# Low-latency machine learning inference on FPGAs

**Javier Duarte**
University of California San Diego
Fermi National Accelerator Laboratory

**Christian Herwig, Burt Holzman, Sergo Jindariani, Benjamin Kreis, Mia Liu, Ryan Rivera, Nhan Tran**
Fermi National Accelerator Laboratory

**Song Han, Phil Harris, Dylan Rankin**
MIT

**Vladimir Loncar, Jennifer Ngadiuba, Maurizio Pierini, Sioni Summers**
CERN

**Scott Hauck, Shih-Chieh Hsu**
University of Washington

**Zhenbin Wu**
University of Illinois Chicago

**Edward Kreinar**
HawkEye360

## Abstract

Machine learning methods are ubiquitous in particle physics and have proven to be very performant. One unique application within particle physics is the FPGA-based trigger and data acquisition systems that require sub-microsecond latencies and specialized hardware. Accelerating these algorithms in low-latency specialized hardware, such as FPGAs, is also of great interest in many scientific fields. Development of machine learning algorithms on these systems has historically been difficult due to the specific expertise required to program FPGAs. We present a compiler based on High-Level Synthesis (HLS) called `hls4ml` to build machine learning models in FPGAs The use of HLS increases accessibility across a broad user community and allows for a drastic decrease in firmware development time. We demonstrate the effectiveness of this tool by synthesizing fully-connected neural networks of varying sizes and bit precisions. We then explore the required design space needed to map out the FPGA resources and latency.

## 1 Introduction

Machine learning (ML) methods deployed in data processing have been demonstrated to be extremely effective in many different tasks across particle physics. One interesting use case is the first stage of real-time data processing, called the Level-1 (L1) trigger, for experiments at the Large Hadron Collider (LHC) at CERN. Due to the extreme frequency of 40 MHz at which proton bunches collide at the LHC, data rates at the two multipurpose experiments, CMS [1] and ATLAS [2], are of the order of hundreds of terabytes per second. With such high data rates, the real-time system must filter events to reduce data rates to manageable levels for offline processing. Because of the extreme input data rates and the size of the data buffers, the L1 trigger system typically uses custom hardware with ASICs or FPGAs to handle the initial data rate using pipelined algorithms with latencies of hundreds of nanoseconds. As the LHC collision intensity increases, processing of this data becomes every more restrictive. The use of sophisticated ML algorithms in the L1 trigger would allow LHC

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

experiments to preserve or even enhance potential physics signatures such as those related to the Higgs boson, dark matter, and hidden sectors [3] that would otherwise be lost. In this study, we explore the implementation of neural networks in FPGAs, mapping out resource usage and latency for various deep neural network architectures and hyperparameters, and we demonstrate the feasibility of deep learning techniques in sub-microsecond FPGA applications. We also present a compiler based on High-Level Synthesis (HLS) called `hls4ml` to rapidly prototype machine learning models in FPGAs. In this work, we focus on the LHC trigger application. However, these technologies apply to many real-time detector systems across fundamental physics where low latency high throughput is critical.

The results presented use `hls4ml` to scan the network precision and parallelization to optimize DSP and other resources. We discuss generic techniques such as network compression, parallelization, and reduced precision, which can be applied to design efficient neural network implementations tailored for different applications in the physical sciences. The accessibility and ease of configurability in HLS allow for physicists to quickly develop and optimize machine learning algorithms targeting FPGA hardware.

## 2 Core concepts

As a proof of concept, we investigate for use in FPGAs is created for the task of classifying a jet as either a quark ($q$), gluon ($g$), W boson ($W$), Z boson ($Z$), or top quark ($t$) jet. Jets are collimated showers of particles that result from the decay and hadronization of quarks $q$ and gluons $g$.

At the LHC, due to the high collision energy, a particularly interesting jet signature emerges from overlapping quark-initiated showers produced in decays of heavy standard model particles, such as $W$ and $Z$ bosons and top quarks. It is the task of jet substructure [4–16] to distinguish the various radiation profiles of these jets from backgrounds consisting mainly of quark ($u, d, c, s, b$) and gluon-initiated jets.

The model is trained using a data set consisting of simulated jets with an energy of $p_{\mathrm{T}} \approx 1$ TeV, originating from light quarks, gluons, $W$ and $Z$ bosons, and top quarks produced in $\sqrt{s} = 13$ TeV proton-proton collisions [17, 18]. Jets are clustered from individual reconstructed particles, using the anti-$k_{\mathrm{T}}$ algorithm [19, 20] with jet-size parameter $R = 0.8$. This corresponds to a simplified version of the real-world scenario, in which jets with any $p_{\mathrm{T}}$ values would be processed.

We use 16 jet substructure observables [21–24]: $m_{\mathrm{mMDT}}$, $N_2^{\beta=1,2}$, $M_2^{\beta=1,2}$, $C_1^{\beta=0,1,2}$, $C_2^{\beta=1,2}$, $D_2^{\beta=1,2}$, $D_2^{(\alpha,\beta)=(1,1),(1,2)}$, $\sum z \log z$, and multiplicity [17, 18], as inputs to a neural network classifier. The input features are standardized by removing the mean and scaling to unit variance. The architecture is a fully-connected neural network with three hidden layers containing 64 neurons in the first layer and 32 neurons in the second and third layers. The activation function for the hidden layers is a rectified linear unit (ReLU) [25] while the output layer activation function is a softmax function to provide probabilities for each of the five classes.

The performance of the neural network classifier is shown in Fig. 1. Top-quark jets, by virtue of their large mass and three-prong nature, exhibit the best separation from the rest of the jet types. The $W$ and $Z$ jets are similar in performance because of their masses and two-prong nature while quark and gluon jets are notoriously challenging to classify [26].

## 3 Implementation

To adapt the network to an FPGA, we first evaluate the neural network with fixed point precision denoted by `<X,Y>` where `Y` is the number of bits representing the signed number above the binary point (i.e. the integer part), and `X` is the total number of bits. We scan the number of integer bits and then scan the number fractional bits.With `<16,6>` fixed-point precision, the fixed point calculations reproduce the receiver operating characteristic (ROC) curve expected with full floating-point precision with a negligible loss in performance.

With `hls4ml`, we explore the FPGA design space of this dense network through

- **compression**, the three-hidden-layer model with 70% of the parameters removed using iterative retraining with $L_1$ regularization and magnitude-based pruning [27, 28, 17];

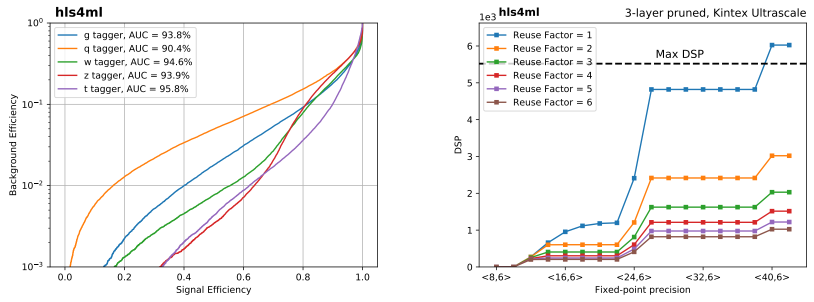

Figure 1: (Left) Performance of the deep neural network classifier quantified in a receiver operating characteristic (ROC) curve of signal efficiency versus mis-identification rate for quark, gluon, $W$ boson, $Z$ boson, and top quark jet identification. The mis-identification rate is based on an equal admixture of the other non-signal jet types. (Right) DSP usage in the compressed three-hidden-layer model as a function of the network precision. The various curves illustrate resource usage for different resource usage factors. The latency is given in clock cycles for a 200 MHz clock frequency.

- **quantization**, the precision of the inputs, weights, and biases;
- **parallelization**, the number of times a given multiplier is used for a layer computation; using a multiplier once is the most parallel (and quickly) a layer can be computed and is what we call a *reuse factor* of 1.

With these variables as handles to control the implementation of the network, we monitor the following firmware implementation metrics:

- **resources**: DSPs, Flip-Flops (FFs), and Look Up Tables (LUTs);
- **latency**: the time it takes to compute the full network;
- **initiation interval**: the time before a new set of inputs can be accepted.

The three-layer model is synthesized for a Xilinx Kintex Ultrascale FPGA with part number `xcku115-flvb2104-2-i` The clock frequency is fixed at 200 MHz, which is typical for the first stages of LHC triggers. Resource usage estimates are taken from the Vivado HLS 2017.2 [29] synthesis step and are found to be conservative when compared to implementation. While conservative, the short time required to make HLS resource estimates makes them useful for rapidly prototyping different network designs and deriving useful trends.

In Fig. 1 (right), we report the DSP usage as a function of precision of the fixed point calculations, `<X,6>`. Different curves are shown for different values of the reuse factor. As the reuse factor increases, the DSP usage is reduced proportionally to the reuse factor. The DSP resource usage has steps in resource usage as a function of the network precision due to the maximum precision of the Xilinx FPGA DSPs. In the figure, we also indicate the maximum number of DSPs available in this particular Xilinx FPGA. The corresponding LUTs (FFs) consumption stays below the 30% of the available capacity. The latency of the network inference stays between approximately 10 and 35 clock cycles (50 to 175 ns), with a small dependence on the fixed-point precision. It increases by 4 clocks with each increment in the reuse factor, corresponding to the four layers of neuron values that must be computed. By design, the initiation interval and the reuse factor match as a new input can be introduced to the algorithm only when all multiplications for a given layer are completed.

## 4 Recent features and expanded capability

To enhance the flexibility of `hls4ml`, several additions have been developed, including:

- extension of dense networks to allow for significantly larger networks in terms of the number of neurons per layer;
- inclusion of zero-suppressed weights for weights stored in on-chip memory or block random access memory (BRAM) reducing the use of on-chip logic registers;

- addition of binary and ternary matrix multiplication.

For each of these additional components, the ultimate goal is to allow for full flexibility to implement larger ML algorithms in applications where the latency restrictions are not as strict as within the L1 trigger.

When scaling to large dense networks with more than $10^4$ multiplications, the operations need to be restructured to effectively utilize the BRAM. In this scenario, weights are stored in the BRAM and then systematically retrieved from the BRAM and distributed to the DSPs. To allow for a maximally distributed set of weights, the reuse factor is utilized to ensure that only a fixed number of multiplications is performed per clock period. This number is functionally limited to the number of DSPs on a given FPGA. Thus, to maximize the resource efficiency, the matrix multiplication operations should be restructured in time so that a fraction of the multiplications (as determined by the reuse factor) are done per clock period and this fraction is immediately accumulated. This approach minimizes bottlenecks at the accumulation step and utilizes the largest number of DSPs. Additionally, an optimization of the HLS code structure was performed to ensure that the resource usage is comparable with the equivalent RTL implementation. With this, dense networks having as many as $10^6$ weights can be inferred on a single FPGA without going off-chip to read the network weights.

Large matrix multiplication with sparse matrices can be further optimized through the use of zero suppression. In this scenario, weights are stored with an additional pair of indices that determine the row and column of each weight. These indices and the weight are encoded into a single number. Provided the weight precision, and index precision is below 32 bits, a single BRAM element can store each weight. In this way, the usage of the on-chip memory can further be optimized. Multiplication with sparse matrices then requires that one carry the index around until the final accumulate. As with the large dense network implementations, to fix the number of DSPs used per clock, sparse matrix multiplication can be resized with the reuse factor. Here, only non-zero elements are multiplied and then the results are fanned out to an output vector of the full size of the layer.

Lastly, dedicated support has been added for networks with binary [30] or ternary [31] weights. Weights are stored as binary or ternary numbers and the multiplication can thus be performed through bit flip operations. This avoids the use of DSPs, enabling the implementation of larger networks, and consequently allows for a simplified routing scheme. Due to the lower precision of the weights, this requires significantly larger networks to perform at the same accuracy.

To understand the scaling, we consider three benchmark dense networks trained with the MNIST database of handwritten digits in `Keras`. The first model consists of 784 inputs, 3 hidden layers each with 128 hidden neurons and ReLU activation functions, and finally 10 outputs for classifying digits. The second (third) model is a binary (ternary) neural network [30] with batch normalization [32] applied before each binary (ternary) $\tanh$ activation function. The final model is the same as the first but with 95% of the weights removed using $L_1$ regularization and magnitude-based pruning [27, 28, 17]. The first model corresponds to $134 \times 10^3$ multiplications, and thus, at a minimum would require 134 clock periods of $10^3$ DSP operations. To obtain the minimum latency when targeting an initiation interval of 128, we reshape the matrix multiplication with a reuse factor of 112 for the first layer and 128 for the other layers. For targeting an initiation interval of 4096, we set the reuse factor to 4096 for all layers. The accuracy, FPGA resources, and latency for these models are shown in Table 1. Further optimization is required to reduce the LUT usage for the 95% pruned model with zero-suppression and an initiation interval of 128.

Table 1: A summary of the MNIST dense models' performance on an FPGA implemented with different reuse factors and numerical precisions. Numbers are shown as percentages of the FPGA components, DSPs, BRAM, FFs, and LUTs, of a Xilinx Kintex Ultrascale FPGA with part number `xcku115-flvb2104-2-i` at 200 MHz based on Vivado HLS 2018.2 [29]. The initiation interval in clock periods is the same as the largest reuse factor used in the model.

| **Model** | **Initiation Interval** | **Accuracy** | **Latency** | **DSP** | **BRAM** | **FF** | **LUT** |
|---|---|---|---|---|---|---|---|
| MNIST dense model | 128 | 0.97 | 2.6 $\mu$s | 21% | 45% | 12% | 33% |
| MNIST binary dense model | 128 | 0.93 | 2.6 $\mu$s | 0% | 33% | 7% | 39% |
| MNIST ternary dense model | 128 | 0.95 | 2.6 $\mu$s | 0% | 33% | 7% | 40% |
| MNIST dense model, 95% pruned with zero suppression | 128 | 0.96 | 2.8 $\mu$s | 1% | 34% | 13% | 164% |
| MNIST dense model | 4096 | 0.97 | 68.1 $\mu$s | 1% | 66% | 27% | 83% |
| MNIST dense model, 95% pruned with zero suppression | 4096 | 0.96 | 82.1 $\mu$s | 0% | 34% | 9% | 25% |

## 5 Outlook

We introduce `hls4ml`, a deep neural network compiler based on HLS capable of porting fully-connected networks to an FPGA trained from conventional training frameworks such as `Keras` and `PyTorch`. We focus on the application of real-time event reconstruction and filtering at the LHC in custom electronics with FPGAs although the methods are broadly applicable to many real-time detector systems in the physical sciences that can benefit from machine learning methods. We consider two specific case studies: a fully-connected neural network to identify jets and larger fully-connected neural networks to identify handwritten digits from the MNIST data set. For the first study, we show it is possible to implement a fully-connected three-hidden-layer network in a Xilinx Kintex Ultrascale using roughly 10% of the available DSPs and latency of approximately 75–150 ns with a clock frequency of 200 MHz. This fits well into the allowed hardware trigger reconstruction budget of LHC detectors such as ATLAS and CMS. For the second study, we extend the capabilities and flexibility of `hls4ml` to allow larger neural network architectures for applications with less stringent time constraints of approximately 1–100 $\mu$s.

## References

[1] CMS collaboration, *The CMS Experiment at the CERN LHC*, *JINST* **3** (2008) S08004.

[2] ATLAS collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, *JINST* **3** (2008) S08003.

[3] Y. Gershtein, *CMS Hardware Track Trigger: New Opportunities for Long-Lived Particle Searches at the HL-LHC*, *Phys. Rev.* **D96** (2017) 035027 [1705.04321].

[4] A. J. Larkoski, I. Moult and B. Nachman, *Jet Substructure at the Large Hadron Collider: A Review of Recent Advances in Theory and Machine Learning*, 1709.04464.

[5] J. Cogan, M. Kagan, E. Strauss and A. Schwarztman, *Jet-Images: Computer Vision Inspired Techniques for Jet Tagging*, *JHEP* **02** (2015) 118 [1407.5675].

[6] J. Pearkes, W. Fedorko, A. Lister and C. Gay, *Jet Constituents for Deep Neural Network Based Top Quark Tagging*, 1704.02124.

[7] G. Louppe, K. Cho, C. Becot and K. Cranmer, *QCD-Aware Recursive Neural Networks for Jet Physics*, 1702.00748.

[8] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman and A. Schwartzman, *Jet-images — deep learning edition*, *JHEP* **07** (2016) 069 [1511.05190].

[9] D. Guest, J. Collado, P. Baldi, S.-C. Hsu, G. Urban and D. Whiteson, *Jet Flavor Classification in High-Energy Physics with Deep Neural Networks*, *Phys. Rev.* **D94** (2016) 112002 [1607.08633].

[10] S. Macaluso and D. Shih, *Pulling Out All the Tops with Computer Vision and Deep Learning*, 1803.00107.

[11] K. Datta and A. J. Larkoski, *Novel Jet Observables from Machine Learning*, *JHEP* **03** (2018) 086 [1710.01305].

[12] A. Butter, G. Kasieczka, T. Plehn and M. Russell, *Deep-learned Top Tagging with a Lorentz Layer*, 1707.08966.

[13] G. Kasieczka, T. Plehn, M. Russell and T. Schell, *Deep-learning Top Taggers or The End of QCD?*, *JHEP* **05** (2017) 006 [1701.08784].

[14] P. T. Komiske, E. M. Metodiev and M. D. Schwartz, *Deep learning in color: towards automated quark/gluon jet discrimination*, *JHEP* **01** (2017) 110 [1612.01551].

[15] A. Schwartzman, M. Kagan, L. Mackey, B. Nachman and L. De Oliveira, *Image Processing, Computer Vision, and Deep Learning: new approaches to the analysis and physics interpretation of LHC events*, *J. Phys. Conf. Ser.* **762** (2016) 012035.

[16] J. M. Butterworth, A. R. Davison, M. Rubin and G. P. Salam, *Jet substructure as a new Higgs search channel at the LHC*, *Phys. Rev. Lett.* **100** (2008) 242001 [0802.2470].

[17] J. Duarte et al., *Fast inference of deep neural networks in FPGAs for particle physics*, *JINST* **13** (2018) P07027 [1804.06913].

[18] E. Coleman, M. Freytsis, A. Hinzmann, M. Narain, J. Thaler, N. Tran et al., *The importance of calorimetry for highly-boosted jet substructure*, *JINST* **13** (2018) T01003 [1709.08705].

[19] M. Cacciari, G. P. Salam and G. Soyez, *The Anti-k(t) jet clustering algorithm*, *JHEP* **04** (2008) 063 [0802.1189].

[20] M. Cacciari, G. P. Salam and G. Soyez, *FastJet user manual*, *Eur. Phys. J. C* **72** (2012) 1896 [1111.6097].

[21] D. Adams et al., *Towards an Understanding of the Correlations in Jet Substructure*, *Eur. Phys. J.* **C75** (2015) 409 [1504.00679].

[22] A. J. Larkoski, S. Marzani, G. Soyez and J. Thaler, *Soft Drop*, *JHEP* **05** (2014) 146 [1402.2657].

[23] A. J. Larkoski, G. P. Salam and J. Thaler, *Energy Correlation Functions for Jet Substructure*, *JHEP* **06** (2013) 108 [1305.0007].

[24] I. Moult, L. Necib and J. Thaler, *New Angles on Energy Correlation Functions*, *JHEP* **12** (2016) 153 [1609.07483].

[25] V. Nair and G. E. Hinton, *Rectified linear units improve restricted Boltzmann machines*, in *Proceedings of ICML*, vol. 27, pp. 807–814, 06, 2010.

[26] L. Asquith et al., *Jet Substructure at the Large Hadron Collider : Experimental Review*, 1803.06991.

[27] S. Han, J. Pool, J. Tran and W. J. Dally, *Learning both weights and connections for efficient neural networks*, *CoRR* **abs/1506.02626** (2015) [1506.02626].

[28] S. Han, H. Mao and W. J. Dally, *Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding*, *CoRR* **abs/1510.00149** (2015) [1510.00149].

[29] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons and F. Morgan, *Xilinx vivado high level synthesis: Case studies*, in *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET Year*, pp. 352–356, 2014.

[30] M. Courbariaux and Y. Bengio, *Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1*, *CoRR* **abs/1602.02830** (2016) [1602.02830].

[31] F. Li and B. Liu, *Ternary weight networks*, *CoRR* **abs/1605.04711** (2016) [1605.04711].

[32] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, *CoRR* **abs/1502.03167** (2015) [1502.03167].