Streaming Convolutional Neural Network FPGA Architecture for RFSoC Data Converters

Andrew Maclellan*, Louise H. Crockett, and Robert W. Stewart
Department of Electronic and Electrical Engineering
University of Strathclyde, Glasgow, Scotland
*a.maclellan@strath.ac.uk

***Abstract*—This paper presents a novel Convolutional Neural Network (CNN) FPGA architecture designed to perform processing of radio data in a streaming manner without interruption. The proposed architecture is evaluated for radio modulation classification tasks implemented on an AMD RFSoC 2x2 development board and operating in real-time. The proposed architecture leverages optimisation such as the General Matrix-to-Matrix (GEMM) transform, on-chip weights, fixed-point arithmetic, and efficient utilisation of FPGA resources to achieve constant processing of a stream of samples. The performance of the proposed architecture is demonstrated through accuracy results obtained during live modulation classification, while operating at a sampling frequency of 128 MHz before decimation. The proposed architecture demonstrates promising results for real-time, time-critical CNN applications.**

***Index Terms*—deep learning, wireless communications, FPGA, RFSoC, PYNQ, modulation classification.**

I. Introduction

Deep Learning (DL) has emerged as a powerful tool for improving Physical Layer (PHY) wireless communications. As new technologies such as 5G and 6G emerge, DL will play a crucial role in all layers of the communication stack. Previous research has shown impressive results in various DL applications for wireless communications such as channel estimation [1]–[3], signal identification [4], decoding [5], and synchronisation [6].

DL in communications is a rapidly developing area, with breakthrough results being reported often. To fully exploit these algorithms they must be deployed on existing and new radio transceivers, many of which use FPGA and SoC architectures due to their low-power and parallel processing capabilities. This work targets the AMD RFSoC 2x2 development board, which features a Radio Frequency System on Chip (RFSoC) device. RFSoC devices combine high-accuracy analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) operating at Gsps with programmable logic for custom algorithm implementations [7].

This paper builds upon prior work in modulation classification, including the original paper by O'Shea et al. [4], which established a dataset and CNN model for the task, and Umuroglu et al.'s low-precision data flow CNN architecture [8]. Our work differs from [8] as we have implemented a data flow model using higher precision weights to keep the CNN model similar to O'Shea et al.'s. Related works have implemented CNN architectures for modulation on an RFSoC ZCU111 development board [9], with one work classifying between three modulation schemes received at the ADC [10]. In contrast, our novel hardware-friendly CNN architecture for the AMD RFSoC 2x2 development board includes on-chip storage of weights, 18-bit fixed-point precision, and preserves the data flow structure of wireless communications pipelines by stream processing of every sample without sacrificing input speed. We demonstrate the effectiveness of our architecture with an implementation of modulation classification on the RFSoC board that predicts modulation schemes received on the ADC.

II. Application - Modulation Classification

Modulation classification is the task of identifying which modulation scheme is used in a radio signal. In telecommunications, modulation refers to the process of encoding information onto a carrier signal for transmission. Modulation classification is an important task in a communication system, as the receiver must be able to accurately identify the modulation scheme used in order to properly decode a signal.

In our modulation classifier we aim to identify one of 8 modulation schemes used on a signal: QPSK, BPSK, QAM16, QAM64, 8PSK, PAM4, GFSK, and CPFSK. The CNN dimensions used can be seen in Table I.

TABLE I
Dimensions of CNN used for Modulation Classification.

Layer type	Dimensions	Activations
Input	2×128	-
Convolution	$64 \times 3 \times 1$	ReLU
Convolution	$16 \times 3 \times 2$	ReLU
Fully-connected	1984×128	ReLU
Fully-connected	128×8	Softmax
Output	1×8	-

A. Dataset Creation & Training the CNN

The dataset was designed to train a CNN to identify modulation schemes in the received signal affected by the RFSoC loopback and a simulated channel. $30,000$ frames of $4,096$ samples each, representing various modulation schemes, were generated and passed through a Rician channel with added noise achieving an signal-to-noise ratio (SNR) of 30dBm. The digital-to-analog converter (DAC) transmitted the data, which was then captured in 128 sample frames upon loopback reception (DAC to ADC via an RF cable). This recorded signal serves as the training dataset for the CNN model [11].

The recorded data was then used to train the CNN model in Tensorflow [12]. The data was split into training, validation, and testing with a $80/10/10$ ratio split. The network was trained for 20 epochs with a batch size of 128. The first three layers used L2 kernel regularisation. The resulting trained layer weights and biases were saved and quantised to 18-bits using MATLAB's Fixed-Point Designer [13]. Fixed-Point Designer selects the best fractional point by allocating the minimum number of integer bits needed to represent the largest weight/bias for each layer.

III. Real-time Streaming CNN Architecture

Our proposed architecture is designed to perform DL modulation classification continuously on the received decimated RF data with no pauses. In our Tensorflow training, we provided finite frames of data to the model and trained it to identify signals based on this buffer of data. While the implementation of the DL model on hardware also makes a decision on a finite frame of data, we have adapted the layers to allow for samples to flow through uninterrupted. This means that the implemented DL model's latency is deterministic and therefore can be synchronised with other parts of a receiver design. Additionally, applications such as decoding, channel estimation, or anomaly detection where dropping samples can negatively affect the performance of the receiver system, can benefit from this streaming-based CNN architecture.

The design philosophy of this architecture is centered on the data-flow model, treating incoming data as a continuous stream of infinite samples and processing them in real-time. A significant challenge encountered in this approach is the increased production of output samples by convolutional layers, leading to potential loss in samples due to processing lag. To address this issue, the overall clock rate of the model was increased by a factor of 32. The proposed DL model follows a Digital Down Converter (DDC), which has reduced the sampling rate of the incoming signal by a factor of 32 from the RF-ADC input rate.

The following sections will describe the design choices and methodologies for implementing the proposed CNN architecture on the RFSoC.

A. Input Data Pre-processing

The received IQ signal from the antenna is digitised in the ADC and modulated to baseband before entering the FPGA at a rate of 128 Msps. To minimise computational requirements while adhering to the Nyquist sampling theorem, the signal is passed through a filter decimation chain that reduces the sampling rate by a factor of 32, to 4 Msps. The decimation chain implements a low-pass filter with stopband frequency equal to: $f_{stop} = \frac{1}{32} \times \frac{fs}{2}$, to eliminate any aliased components while maintaining the 1MHz bandwidth of the modulated signals. Since the received signal is complex, it is split into two paths and each path is decimated with identical filters. Once decimated, the I and Q samples are interleaved into a one-dimensional input before entering the first convolutional layer.

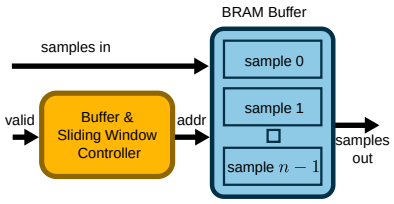

Fig. 1. Samples written into buffer with controller.

B. Convolutional Layers

Convolutional layers are the main building blocks of CNNs. They are able to extract feature maps from an input by sliding a set of filters (or kernels) over the input and computing the dot product between the filter weights and the overlapping input values. The output resulting from a convolution operation is a feature map. These layers are very useful for identifying localised features within an input, however, convolutional layers are also computationally complex. In our architecture, we have used the General Matrix-to-Matrix (GEMM) Transform to perform the convolutions and to minimise their complexity. In simple terms, the GEMM transform for convolutional layers simplifies the whole calculation for a layer into one large matrix-to-matrix multiplication [14].

The convolutional layers are split into three stages: the input data buffer, the sliding window generator, and the matrix-to-vector multiplier. The input data buffer receives the samples into the layer and buffers them in 1-dimensional on-chip UltraRAM (URAM). The sliding window generator then performs the GEMM transform on the input samples so that they can be multiplied with the pre-processed layer filters stored in Block-RAM. The sliding window generator outputs the resulting matrix one sample or row at a time and multiplies them with the GEMM transformed filter weights to produce the layer output.

1) Bursting Data: After the data has undergone decimation, it is fed into the DL model at a slower rate than the model's sample rate. To address this, we buffer the samples into Block RAM and output them in 256-sample long bursts, clocked at the DL model's sample rate. This bursting operation is implemented using a ping-pong buffer to prevent any data loss during conversion. The incoming samples are quantised to `int16` precision for each I and Q sample.

2) GEMM Transform and Sliding Window: Once the samples are buffered, the Sliding Window Controller (SWC) reads out the GEMM-transformed samples. The 18-bit filter/kernel weights were transformed into a matrix representation. The interleaving of C channels for each filter N was performed first. Then the unrolling and concatenation of each filter N formed a single matrix, denoted as $\tilde{\Theta}^{\text{conv}}$, with dimensions $N \times CKJ$ (where N is the number of filters, C is the number of channels, and J and K refer to the number of columns and rows of the filter weights, respectively). This transformation was conducted offline and the filter weights were stored on-chip for efficient access. The illustration of the matrix transform for the filters can be seen in Figure 2.

The GEMM transform for the input samples to a convolutional layer involves the following steps:

- Re-shaping the input into a 2-dimensional matrix where the channels C of the input are interleaved. Typically, the input to a layer is a 3-dimensional tensor.
- Striding and replication. The input is replicated based on the stride of the layer's filter window.
- Unrolling. For each step of the stride, the input samples under the filter window are unrolled and concatenated to make the resulting matrix $\tilde{X}^{\text{conv}}$.

Unlike the filter weights, the input transformation has to be performed during operation. A downside to the transformation approach is that the input samples have to be replicated in order to achieve the same output as a sliding window convolution. We implemented a SWC to perform the transform while running live and receiving a continuous stream of input samples.

In the model architecture, seen in Figure 1, the SWC handles the reading and writing of samples to on-chip RAM. The controller contains a state machine that performs the sliding window transformation, depicted in Figure 2, and reads the samples out of memory based on the resulting order.

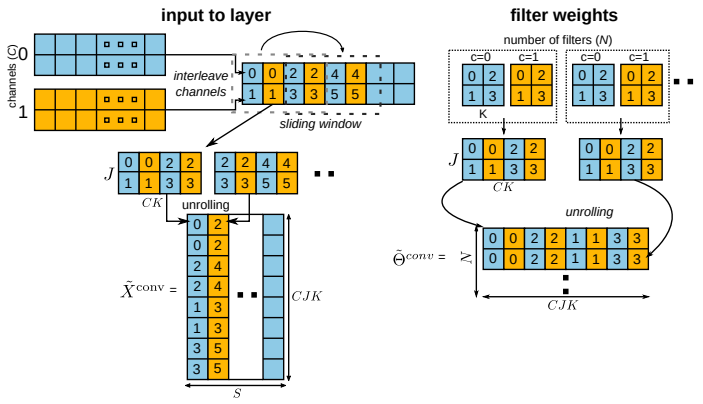

Fig. 2. GEMM Transform for input samples and filter weights.

Firstly, the columns, K of each input channel, C, are interleaved to form one 2-dimensional matrix. Next, the flattened sliding window is passed over the 2-dimensional input and splits it into smaller matrices based on how many strides S the sliding window performed. Each of these split matrices are then unrolled and concatenated to form the resulting GEMM-transformed input matrix, $\tilde{X}^{\text{conv}}$, with dimensions $CJK \times S$. The SWC sends storage addresses to the on-chip RAM to read out the resulting matrix.

3) Matrix to Vector Multiplication: Once the input has been transformed into a matrix, it is then multiplied with the transformed filter weights in the matrix-vector multiplier stage to produce the output of the convolution $\tilde{Y}^{\text{conv}} = \tilde{\Theta}^{\text{conv}} \times \tilde{X}^{\text{conv}}$.

In this proposed architecture, we present multiple implementations of the matrix-vector multiplier. The first option involves parallel processing of vector-vector multiplications, with input vectors sourced from the SWC. The second option performs the matrix-vector calculation sample-by-sample using N multiply-accumulate (MAC) units, where N is the number of filters in the convolutional layer. This latter design is optimized for larger multiplications. Finally, another option is a serialised implementation of the matrix-vector multiplier that time-shares one (or a small number) of MAC units, depending on the number of spare clock cycles.

All implementations maintain a 18-bit fixed-point precision on the output and carefully minimise the wordlength growth. All optimisations necessitate a higher sampling rate than the input rate of the model. This requirement is driven by multiple factors, including the need to replicate samples in the GEMM transform stage and the increased number of samples output in certain convolutional layers.

C. Dense Layers

A dense layer in a CNN is a fully connected layer where every neuron in the layer is connected to every neuron in the previous layer. The dense layer performs a matrix multiplication between the inputs and weights: $\tilde{Y}^{\text{dense}}_{n\times 1} = \tilde{W}^{\text{dense}}_{n\times p} \times \tilde{X}^{\text{prev}}_{p\times 1}$, where p is the number of features in X and n is the number of neurons in the dense layer.

Similarly to Section III-B3, the matrix-vector multiplication in the dense layers can be implemented in a number of ways for resource saving and throughput optimisations, as mentioned previously, while maintaining an 18-bit fixed-point precision at the output.

D. Activations and Bias

The activations and bias terms play a critical role in the functioning of a neural network. The activations, represented by $f(x)$, introduce non-linearity into the outputs of each layer through the application of non-linear functions. This ability to model non-linear data structures is crucial for the success of CNNs. The resulting output of a dense layer with activations and biases included is denoted by: $\tilde{Y}^{\text{dense}} = f(\tilde{W}^{\text{dense}} \times \tilde{X}^{\text{prev}} + b)$.

The bias terms, b, adjust the decision boundaries of the individual neurons. In the context of modulation classification, the Rectified Linear Unit (ReLU) activation function, represented as $y = \max(0, x)$, was utilised on a stream of input samples. The bias term was then added to either a vector or stream of samples, accompanied by a counter to index the specific bias value to be added.

IV. Resource Utilisation

Table II presents the resource utilisation of the modulation classification CNN model when implemented on the AMD RFSoC 2x2 development board which includes a Zynq Ultrascale+ XCZU28DR part. The results show that the CNN model utilises approximately 10% of the available DSP slices and 15% of the available BlockRAMs (BRAMs). The DSPs slices are used to perform the vector-matrix multiplication operations in each layer of the model, while the BRAMs store the intermediate samples from previous layers and the layer weights. This on-chip storage of intermediate results and weights helps to reduce the latency of the CNN model and improve its overall performance.

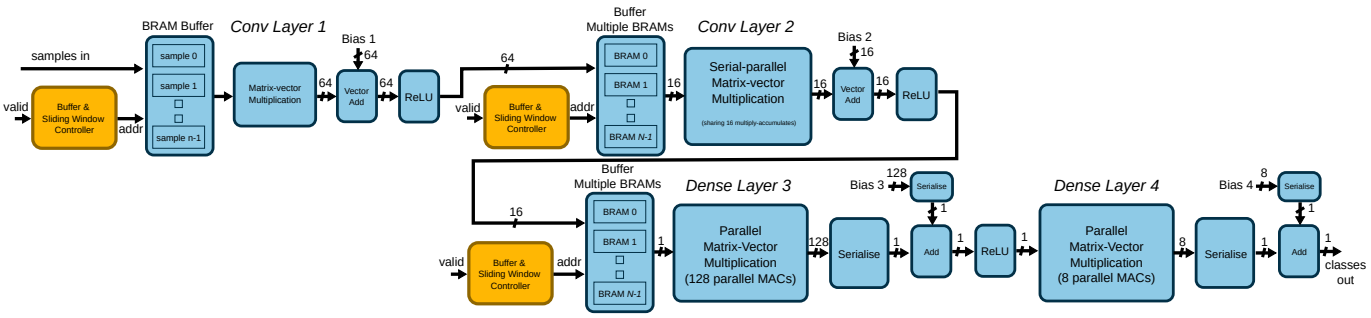

Fig. 3. Overall HDL CNN architecture for modulation classification.

TABLE II
FPGA RESOURCE UTILISATION OF MODULATION CLASSIFICATION CNN.

	Slice (LUTs)	Slice (Register)	DSPs	BRAMs	URAMs
Conv 1	3,810	4,008	64	0	1
Conv 2	5,359	8,815	256	32	0
Dense 1	8,343	15,569	128	136.5	0
Dense 2	593	592	8	0	0
Total	18,389	29,277	456	169.5	1
(%)	(4.32%)	(3.44%)	(10.67%)	(15.65%)	(1.25%)

V. PERFORMANCE RESULTS

The architecture described in Figure 3 depicts the overall architecture of the CNN for the applications of modulation classification. The interleaved IQ samples are received from the ADC as `int16` precision and converted to 18-bit fixed-point precision for the rest of the model. The weights for each layer are stored in BRAM as 18-bit fixed point.

On AMD RFSoC 2x2 board, a variety of modulation schemes were transmitted through the DAC and received by the ADC connected in loopback. The Processing System (PS) was used to send the modulated data to the Programmable Logic (PL) through a Direct Memory Access (DMA) IP. The transmission was executed using a cyclic buffer, followed by a rate change of 32 through a filter interpolation chain.

The received signals were processed through a decimation filter chain with a rate change of 32, prior to being fed into the trained CNN model for classification at a throughput of 128Mbit/s. The communication between the PS and PL was facilitated through the use of PYNQ, enabling the modulation schemes to be efficiently transmitted to the DMA. The confusion matrix plotted in Figure 4 highlights the performance capabilities of this proposed architecture. As can be seen, the model identifies all modulation schemes with a 70%+ accuracy, except for QAM64. The model confuses QAM16 and QAM64 more often than other classes because these two modulation schemes have the most similar waveforms. The drop in accuracy for QAM64 is due to the quantisation of the filter weights from trained floating-point values. This loss in precision will cause the model to struggle classifying modulation schemes. It becomes weaker at identifying the finer details between classes. To minimise the loss in accuracy through quantisation, future work could include training the model with fixed-point quantisation applied to the weights during training.

Fig. 4. Accuracy metrics for modulation classification with RFSoC in loopback.

VI. CONCLUSION

In this paper, we proposed a novel streaming CNN architecture designed to work with radio receivers. The objective was to create a data flow CNN model that can be seamlessly integrated into wireless communication pipelines. The proposed architecture processes every sample received at the input with no pauses, which makes it an ideal solution for time and data sensitive applications such as decoding, channel estimation, and others.

Experimental results showed mostly 70% or higher accuracy (except for QAM64) in modulation classification tasks when implemented on an AMD RFSoC 2x2 development board running at a clock rate of 128 MHz and accepting samples output from a DDC at 4 Msps. These results demonstrate the viability and practicality of the proposed architecture for real-time wireless communication applications. In conclusion, the proposed CNN architecture can be considered a promising solution for various time-critical wireless communication applications that demand high throughput and continuous operation.

VII. ACKNOWLEDGEMENTS

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) [EP/R513349/1].

REFERENCES

[1] Ye, H., Li, G. & Juang, B. Power of Deep Learning for Channel Estimation and Signal Detection in OFDM Systems. *IEEE Wireless Communications Letters*. 114-117 (2018)

[2] Shaddad, R., Saif, E., Saif, H., Mohammed, Z. & Farhan, A. Channel estimation for intelligent reflecting surface in 6G wireless network via deep learning technique. *2021 1st International Conference On Emerging Smart Technologies And Applications, ESmarTA 2021*. (2021)

[3] Rahman, M., Shahjalal, M., Ali, M., Yoon, S. & Jang, Y. Deep Learning Based Pilot Assisted Channel Estimation for Rician Fading Massive MIMO Uplink Communication System. *International Conference On Ubiquitous And Future Networks, ICUFN*. pp. 470-472 (2021)

[4] O'Shea, T., Corgan, J. & Clancy, T. Convolutional Radio Modulation Recognition Networks. (2016), http://arxiv.org/abs/1602.04105

[5] Zhang, B., Sui, Y., Huang, L., Liao, S., Deng, C. & Yuan, B. Algorithm and Hardware Co-design for Deep Learning-powered Channel Decoder: A Case Study. *IEEE/ACM International Conference On Computer-Aided Design, Digest Of Technical Papers, ICCAD*. (2021)

[6] Wu, H., Sun, Z. & Zhou, X. Deep Learning-based Frame and Timing Synchronization for End-to-End Communications. *Journal Of Physics: Conference Series*. (2019)

[7] AMD RFSoC-PYNQ. , https://www.rfsoc-pynq.io/

[8] Umuroglu, Y., Fraser, N., Gambardella, G., Blott, M., Leong, P., Jahre, M. & Vissers, K. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. *FPGA 2017 - Proceedings Of The 2017 ACM/SIGDA International Symposium On Field-Programmable Gate Arrays*. pp. 65-74 (2016)

[9] Kumar, S., Mahapatra, R. & Singh, A. Automatic Modulation Recognition: An FPGA Implementation. *IEEE Communications Letters*. 2062-2066 (2022)

[10] Tridgell, S., Boland, D., Leong, P., Kastner, R., Khodamoradi, A. & Siddhartha Real-time automatic modulation classification using RFSoC. *Proceedings - 2020 IEEE 34th International Parallel And Distributed Processing Symposium Workshops, IPDPSW 2020*. pp. 82-89 (2020,5)

[11] Due to legal/commercial issues, data underpinning this publication may be restricted. Further information about the data and conditions for access are available from the University of Strathclyde KnowledgeBase at https://doi.org/10.15129/95f907fb-4cb2-4365-93ac-c36165053999.

[12] Tensorflow. , https://www.tensorflow.org/

[13] MATLAB Fixed-Point Designer., https://uk.mathworks.com/products/fixed-point-designer.html

[14] Bottleson, J., Kim, S., Andrews, J., Bindu, P., Murthy, D. & Jin, J. ClCaffe: OpenCL accelerated caffe for convolutional neural networks. *Proceedings - 2016 IEEE 30th International Parallel And Distributed Processing Symposium, IPDPS 2016*. pp. 50-57 (2016)