Smart pixel sensors: towards on-sensor filtering of pixel clusters with deep learning

Jieun Yoo[1], Jennet Dickinson[2], Morris Swartz[3], Giuseppe Di Guglielmo[2,4], Alice Bean[5], Doug Berry[2], Manuel Blanco Valentin[4], Karri DiPetrillo[6], Farah Fahim[2,4], Lindsey Gray[2], James Hirschauer[2], Shruti R. Kulkarni[7], Ron Lipton[2], Petar Maksimovic[3], Corrinne Mills[1], Mark S. Neubauer[8], Benjamin Parpillon[2,1], Gauri Pradhan[2], Chinar Syal[2], Nhan Tran[2,4], Dahai Wen[3], Aaron Young[7]
[1] University of Illinois Chicago, [2] Fermi National Accelerator Laboratory, [3] Johns Hopkins University, [4] Northwestern University, [5] University of Kansas, [6] The University of Chicago, [7] Oak Ridge National Laboratory, [8] University of Illinois Urbana-Champaign
USA

ABSTRACT

High granularity silicon pixel sensors are at the heart of energy frontier particle physics collider experiments. At an collision rate of 40 MHz, these detectors create massive amounts of data. Signal processing that handles data incoming at those rate and intelligently reduces the data within the pixelated region of the detector *at rate* will enhance physics performance and enable physics analyses that are not currently possible. Using the shape of charge clusters deposited in an array of small pixels, the physical properties of the traversing particle can be extracted with locally customized neural networks. In this first work, we present a neural network that can be embedded into the on-sensor readout and filter out hits from low momentum tracks, reducing the detector's data volume by 54.4-75.4%. The network is designed and simulated as a custom readout integrated circuit with 28 nm CMOS technology and is expected to operate at less than 300 μW with an area of less than 0.2 mm^2.

1 INTRODUCTION

With billions of readout channels and event rates as high as 40 MHz, pixel detectors in high-energy colliders generate petabytes of data per second [1, 2].The particle properties extracted from pixel detector data play a crucial role in physics measurements, but despite its importance to collider physics programs, pixel detector information is difficult to read out. The large data volume calls for multiple methods of data reduction. Zero suppression is employed to read out only active pixels, and the accumulated charge per pixel is digitized and represented with only a few bits. Even with these techniques, data rates at the current CMS and ATLAS experiments exceed bandwidth constraints for read out at the collision frequency of 40 MHz. Collisions of interest are selected using a hardware-based trigger system, which uses information from the other (non-pixel) detector subsystems to select events at a rate of < 1 MHz. Events that contain new physics only in the pixel data are not selected by the low-level trigger and are lost forever.

In this paper, we seek to overcome the limitations of pixel readout with local data reduction in dedicated circuits before transmitting information off of the detector. While pixel readout has traditionally been treated as a *lossless compression* task, we explore the paradigm of lossy compression within a given event to enable lossless readout over all possible collisions. To enable this strategy with optimal performance and yet keep the algorithms reconfigurable, we explore the use of neural networks with reprogrammable weights. A machine learning approach is required due to the complicated pulse shapes, a combination of drift and induced currents, generated by the pixel detector. The contribution from those two components is dependent on the sensor geometry where the charged particle impacts the detector and the particle's trajectory.

In this work, we develop, design, and study a neural network that will filter out pixel clusters originating from low-momentum charged particle tracks. We then optimize, design, and simulate that neural network in an integrated circuit and study its performance, power, area, and latency.

Related Work

Data compression of silicon tracking information, in both strip and pixel detectors, is explored in Ref. [3–5]. These works contain detailed studies of both lossless and lossy compression of the digital information content of the pixel and strip tracker detectors. These references provide a good baseline for understanding what data compression factor is possible for current tracking systems. Our work expands on these previous studies by: looking at analog and timing information; considering future pixel dimensions, studying the potential for cluster filtering by the track momentum; and designing a first implementation of on-chip detector algorithms.

Ref. [6] explores the use of pixel cluster shapes offline (off-detector) to extract direction information and reduce tracking combinatorics and complexity. The technique described in this study can be used in the future to extract directional information from charge clusters in a single pixel layer, providing similar benefits in terms of reduced algorithm complexity for tracking downstream, which is particularly important for online data processing systems.

Finally, our study relies on previous work for translating neural network algorithms into circuits using the `hls4ml` [7, 8] workflow. In particular, the first implementation using `hls4ml` to build a reconfigurable ASIC [9] for calorimeter on-detector data compression provides a basis for much of the technology developed in this paper.

2 SENSOR GEOMETRY AND DATASET

Our studies are based on a simulated dataset of silicon pixel clusters produced by charged particles (pions) [10], and in the rest of the paper, the key kinematic property for particles is the *transverse momentum* (p_T), whose unit is *electronvolt* (eV).

Fig. 1.a sketches out characteristics of the pixel sensor. Within the pixel sensor area, we define a *cluster region of interest*, shown

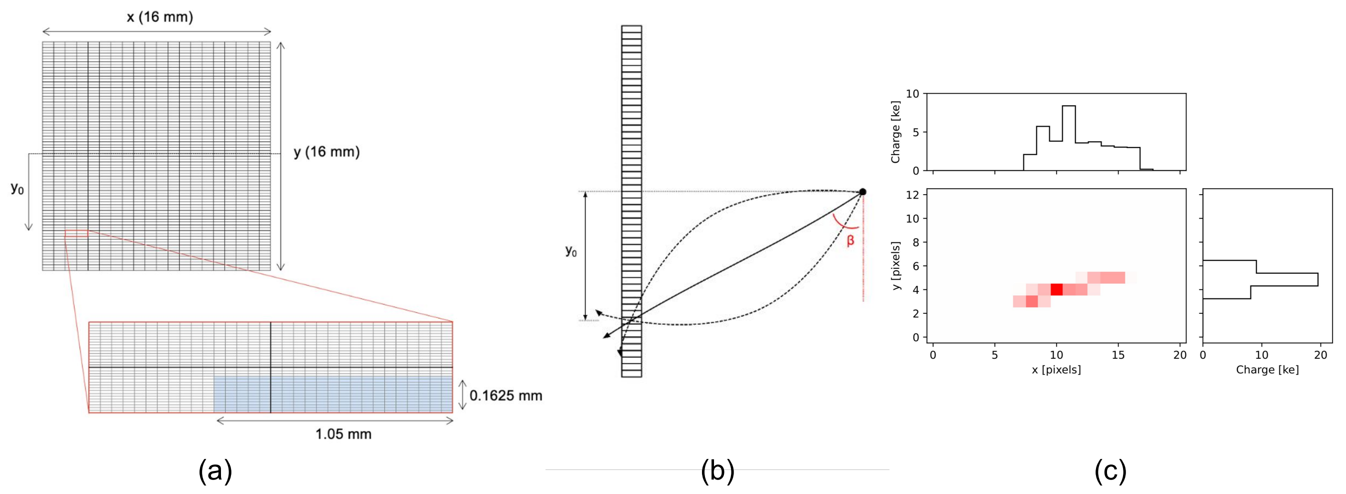

Figure 1: (a) A schematic of the pixel sensor area and the specific region of interest (blue) of 21×13 pixels for a given cluster. The magnetic field is parallel to the sensor x coordinate. (b) A diagram of three charged particles traversing our simulated silicon sensor at the same y_0 position. The sensor is viewed in the bending plane of the magnetic field. The solid track corresponds to a charged particle with high p_T, while the two dashed tracks correspond to low p_T particles with opposite charge. (c) An example of charge cluster and the corresponding x- and y-profile projections. The color scale represents the collected charge. The cluster is locate in $y_0 = 2.3$ mm and has $p_T = 1.9$ GeV.

in blue, which corresponds to 21×13 pixels in x and y, respectively. This region is large enough to fully encompass a charge cluster and serves as input to the ML algorithm used to extract cluster features. The position (x, y) where the charged particle traverses the sensor mid-plane is uniformly distributed across the central 3×3 pixel array. The shape of the charge deposited in the pixel array is sensitive to this position and to the particle's angle of incidence. The incident angle in the $x - z$ plane is denoted by α, and by β in the $y - z$ plane (see Fig. 1.b).

Due to the bending of charged particle tracks in the magnetic field, the shape of the charge cluster also depends on the particle's p_T, which is highly correlated with β. The shape of the cluster also depends strongly on its azimuthal position with respect to the center of the sensor, which is denoted by the coordinate y_0.

For a given cluster, the sum over pixel columns projects the cluster shape onto the x-axis: this distribution is referred to as the *x-profile*. The sum over pixel rows, *y-profile*, which projects the cluster shape onto the y-axis, is sensitive to incident angle β and therefore to the particle's p_T. Finally, Fig. 1.c shows an example of cluster with the corresponding x- and y-profile projections.

3 NEURAL NETWORK DESIGN

In our exploration, we designed and compared three neural network classifier to identify clusters initiated by high p_T charged particles. The high p_T signal class contains tracks with $p_T > 200$ MeV.

The simulated dataset of 800,000 clusters is used for training. The simulated dataset is split into a training set (80%) and a test set (20%) to be used for evaluation of the algorithm's performance. All models were implemented in TensorFlow (`v2.10.0`) [11] using the Keras API [12]. Neural network trainings were run for 200 epochs where early stopping was used if the loss function showed no improvement after 20 epochs. A batch size of 1024 was used in all models. The Adam optimizer [13] with a learning rate of 0.001 was used in conjunction with the Keras Sparse Categorical Cross entropy loss function in all models.

The models are trained with three output categories: positively charged and $p_T < 200$ MeV; negatively charged and $p_T < 200$ MeV; and $p_T > 200$ MeV (high p_T), both positively and negatively charged. A softmax activation was used in all models to generate classification probabilities between 0 and 1, and each cluster is assigned the classification label corresponding to the highest probability.

The charge cluster shape along the axis parallel to the magnetic field direction (sensor x) is assumed to be largely uncorrelated with track p_T. The projection of the cluster shape onto the sensor y-axis is therefore used as the training input to the classifier.

Three training setups and associated models are developed corresponding to input features of different complexity in order to demonstrate how additional information improves p_T discrimination:

Model 1 uses two input features the cluster y-size and position y_0. y-size is the number of columns with nonzero charge deposited after 4 nanoseconds. The model consists of one dense layer with 128 neurons and 771 parameters. This model provides a test of performance with minimal information provided to the neural network.

Model 2 leverages position y_0 and y-profile information that is the amount of charge collected in each row of pixels after 4 nanoseconds. The total number of input features are 14:

Model	Sig. efficiency	Bkg. rejection
Model 1	84.8 %	26.6 %
Model 2	93.3 %	25.1 %
Model 3	97.6 %	21.7 %

Table 1: Comparison of model performance in terms of signal efficiency and background rejection.

Model	Sig. efficiency	Bkg. rejection
Full precision	93.3 %	25.1 %
Quantized inputs	88.8 %	25.8 %
Quantized weights & inputs	87.3 %	28.2 %

Table 2: Comparison of the baseline model performance at different stages of the quantization in terms of signal efficiency and background rejection.

13 y-profile and one y_0 position. This model consists of one dense layer with 128 neurons and 2307 parameters.

Model 3 leverages timing information associated with the y-profile. This is the most complex model takes as input the cluster y-profile distribution at eight time slices (13×8 features) and the y_0 position (1 feature). This model uses a convolutional neural network (CNN) to pass a time-lapse picture of the cluster charge to the network. The final model contains 83,331 parameters.

Table 1 compares two figures of merit for each model. The *signal efficiency* is the fraction of clusters with $p_T > 2$ GeV that are classified as high p_T; the *background rejection* is the fraction of clusters with $p_T < 2$ GeV that are classified as low p_T.

Model 1 (y-size and y_0) has the simplest architecture and achieves the highest data reduction rate. However, the information contained in only two features is insufficient for achieving a high signal efficiency, and this model selects less than 85% of tracks with $p_T > 2$ GeV. Model 2 (cluster y-profile) achieves an accuracy of 93.3% for tracks with $p_T > 2$ GeV, and remains sufficiently compact for implementation on-ASIC. The inclusion of timing information in Model 3 achieves an additional 4% gain in the signal efficiency and is the most accurate overall. However, the extraction of time-sliced charge information presents challenges to the chip architecture that merit further study but remain outside the scope of this work. *Model 2 is therefore chosen as the baseline model for our hardware implementation.*

4 MODEL QUANTIZATION

For implementation on-ASIC, the neural network weights must also be quantized to a precision of a few bits without significant loss in performance. The QKeras library [14, 15] is used to perform quantization-aware training, enabling an early evaluation of the impact of low bit precision on the model's performance. This allows an initial assessment of the trade-offs between accuracy and resource utilization before finalizing a design for the ASIC implementation. The quantized baseline model consists of a quantized dense layer and a quantized representation of a ReLU activation function. Additionally, a Batch Normalization (BN) layer that provides a regularization effect during training, akin to dropout, was incorporated to prevent overfitting. A softmax activation generates classification probabilities between 0 and 1, and each cluster is assigned the classification label corresponding to the highest probability. The final quantized model has 2,563 parameters, uses five-bit weights, 10-bit activations, and is illustrated in Fig. 2.

Table 2 compares the performance of the full precision model, the model with quantized input features only, and the model with quantized input features and weights.

Following the selection of a baseline quantized model, an additional architechtural exploration was performed that fully accounts for hardware constraints. During this stage, five major design decisions were made in order to compress the model without significant loss of performance. Fig. 2.b depicts the final model architecture, with all updates to the design incorporated.

In summary: **(I)**. We folded the BN layer into the Dense layer. This optimization technique combines the BN parameters with the Dense layer's weights and biases, effectively reducing computational costs during inference while maintaining the model's performance [16]. **(II)**. We examined the trade-off between accuracy and area by considering the reduction in the number of neurons and bit precision. In our exploration, we sought to balance network complexity and hardware area while striving to maintain the accuracy of the original network outlined in Sec. **??**. The optimal model was found to reduce the number of neurons of the first dense layer from 128 to 58. **(III)**. We reduced the number of bits in the weights and activations by 2 bits each. This produced a hardware implementation with a third of the area originally necessary with a drop in signal efficiency of only 3.5%. **(IV)**. To further reduce the computational complexity of the model without sacrificing the overall predictive capability, we utilized an Argmax function as the final layer instead of the conventional Softmax activation function. **(V)**. We opted to remove the y_0 coordinate from the model entirely. Rather than directly providing the network with the value of this coordinate, we train the baseline network many times on clusters in restricted ranges of y_0: this is referred to as the *region specific implementation.* The network architecture in each y_0 region is identical, but the values of the reprogrammable network weights can be tuned based on the ASIC's y_0 position. The input y-profile distribution is then expanded from 13 to 16 bins by padding with zeros, so that the pixel array can be comprised of a round number.

The final optimized model has 1,163 trainable parameters, which is a reduction of 55% with respect to the original model.

5 HARDWARE IMPLEMENTATION WITH HLS

To translate the algorithm from a quantized graph representation into an optimal hardware implementation, we use the `hls4ml` workflow. It is an open-source Python framework for co-design and translating machine learning algorithms into hardware implementations [17]. `hls4ml` translates the quantized model into C++ code for Siemens Catapult HLS [18]. The HLS tool generates a hardware description at the register-transfer level (RTL) for the traditional ASIC flow. We have opted to fully parallelize the hardware logic, resulting in a combinational implementation of our models from HLS. This choice is driven by the desire to minimize the latency of

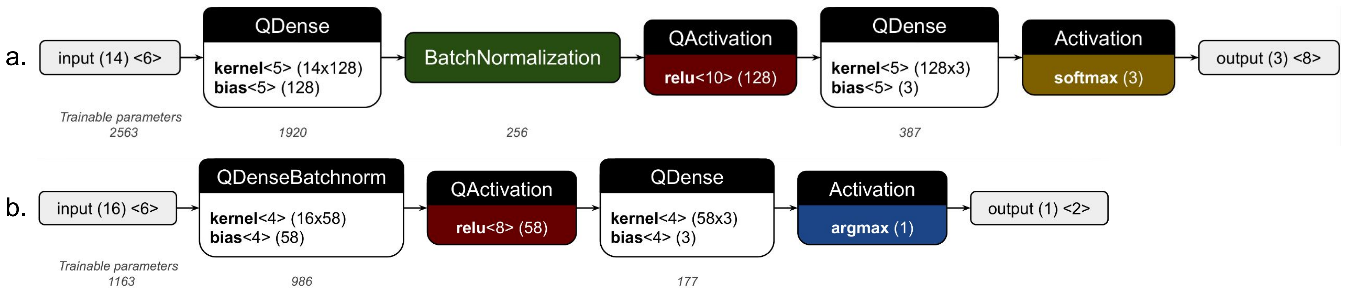

Figure 2: (a) Baseline quantized model consisting of 14 inputs (13 bins of y-profile plus y_0), three outputs, and two hidden layers containing 128 and three neurons each. For each layer, the dimension is reported in round brackets and the bit-width of the fixed-point representation in angular brackets. (b) Final model architecture with reduced bit-precision and trainable parameters. For each layer, we report the dimensions in round brackets and the bit-width of the fixed-point representation in angular brackets.

the neural network. We integrated the resulting RTL design into the system alongside registers and data movers.

5.1 Integrated sensing and edge computing

Finally, the ML algorithm must be integrated into readout integrated circuit (ROIC). The physical layout of the super-pixel is shown in Figure 3. The size of the digitally implemented super-pixel is $889\,\mu m \times 222\,\mu m$. The green areas correspond to the analog circuit islands, while the red contains the digital logic. Given our design, the registers require a one-time setup and leakage at low temperatures is deemed insignificant, thus the combinatorial logic is projected to account for the majority of the power utilization. The anticipated power consumption of the digital logic for 28nm CMOS technology is roughly 300 μW, measured when toggling 50% of the inputs every 25 ns (clock cycle).

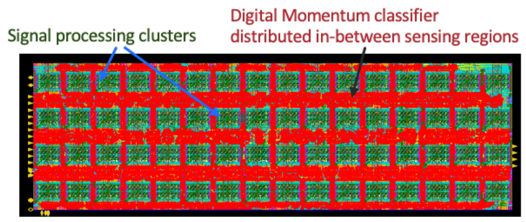
Signal processing clusters
Digital Momentum classifier distributed in-between sensing regions

Figure 3: Physical layout of the 16×16 pixel array digital implementation. The analog islands are in green and the digital implementation of the neural network are in red.

6 CONCLUSIONS

This study makes a number of novel contributions. First, we introduced a first public dataset which can be used for the benchmark task of pixel on-sensor data reduction. We developed and compared a number of neural-network-based approaches from simple cluster size information to cluster distributions to time-evolved cluster distributions and present their performance for the clustering filtering task. Finally, we optimizes the algorithm for circuit implementation by exploring quantization of inputs and neural network parameters. We synthesized and integrated the algorithm in a readout integrated circuit. The design is expected to operate at less than 300 μW with an area of less than 0.2 mm^2.

REFERENCES

[1] The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008. Also published by CERN Geneva in 2010.
[2] The CMS experiment at the CERN LHC. The Compact Muon Solenoid experiment. *JINST*, 3:S08004, 2008. Also published by CERN Geneva in 2010.
[3] M Garcia-Sciveres and X Wang. Data compression considerations for detectors with local intelligence. *Journal of Instrumentation*, 9(10):C10011, 10 2014.
[4] Maurice Garcia-Sciveres and Xinkang Wang. Data encoding efficiency in binary strip detector readout. *Journal of Instrumentation*, 9(04):P04021, 2014.
[5] Maurice Garcia-Sciveres and Xinkang Wang. Data encoding efficiency in pixel detector readout with charge information. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 815:18–22, 2016.
[6] Patrick J Fox, Shangqing Huang, Joshua Isaacson, Xiangyang Ju, and Benjamin Nachman. Beyond 4d tracking: using cluster shapes for track seeding. *Journal of Instrumentation*, 16(05):P05001, 2021.
[7] Javier Duarte et al. Fast inference of deep neural networks in FPGAs for particle physics. *JINST*, 13(07):P07027, 2018.
[8] FastML Team. fastmachinelearning/hls4ml, 2023.
[9] Giuseppe Di Guglielmo, Farah Fahim, Christian Herwig, Manuel Blanco Valentin, Javier Duarte, Cristian Gingu, Philip Harris, James Hirschauer, Martin Kwok, Vladimir Loncar, et al. A reconfigurable neural network asic for detector front-end data compression at the hl-lhc. *IEEE Transactions on Nuclear Science*, 68(8):2179–2186, 2021.
[10] Morris Swartz and Jennet Dickinson. Smart pixel dataset, November 2022.
[11] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
[12] Francois Chollet et al. Keras, 2015.
[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.
[14] Claudionor N. Coelho, Aki Kuusela, Shan Li, Hao Zhuang, Jennifer Ngadiuba, Thea Klaeboe Aarrestad, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, and Sioni Summers. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence*, 3(8):675–686, 6 2021.
[15] Erwei Wang, James J. Davis, Daniele Moro, Piotr Zielinski, Jia Jie Lim, Claudionor Coelho, Satrajit Chatterjee, Peter Y. K. Cheung, and George A. Constantinides. Enabling binary neural network training on the edge, 2021.
[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.
[17] FastML Team. fastmachinelearning/hls4ml, 2021.
[18] Siemens. Catapult HLS. https://eda.sw.siemens.com/en-US/ic/ic-design/high-level-synthesis-and-verification-platfor.