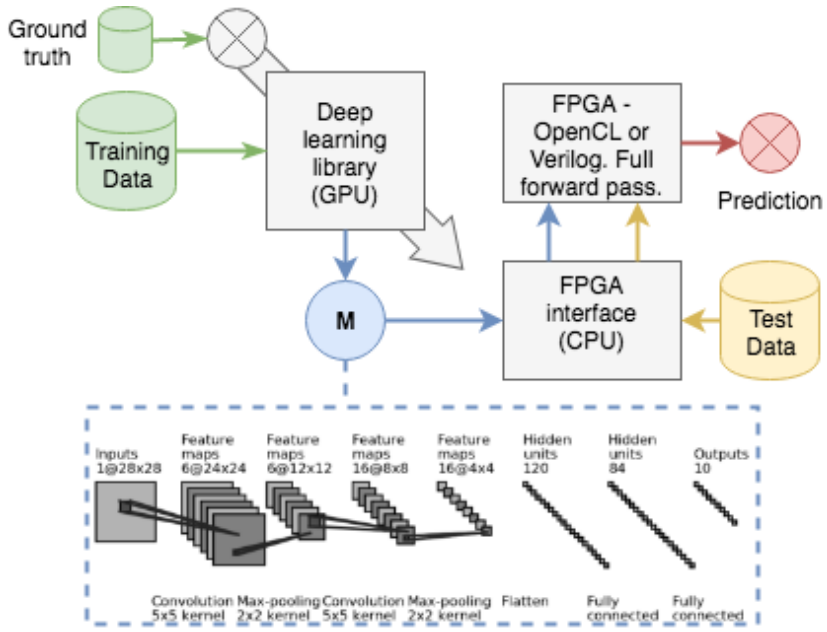

Figure 6: A diagram describing the process used to perform inference on an FPGA.

3. Implementation Specifics

3.1. Complete Workflow

Because our motivation was to perform inference with low latency, we chose to train our networks on GPUs and then transfer the weights to our FPGA. There are published implementations of backpropagation on FPGAs[16], which focus on minimizing the power consumption of training. Figure 6 shows the high level process we used in our implementation. We developed and trained our CNN model using Keras[15], a Python deep learning library. The trained model then had to be prepared for use in the FPGA. Because floating point operations are significantly costlier than integer operations we modified our weights to be integers. After completing these steps, we mapped the weights to predefined memory locations on the FPGA and were ready to perform inference.

3.2. Integer Weights

Model	Bit-width	Top-1 error	Top-5 error
ResNet-18 ref	32	31.73%	11.31%
INQ	5	31.02%	10.90%
INQ	4	31.11%	10.99%
INQ	3	31.92%	11.64%
INQ	2 (ternary)	33.98%	12.87%

Figure 7: Significant decreases in weight precision does not necessarily reduce accuracy[17].

Floating point multiplication is the primary cost of a CNN forward pass, the process going from an input an output, which has made the use of GPUs a near requirement for deep learning. In order to take advantage of FPGAs, we used 8 and 16 bit integer weights rather than the standard 32 bit float. This significantly reduced the cost of multiplication and meant we were able to perform two multiplications simultaneously using a single

DSP. A large variety of techniques for reducing weight precision have been studied in recent years, many of which show lower precision weights will often perform as well, or sometimes even better, than their high precision counterparts[17]. This can be seen in Figure 7.

3.3. Firmware Implementation

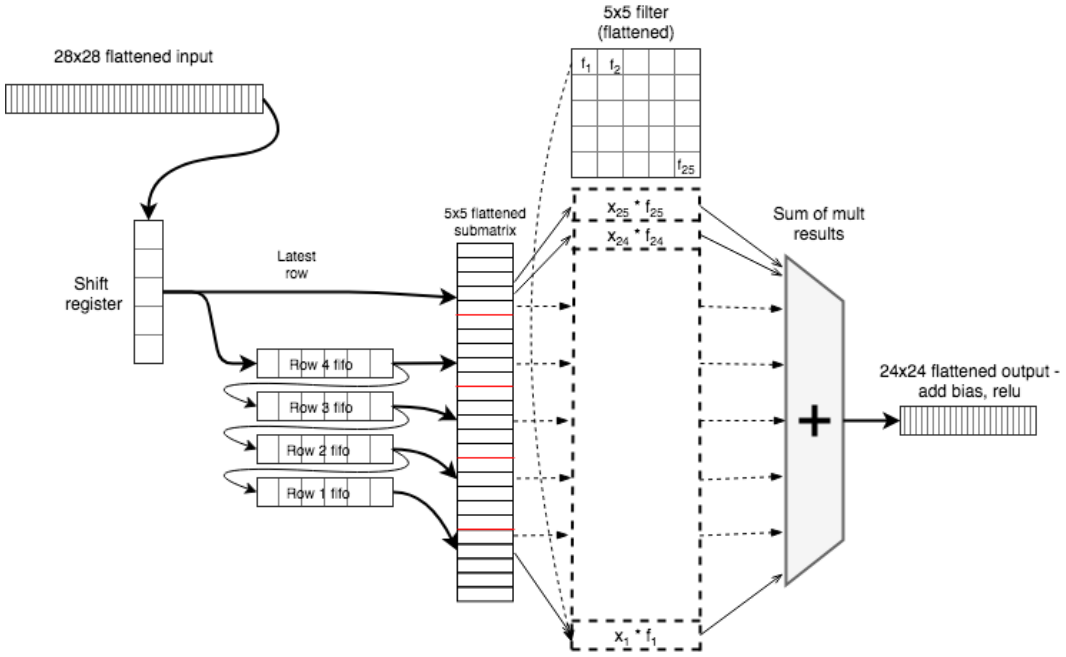

Figure 8: A diagram showing the firmware implementation of the first convolution layer in LeNet V.

In order to achieve a low latency solution without sacrificing throughput, we developed a fully pipelined CNN forward pass in firmware. Our implementation consists of VHDL modules dedicated to convolution, pooling, and dense layers. Figure 8 describes the algorithm which allows us to pipeline and parallelize a single convolution layer. Data is input as a flattened matrix and

then gradually enters a shift register which offloads new values to a FIFO and old values to a submatrix. Each FIFO is the size of a row in a filter matrix so that when values are pushed onto the flattened submatrix they properly match the values which are multiplied during the convolution. The results of the multiplications are summed, after which we add bias and use our activation function. This implementation is fully pipelined because it allows for a new input to be streamed in without any delay and without interfering with the previous input. This process can be sped up by increasing the number of cells entering the shift register per clock cycle. Parallelization is possible in multiple locations. The most obvious way is by performing convolutions for separate filters in parallel (with the same input). Parallelization can also be achieved by allocation more DSPs to the multiplication of the 5x5 submatrix with the 5x5 filter, a step which can have it's latency reduced to 1 clock cycle. This implementation allows us to allocate resources efficiently;

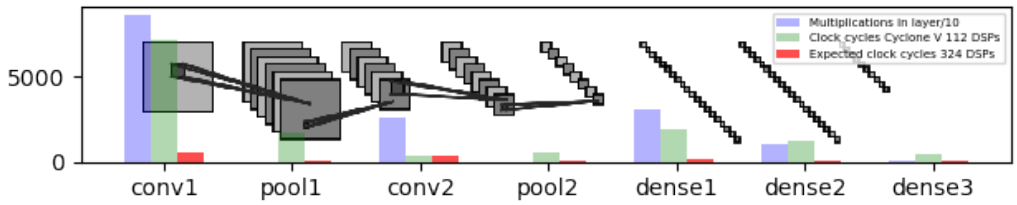

Figure 9: A small increase in the resources available at each layer can significantly reduce the number of clock cycles. Red bars show the number of clock cycles per layer with 324 DSPs available for the full implementation and green the number of clock cycles with 112 DSPs.

for example the multiplication of the submatrix with the filter takes 25 clock cycles when a single multiplier is allocated to the step. If just one more multiplier is allocated to the step it only takes 13 clock cycles, nearly dou-

bling the speed. Unfortunately, there are diminishing returns and it would take two more multipliers to double the speed again. Figure 9 shows how significant a small increase in DSP availability can be on the latency of an implementation. This knowledge allows us to allocate DSPs to steps which benefit most from parallelization.

3.4. Resource Consumption

Because of the large number of multiplications in a CNN forward pass, smaller FPGAs will quickly be resource starved. We demonstrated our implementation using an Altera Cyclone V, a small FPGA, requiring a resource conscious approach. Figure 10 shows a comparison in resource usage of our OpenCL implementation and our VHDL implementation of LeNet V. It is worth noting that neither implementation was fully optimized and that a lack of resources on the Cyclone V significantly reduced the performance of our implementation. We show the large disparity in DSP availability between a Cyclone V and Stratix 10, a resource which was fully used in the VHDL implementation. It is also interesting to note that our OpenCL implementation used roughly a third of the available DSPs, which may be a contributing factor to the higher latency.

	DSPs	Memory (Block RAMs)	Clock cycles
OpenCL LeNet	35	273	1176k
VHDL LeNet (resource conscious)*	112	300	50k
Available (Cyclone V)[1]	112	557	--
Available (Stratix 10)[2]	5,760	11,721	--

Figure 10: FPGA resource availability. Resource conscious LeNet assumes Cyclone V resources are available. Available resources from data sheet[18].

4. Discussion

A deep learning solution to the tracking problem has not yet been accepted, but our work shows extremely low latency CNN inference is possible using FPGAs. We can roughly predict the latency of inference for a CNN using our implementation given the number of DSPs available and the number of multiplications present in the network. Other factors impact the latency, but because DSPs will usually be the limiting factor, we can use them to predict latency. Figure 11 shows an exploration of CNN design space, which meets the latency restrictions of the L1 trigger. At 400MHz, 10μs is 4000 clock cycles. Therefore under optimal conditions an FPGA with 5000 DSPs will be able to perform a maximum of 40,000,000 multiplications in the 10μs. Our LeNet implementation only has 43,000 multiplications, but larger networks often exceed hundreds of millions of multiplications. While technically this can be parallelized over multiple FPGAs, this would be tremendously costly, so now we must create DNN that solves

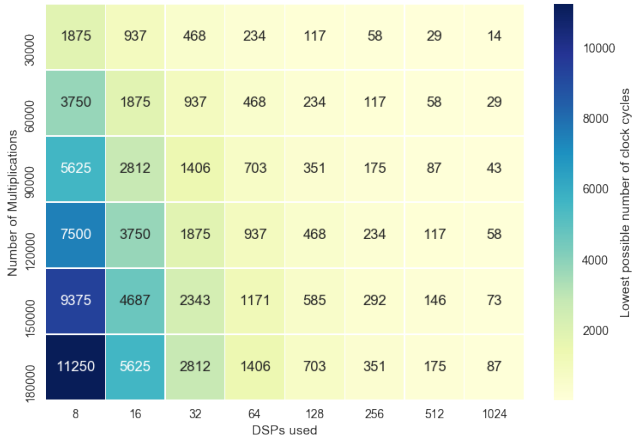

Figure 11: A heatmap showing how the number of DSPs allocated to convolution layers impacts the number of clock cycles for convolutions of different sizes.

the tracking problem and fits within our design space.

5. Conclusion and Future Work

We demonstrated a firmware implementation of a CNN on an FPGA which gives us predictable and constant real-time latency. We showed that our implementation can support the low latencies required at the LHC with restrictions on the size of the CNN. Our implementation can function without a CPU co-processor so that data can be streamed directly into the FPGA minimizing data transfer latencies. Our CNN implementation shows the potential for low latency implementations of other DNNs because of the parallel nature of DNNs, and the similarity between CNNs and other DNNs.

In the future, we plan to more thoroughly benchmark our implementation by streaming data directly to the FPGA, as well as implementing larger CNNs on the scale of those which may solve the tracking problem. We also plan to explore alternative DNNs as a solution to the tracking problem and plan on implementing new layers in firmware in order to support different DNNs.

References

[1] Farrell, Steven, Dustin Anderson, Paolo Calafiura, Giuseppe Cerati, Lindsey Gray, Jim Kowalkowski, Mayur Mudigonda et al. "The HEP. TrkX Project: deep neural networks for HL-LHC online and offline tracking." In EPJ Web of Conferences, vol. 150, p. 00003. EDP Sciences, 2017.

[2] LeCun, Yann, Lon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86, no. 11 (1998): 2278-2324.

[3] Evans, Lyndon, and Philip Bryant. 2008. "LHC Machine". Journal Of Instrumentation 3 (08): S08001-S08001. doi:10.1088/1748-0221/3/08/s08001.

[4] Credit to Andy Salzburger.

[5] Aad, Georges, E. Abat, J. Abdallah, A. A. Abdelalim, A. Abdesselam, O. Abdinov, B. A. Abi et al. "The ATLAS experiment at the CERN large hadron collider." Journal of instrumentation 3, no. 8 (2008): S08003-S08003. Harvard

[6] Amstutz, Christian, F. A. Ball, M. N. Balzer, J. Brooke, L. Calligaris, D. Cieri, E. J. Clement et al. "An FPGA-based track finder for the L1 trigger of the CMS experiment at the high luminosity LHC." In Real Time Conference (RT), 2016 IEEE-NPSS, pp. 1-9. IEEE, 2016.

[7] Rossi, Lucio. LHC upgrade plans: Options and strategy. No. CERN-ATS-2011-257. 2011.

[8] Owens, John D., Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. "GPU computing." Proceedings of the IEEE 96, no. 5 (2008): 879-899.

[9] Chen, Yifeng, Xiang Cui, and Hong Mei. "Large-scale FFT on GPU clusters." In Proceedings of the 24th ACM International Conference on Supercomputing, pp. 315-324. ACM, 2010.

[10] Qiu, Jiantao, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu et al. "Going deeper with embedded fpga platform for convolutional neural network." In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 26-35. ACM, 2016.

[11] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521, no. 7553 (2015): 436. Harvard

[12] Altera, S. D. K. "for OpenCL Programming Guide." (2014).

[13] Altera Cyclone, I. I. "Device Handbook." (2013).

[14] Stone, John E., David Gohara, and Guochun Shi. "OpenCL: A parallel programming standard for heterogeneous computing systems." Computing in science and engineering 12, no. 3 (2010): 66-73.

[15] Chollet, Franois. "Keras." (2015): 128.

[16] Liu, Zhiqiang, Yong Dou, Jingfei Jiang, Qiang Wang, and Paul Chow. "An FPGA-based Processor for Training Convolutional Neural Networks." 2017 International Conference on Field Programmable Technology (ICFPT), 2017. doi:10.1109/fpt.2017.8280142.

[17] Zhou, Aojun, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. "Incremental network quantization: Towards lossless cnns with low-precision weights." arXiv preprint arXiv:1702.03044 (2017).

[18] Altera datasheets:
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/pt/cyclone-v-product-table.pdf
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/pt/stratix-10-product-table.pdf