IEEE Open Journal of
Circuits and Systems

Received 2 June 2024; revised 7 August 2024 and 30 October 2024; accepted 22 November 2024. Date of publication 3 December 2024; date of current version 6 January 2025.

Digital Object Identifier 10.1109/OJCAS.2024.3509627

RFSoC Modulation Classification With Streaming CNN: Data Set Generation & Quantized-Aware Training

ANDREW MACLELLAN, LOUISE H. CROCKETT, AND ROBERT W. STEWART

StrathSDR, Department of Electronic and Electrical Engineering, University of Strathclyde, G1 1XW Glasgow, U.K.

This article was recommended by Associate Editor T. Kawahara.

CORRESPONDING AUTHOR: A. MACLELLAN (e-mail: a.maclellan@strath.ac.uk)

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

ABSTRACT This paper introduces a novel FPGA-based Convolutional Neural Network (CNN) architecture for continuous radio data processing, specifically targeting modulation classification on the Zynq UltraScale+ Radio Frequency System on Chip (RFSoC) operating in real-time. Evaluated on AMD's RFSoC2x2 development board, the design integrates General Matrix Multiplication (GEMM) optimisations and fixed-point arithmetic. We also present a method for creating Deep Learning (DL) data sets for wireless communications, incorporating the RFSoC into the data generation loop. Furthermore, we explore quantised-aware training, producing three modulation classification models with different fixed-point weight precisions (16-bit, 8-bit, and 4-bit). We interface with the implemented hardware through the open-source PYNQ project, which combines Python with programmable logic interaction, enabling real-time modulation prediction via a PYNQ-enabled Jupyter app. The three models, operating at a 128 MHz sampling rate prior to the decimation stage, were evaluated for accuracy and resource consumption. The 16-bit model achieved the highest accuracy with minimal additional resource usage compared to the 8-bit and 4-bit models, making it the optimal choice for deploying a modulation classifier at the receiver.

INDEX TERMS Deep learning (DL), wireless communications, AMD, FPGA, RFSoC, PYNQ, modulation classification, convolutional neural network (CNN), inference, data-flow, general matrix multiplication (GEMM) transform, data set generation, artificial intelligence (AI), quantized-aware training.

I. INTRODUCTION

DEEP Learning (DL) has emerged as a powerful tool for improving Physical Layer (PHY) wireless communications. As new technologies such as 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], [2], [3], signal identification [4], decoding [5], and synchronization [6]. In other fields that make use of Artificial Intelligence (AI), the typical sizes of DL models can vary depending on the task that is being addressed. For the field of computer vision, applications like object recognition can use models such as the Tiny-YOLO model with 8-million parameters to achieve good results. A model of this size could be run on most desktop graphics cards. Much bigger models are used for Large Language Models (LLMs) such as GPT-3.5 where its 173-billion parameters are distributed amongst 10's of thousands of high end graphics cards. In comparison, DL models used for PHY-layer wireless communications tasks, such as RadioML's modulation recognition model [4], only have ∼260 thousand parameters, making it feasible for them to be deployed on an edge device like a System-on-Chip (SoC) or Field Programmable Gate Array (FPGA).

DL in the realm of communications is rapidly evolving, frequently yielding impressive results. Realizing the full potential of these algorithms requires their integration into both existing and emerging radio transceivers. Many of these transceivers harness FPGA and SoC architectures, used for their reconfigurability and capacity for parallel processing.

These Radio Frequency System on Chip (RFSoC) devices combine high-precision analog-to-digital converters (ADCs) and digital-to-analog converters (DACs), functioning at Giga-samples per second (Gsps), alongside programmable logic for implementation of custom algorithms [7]. This work targets the AMD RFSoC2x2 development board [8] and the hardware designs were built with MathWorks HDL Coder [9] and AMD Vivado.

Our paper builds upon the foundation laid by previous research in modulation classification, drawing upon the seminal work of O'Shea et al. [4] and the innovative low-precision data flow DL models introduced by Umuroglu et al. [10]. Prior studies, such as those by Hou et al. [11] and Jung et al. [12], have explored CNN architectures and Short Time Fourier Transforms (STFT) for modulation classification on FPGAs. However, these investigations primarily focused on simulated or pre-recorded data and did not encompass real-time received signals. In contrast, Horne et al. [13] demonstrated the classification of radio signals in real-time by employing the Line Hough Transform with spectrograms on the ZCU111 RFSoC. Similarly, Tridgell et al. [14] developed a modulation scheme classifier for real-time received signals, leveraging both the RadioML data set [15] and their own recorded signals. Jentzsch et al. [16] introduced a proof of concept paper applying FINN [10] to modulation classification tasks using the RadioML data set [15]. While previous studies have implemented CNN architectures for modulation classification, only Horne et al. [13] and Tridgell et al. [14] have successfully deployed DL models with real-time received signals. It is worth noting that Horne et al. [13] utilized image processing of spectrograms for their classifications, whereas Tridgell et al. [14] focused on classification of samples from the RFSoC ADCs, limiting their scope to 4 modulation schemes with no added noise.

This paper is a continuation from our previous paper [17] where we introduced our streaming-based architecture and deployed a trained DL model. The network was trained using Tensorflow and the floating-point weights were converted into 18-bit fixed point weights and deployed into an RFSoC device, classifying samples at 30dB SNR. It processes samples without interruption live on the board. In this paper, we will detail our process of creating the custom data set used to train the neural network, which takes into consideration any degrading effects introduced by the transceiver and which classifies signals that have passed through a multipath fading channel with AWGN noise of -20 to 30dB SNR. Additionally, we trained three separate DL models at varying weight quantisations, using quantised-aware training with Brevitas [18]. Each model is deployed on the RFSoC and tested with live signals to evaluate its performance. Our novel hardware-friendly CNN architecture includes on-chip storage of weights and preserves the data flow structure of wireless communications pipelines by stream processing of every sample without interruption. In contrast to [14], we use the same neural network dimensions proposed in [4] and utilize quantised-aware training to achieve good classification accuracy on the fixed-point weights. We believe that a method for translating theoretical deep learning models onto hardware without altering the initial network dimensions is crucial for accelerating the transition from theory to production. In this work, we present results obtained from the live classification of signals received at the ADC of the RFSoC. This approach provides a more realistic expectation of performance once a trained network is deployed in hardware.

The contributions of this paper are:

- A novel streaming-based CNN implementation for RFSoC receivers where every sample is processed with no interruptions to the received data stream.
- A method for generating a data set specifically optimized for training RFSoC applications in deployment.
- Results and insights from quantised-aware training of the modulation classification model across different quantisation parameters.
- Demonstration and evaluation of the modulation classification model through live operation on the RFSoC board.

The remainder of this paper is organized as follows: Section II outlines the motivation for the work, Section III introduces the AMD RFSoC development platform, Section IV is an overview of the modulation classification application, Section V details creation of the data set used in this work, Section VI shows the quantised-aware training process, Section VII details the design of the streaming CNN architecture, Section VIII outlines the overall CNN architecture as a whole, and Section IX discusses the infrastructure for testing, validating and visualizing the system. Sections X and XI discuss the resource utilization and accuracy performances of each model. Finally, Section XII concludes and discusses the findings of the paper.

II. MOTIVATION

As 5G and the upcoming 6G technologies unfold, radio transceivers are set to encounter a growing array of challenges beyond the scope of conventional human-designed methods. These challenges can include: channel estimations at higher velocities, sustainable cognitive radios, and smart signal decoding [19]. In response to these emerging complexities, DL emerges as a valuable tool for radio communications engineers. Integrated into existing radio communications pipelines, DL can effectively address these challenges.

Many radio transceiver architectures operate on SoCs or FPGAs. To seamlessly integrate DL into established radio setups, the neural network architecture must maintain a data flow-like structure akin to a wireless communications pipeline. The data-flow model is where signal processing operations are represented as nodes in a graph, with data flowing between them, enabling efficient parallel processing and resource utilization. This flow of data guarantees

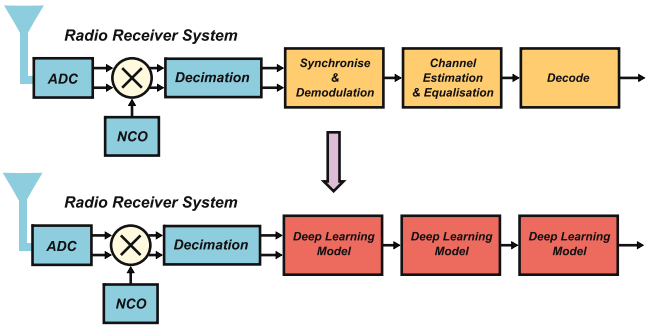

FIGURE 1. Motivation for replacing traditional radio pipeline functionality with deployed deep learning models.

deterministic latency for the neural network, which facilitates seamless alignment and integration with other areas of the radio system. Figure 1 illustrates the potential of DL to augment or replace certain functionalities within an existing data-flow communications pipeline, demonstrating how DL can be integrated while maintaining the flow of data through the design.

The application demonstrated in this paper illustrates that a DL model can be deployed after an ADC to process data without dropping samples. The objective is to establish that DL can be practically and effectively utilised for Physical Layer (PHY) communication systems. We encourage the exploration of other challenging implementations for other DL tasks in wireless communications, such as channel estimation, Dynamic Spectrum Access (DSA), signal decoding, and synchronization. We invite researchers to use our proposed architecture as a reference model to enable DL for these other wireless communication challenges.

III. AMD RFSOC

This study targets AMD's Zynq UltraScale+ Radio Frequency System on Chip (RFSoC) family, specifically the XCZU28DR on the RFSoC2x2 development board [7] (hereafter referred to simply as 'RFSoC'). The XCZU28DR is a Gen1 RFSoC chip that integrates high-precision ADCs and DACs. Its ADCs can achieve a maximum sampling rate of 4.096 Gsps, while the DACs can attain a sampling rate of 6.554 Gsps. Featuring two DACs and two ADCs, the RFSoC2x2's elevated sample rate capabilities open avenues for sophisticated applications across the frequency spectrum, including but not limited to spectrum monitoring and DSA. While this study primarily showcases the utilization of DL for signal identification off the air, the potential extends to fully harness the RFSoC's capabilities in a DSA setup.

The RFSoC facilitates communication between the Processing System (PS), which includes a quad-core Arm Cortex A53 processor, and the Programmable Logic (PL), where custom algorithms can be deployed and accelerated. This work leverages the PYNQ framework for communication between the Arm processor and PL [20]. Developed and maintained by AMD, PYNQ is an open-source framework designed to simplify interactions with SoCs. Employing the Python language, PYNQ seamlessly integrates widely used libraries such as numpy [21], scipy [22], plotly [23], and ipywidgets [24]. Furthermore, PYNQ adopts Jupyter notebooks as an interactive programming environment, a format that is ideal for experimentation and learning [24].

IV. APPLICATION - MODULATION CLASSIFICATION

The focus of this study centers on the task of modulation classification. Modulation classification entails the identification of the modulation scheme employed in a radio signal. In wireless communications, modulation involves encoding information onto a carrier signal for transmission. Modulation classification is a type of signal identification that can aid cognitive radios in spectrum sensing and identifying the primary users of the spectrum. The choice to focus on modulation classification as the application for this work is driven by its prevalence in prior research within the field of DL for wireless communications. Additionally, it offers a foundational benchmark against which other research outputs can be compared, as established in [4].

Within our modulation classifier, the objective is to identify one out of eight possible modulation schemes present in a signal: Quadrature Phase Shift Keying (QPSK), Binary Phase Shift Keying (BPSK), Quadrature Amplitude Modulation (QAM16 & QAM64), 8 Phase Shift Keying (8PSK), Pulse Amplitude Modulation with four levels (PAM4), Gaussian Frequency Shift Keying (GFSK), and Continuous Phase Frequency Shift Keying (CPFSK).

V. DATA SET CREATION - DEEPRFSOC

DeepSig's RadioML data set [15] quickly established itself as a benchmark for evaluating DL architectures within the realm of wireless communications. However, there exists a disparity between the RadioML data set and the actual input received by the ADC of the RFSoC. This difference arises from the absence of additional factors that can deteriorate the signals impacted by the transceiver itself, such as the effects of applying realistic filters that do not have an ideal 'brick wall' response, spurs and harmonics introduced by the data converter, the influence of ADC quantisation, and the possible artifacts associated with the stitching of the interleaved ADCs. Additionally, the RadioML data is provided in fixed length frames of 128 complex samples, yet for real-time classification a continuous data stream is required so that it can be output from the DAC. To address this challenge, it was decided to generate our own data set while utilizing the RFSoC within the generation loop. This is a pivotal step towards creating a data set that closely represents how data passes through the RFSoC, including the analogue and Radio Frequency (RF) processing stages, and ultimately developing a DL network trained with this context in mind.

A. GENERATE TRAINING SAMPLES IN MATLAB

The first step in creating our data set was to generate the desired set of modulated signals and simulate environmental channel effects using MATLAB's Communications

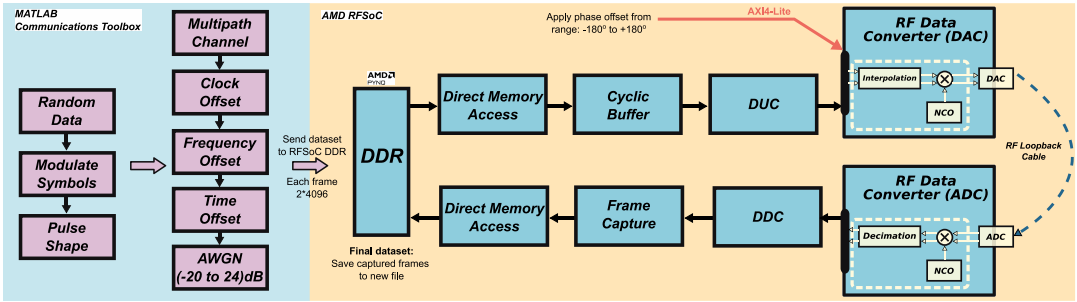

FIGURE 2. Overview of the data set generation process, illustrating the transmission of pre-generated signals using the RFSoC's loop-back path to record the data set.

Toolbox [25]. This approach mirrors the methodology detailed in RadioML [4], where different channel model parameters, frame sizes, and data set sizes were utilised. For each of the eight modulation schemes, 1000 frames of data were generated with 1024 random symbols created for each frame. The symbols were passed through a Raised Cosine pulse-shaping filter with roll-off factor $\alpha = 0.5$, filter length 10, and output samples per symbol equal to 8, resulting in an output length per frame of 8192. The resulting pulse-shaped signal was then passed through a Rician channel model simulating an indoor scenario with two additional reflected paths adding an additional 4.2m and 7.97m to the signal path, respectively. The channel was simulated using the Communications Toolbox with a sampling rate f_s of 128 MHz, three path delays: $\frac{0}{fs}$, $\frac{1.8}{fs}$, and $\frac{3.4}{fs}$ seconds with average path gains: 0, -2, -10, respectively. The K-factor value was set to a scalar of 4 and the maximum Doppler shift equal to 4 Hz while assuming a carrier frequency of 700MHz, simulating a maximum transmitter velocity of 1.7m/s. Subsequently, the channel altered signal has a clock offset simulated, where a random frequency offset between -5 and 5 MHz is applied along with a resulting time offset due to the shifted sampling frequency. Finally, the signal is passed through an Additive White Gaussian Noise (AWGN) channel with varying signal-to-noise ratios (SNR) from the list: $[-20, -16, -12, -8, -4, 0, 4, 8, 12, 16, 20, 24, 28, 30]$ dB. The resulting window is cropped at a random interval to a frame length of 4096, I and Q separated, and the final frame size for each modulation scheme is $1000 \times 2 \times 4096$.

B. TRANSMIT, RECEIVE, AND RECORD

In order to include the RFSoC's signal path as part of the channel the training data passes through, the next step in data set generation was to pass the generated MATLAB data through the RFSoC's DAC and ADC loop-back path. To facilitate the transmission and reception of the pre-generated signals, a bit-stream was designed to transmit the data. An overview of the complete data set generation process, utilizing the RFSoC used for both transmitting and recording the new data set, is depicted in Figure 2.

The data generated in Section V-A was packetised and accessed from within PYNQ with Python [20].

A singular frame from the data set has dimensions 2×4096, and when ready to send, it is stored in PS Double Data Rate (DDR) memory in a pre-allocated contiguous portion of memory. The data is then ready for the Direct Memory Access (DMA) to move it to the PL. In the PL, a buffer waits to receive the data from the DMA. When instructed to, the data is sent to the buffer where the data is stored and read cyclically indefinitely until instructed to stop. This action mimics the transmission of the modulated signal where the data has been altered by a channel. The transmitted signal is then passed through the Digital Up-Conversion (DUC) stages and interpolated to a sampling rate of 128 MHz and passed to the DAC where it is modulated to a desired carrier frequency and sent out of the device. The DAC is connected to the ADC through a loopback cable and the signal is received into the ADC which is tuned to demodulate the signal from the same carrier frequency as applied at the transmit side.

The received signal is passed to the Digital Down Converter (DDC) stages where the sample rate is transformed from 128 MHz to 4 MHz and the resulting signal then enters the Frame Capture Intellectual Property (IP) block. The Frame Capture IP waits for an AXI4-Lite signal from the PS to instruct it to capture the current 128 samples entering the IP. Once the frame has been captured, it is sent to the DMA to be transferred to the PS DDR for storage.

The full data set is created by iterating over all modulation schemes and, for each modulation scheme, iterating over all of the saved SNR noise levels. For each 4096 sample long transmitted signal, the received data is captured 32 times and stored in a new Python dictionary. Figure 2 illustrates this recording process. Once saved, the model is ready to be trained with the data set.

VI. QUANTISED-AWARE TRAINING

Our created data set contains 2.6 million frames across all eight modulation schemes and SNRs, where each frame has 2 channels that are 128 samples long. Each modulation scheme holds 330,000 frames at the varying SNRs. To prepare the data set for training, we first separate the set into training, validation, and testing portions at a 80:10:10 split, respectively. The goal in the training portion of this work is to achieve a DL model that classifies each modulation scheme

TABLE 1. Neural network dimensions.

Layer Type	Dimensions	Activations	Parameters
Input	2×128	-	-
Convolution	$64 \times 3 \times 1$	ReLU	192
Convolution	$16 \times 3 \times 2$	ReLU	6,144
Fully-connected	1984×128	ReLU	253,952
Fully-connected	128×8	Softmax	1,024
Output	1×8	-	

in varying SNR environments and to investigate how the performance of the neural network varies with the number of quantisation levels used for the model weights.

The neural network dimensions in this work remain constant and can be seen in Table 1.

Our neural network structure is similar to the neural network proposed in the original O'Shea paper [4]. We chose to keep the network dimensions similar to demonstrate that no extra neurons were allocated to the quantised networks in comparison to the original floating-point network from [4]. In our previous work [17], we demonstrated the same neural network dimensions with a floating point model and evaluated its performance when the weights and activations were quantised to 18-bits after training. The training framework used in this work is Brevitas [18], a PyTorch fork [26], which is a framework designed for implementing reduced precision hardware data-paths during the training of neural networks. Brevitas extends PyTorch's capabilities by incorporating various options such as quantisation techniques, adjustable bit-widths, and customisable hardware configurations. During training, the quantised networks apply a limitation on the dynamic range of each weight/activation depending on the user's configuration. Quantised-aware training allows for the network to factor in the noise incurred from the quantised weights/activations themselves as part of the training loop, as well as apply a form of regularization through quantisation noise. This results in much better performance in accuracy than quantizing post-training [27], [28], [29], [30].

Three sets of quantisation parameters are evaluated for this paper: 16-bit weights with 16-bit activations, 8-bit weights with 16-bit activations, and 4-bit weights with 16-bit activations. In Brevitas, each model is configured to represent the dimensions shown in Table 1 using a mixture of PyTorch layers and Brevitas layers.

Figure 3 plots the loss for each network quantisation against the number of epochs during training. For the training sequence the loss was calculated using cross entropy loss [31] with an Adam optimizer [32] with parameters: learning rate $= 1e^{-4}$. The batch size was set to 128 and the training sequence was run for 70 epochs unless early stopping was triggered. Early stopping was triggered if the loss did not improve over 8 epochs, as can be seen in the loss plots in Figure 3. For the duration of training, the validation loss was also logged as an indicator of whether the training was overfitting to the data set.

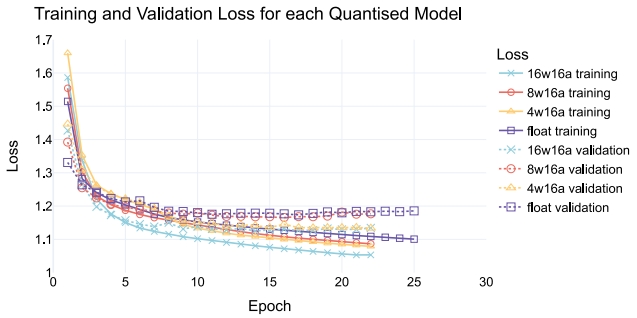

FIGURE 3. Loss over epochs during training sequence for all DL models.

VII. 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 PyTorch/Brevitas 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 based on a finite frame of data, we have adapted the layers (see Section VII-A) to allow for samples to flow through uninterrupted. This means that the implemented DL model's latency is deterministic and therefore can be synchronized 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 our streaming architecture is centred on the data-flow model, treating incoming data as a continuous stream of infinite samples and processing them in real-time. One significant challenge with this approach is the repeated reading of input feature map samples by the convolutional layers, which can result in a potential loss of samples due to processing lag. Given an input feature map with dimensions: channels c, height h, and width w, and a filter with dimensions: number of filters n, matching input channels c, height j, width k, and the number of strides s used during convolution (as defined in Equation (1)), Equation (2) shows the minimum clock rate increase factor R required for each convolutional layer. For instance, the first convolutional layer processes an input with dimensions $(c, h, w) = (1, 2, 128)$ and a filter with dimensions $(n, c, j, k) = (64, 1, 1, 3)$. The clock rate increase factor for this layer, denoted as R_{conv1}, is given by

$$s = (h - j + 1) * (w - k + 1) \tag{1}$$

$$R_{\text{conv1}} = \left\lceil \frac{scjk}{chw} \right\rceil = \left\lceil \frac{252 * 1 * 1 * 3}{1 * 2 * 128} \right\rceil = \left\lceil \frac{756}{256} \right\rceil = 3 \tag{2}$$

The relationship given in (2) can be applied to all convolutional layers in the network, two in our case, and an additional factor of 2 is added due to the upsampling and interleaving stage before data enters the implemented DL model. The overall upsampling ratio required for the

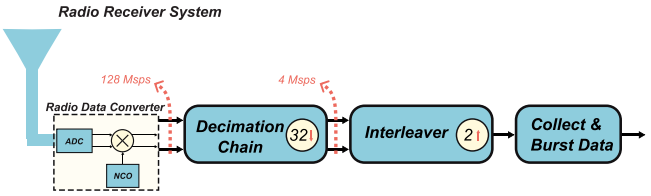

FIGURE 4. CNN input data pre-processing.

network, assuming a fully parallel matrix-vector multiplication is implemented, is given by

$$R_{\text{total}} = R_{\text{conv1}} + R_{\text{conv2}} + 2 \tag{3}$$

where R_{conv1} and R_{conv2} are both equal to 3. The system necessitates a minimum clock increase by a factor of 8. In this research, we elevated the clock rate by a factor of 32 to enable effective hardware resource sharing in the fully connected layers. The proposed DL model follows a DDC, reducing the incoming signal's sampling rate by a factor of 32 from the ADC fabric clock of 128 MHz. The ADC samples at 1.024 Gsps with a PLL clock frequency of 409.6 MHz and an internal decimation rate of 8.

The following sections will describe the design choices and methodologies employed in implementing our CNN architecture on the RFSoC.

A. INPUT DATA PRE-PROCESSING

The received IQ signal from the antenna in Figure 4 is digitized in the ADC and modulated to baseband before entering the FPGA at a rate of 128 Msps. To minimize 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 two low-pass decimation filters with combined 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 to form a one-dimensional input before entering the first convolutional layer.

After the decimation stage, the data is fed into the DL model at a lower sample rate than the model's clock rate. To prevent the CNN from waiting extra clock cycles to receive subsequent samples, we buffer the incoming data into Block RAM (BRAM) and output the data in 256-sample long bursts, clocked at the DL model's clock rate. This bursting operation acts as a ping-pong buffer and it is used to prevent any data loss during conversion. While one buffer fills, the other outputs the previously collected samples. The incoming I and Q samples are quantised to `int16` precision.

B. CONVOLUTIONAL LAYERS

Convolutional layers are the core building blocks of CNNs, pivotal in extracting feature maps from input data. They do this by sliding filters (or kernels) across the input, and computing the dot product between filter weights and overlapping input values. This process yields a feature map where localized features within the input can be identified. However, convolutional layers pose a computational challenge due to the their inherent complexity of passing filters over the input multiple times. For a streaming-based CNN architecture, this repeated operation can greatly slow down the throughput of the deployed model. To circumvent this, our architecture incorporates the General Matrix Multiplication (GEMM) Transform, where it streamlines convolutional operations by condensing the computation into a simple large matrix-matrix multiplication [33].

Within our architecture, convolutional layers are divided into two stages: the input data buffer and sliding window generator, and the matrix-to-vector multiplier. The input data buffer receives and stores samples in a 1-dimensional on-chip RAM. The sliding window generator executes the GEMM transform on the input samples, retrieving them from the buffer in a transformed order for multiplication with pre-processed layer filters stored in BRAM. This generator outputs the resulting matrix incrementally, sample by sample or row by row, and conducts multiplication with the GEMM transformed filter weights to generate the layer output.

The following subsections describe the two stages within the convolutional layers in more detail.

1) GEMM TRANSFORM AND SLIDING WINDOW

After the samples are buffered, the Sliding Window Controller (SWC) proceeds to read the GEMM-transformed samples.

a) Filter/Kernel Transformation: The quantised filter/kernel weights undergo transformation into a matrix representation. Initially, the interleaving of C channels for each filter N takes place. Subsequently, the unrolling and concatenation of each filter N results in a single matrix, denoted as $\tilde{\Theta}^{\text{conv}}$, with dimensions $N \times CKJ$ (where N represents the number of filters, C denotes the number of channels, and J and K denote the number of columns and rows of the filter weights, respectively). This transformation is conducted offline in MATLAB, and the filter weights are stored on-chip for efficient access.

b) Input Data Transformation: The GEMM transform of input samples into a convolutional layer comprises three steps:

- Initially, the input is reshaped into a 2-dimensional matrix, with C interleaved channels. Typically, the input to a layer is a 3-dimensional tensor.
- Striding and replication functions are applied (see Figure 5).
- Unrolling. For each stride step, the input samples under the filter window are unrolled and concatenated, resulting in the matrix $\tilde{X}^{\text{conv}}$.

In contrast to filter weights, the input transformation must be performed during live operation. The drawback of this approach is the necessity to replicate input samples to

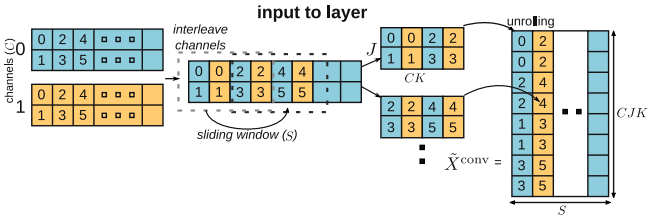

FIGURE 5. GEMM Transform algorithm for the input samples.

achieve the same output as a sliding window convolution. To address this, we implemented a SWC capable of performing the transform in real-time on a continuous stream of input samples.

The Buffer & SWC (depicted in the wider model architecture in Figure 6), manages the reading and writing of samples to on-chip RAM. It incorporates a state machine responsible for executing the sliding window transformation (as illustrated in Figure 5), by reading samples from the buffer memory according to the GEMM transform.

Initially, the columns (K) of each input channel (C) are interleaved to create a single 2-dimensional matrix. Next, the flattened sliding window passes over the 2-dimensional input, dividing it into smaller matrices based on the number of strides (S) performed by the sliding window. The split matrices are then unrolled and concatenated to yield the resulting GEMM-transformed input matrix, $\tilde{X}^{\text{conv}}$, with dimensions $CJK \times S$. The SWC directs storage addresses to the on-chip RAM to retrieve the resulting matrix.

2) MATRIX TO VECTOR MULTIPLICATION

After the input undergoes transformation into a matrix, it proceeds to the matrix-vector multiplier stage where it is multiplied with the transformed filter weights to generate the convolution output.

Within this proposed architecture, we introduce various implementations of the matrix-vector multiplier. The first type of implementation entails parallel processing of vector-vector multiplications, utilizing input vectors from the SWC. The second option executes the matrix-vector calculation sample-by-sample employing N Multiply-ACcumulate (MAC) units, where N represents the number of filters in the convolutional layer. This latter design is particularly optimized for larger multiplications. Lastly, an alternative option involves a serialized implementation of the matrix-vector multiplier, time-sharing one (or a limited number of) MAC units, provided that sufficient spare clock cycles exist.

All implementations maintain 16-bit fixed-point precision at the outputs of each layer, and carefully minimize the wordlength growth through the MAC operations. All of the previously described multiplication optimisations necessitate a higher sampling rate than the input rate of the model, driven by the need to replicate samples in the GEMM transform stage and the increased number of samples output in certain convolutional layers. The clock rate is further increased if any time-sharing of MAC units is possible to save on FPGA resources. The convolutional filters are stored on-chip using one of the three quantisation formats explored in this paper: 16, 8, or 4 bits.

C. DENSE LAYERS

A dense layer in DL models is a fully-connected layer where every neuron is intrinsically linked to each neuron in the preceding layer. The dense layer executes a matrix multiplication operation between the inputs and weights.

Since the dense layer can be calculated as a multiplication of two matrices, the implementation approach can be similar to that described in Section VII-B2 for the convolutional layers. The matrix-vector multiplication can be implemented in a number of ways for resource saving and throughput optimisations, while keeping a 16-bit fixed-point precision at the output. Like the filters in the convolutional layer, the weights for the dense layers are stored on-chip in one of the three quantisation formats investigated in this paper (16, 8, and 4-bits). Unlike the convolutional layers, the weights and the inputs do not require any transformations.

D. ACTIVATIONS

In this work, we focus solely on the effects of quantizing the weights of each layer, deliberately excluding biases to isolate this effect.

The activation function employed in the first three layers of our proposed network is the Rectified Linear Unit (ReLU) function [34]. For the final layer, a Softmax function [35] was used to transform the classification outputs into a probability distribution.

VIII. OVERALL ARCHITECTURE

Figure 6 provides an overview of our hardware data flow CNN architecture. Point 'A' serves as the input for the signal originating from the ADC, decimation filter chain, and interleaver, all passing into the first convolutional layer. A buffer stores 256 samples before the SWC reads the samples back out while applying the GEMM transform. The samples then enter the matrix-vector multiplier block, an assembly of 64 MAC units, each producing an output every 3 clock cycles. Each MAC unit multiplies incoming samples with their corresponding weights, stored in the BRAM. The output dimensions exiting from this multiplier stage are formed into a vector containing 64 samples. At point 'B' the vector then enters the second convolutional layer and is stored in an array of buffers. The SWC performs the GEMM transform, extracting 16 sample long vectors from the array of buffers. The GEMM-transformed input, with dimensions 124×384, undergoes multiplication with transformed weights (384×16). The matrix-vector multiplier anticipates a 16 sample vector from the buffer, requiring $2,976$ vectors to complete one input frame.

The second convolutional layer matrix-vector multiplier adopts a configuration featuring 16 groups of 16 MAC units (256 in total), each operating for 24 clock cycles to produce

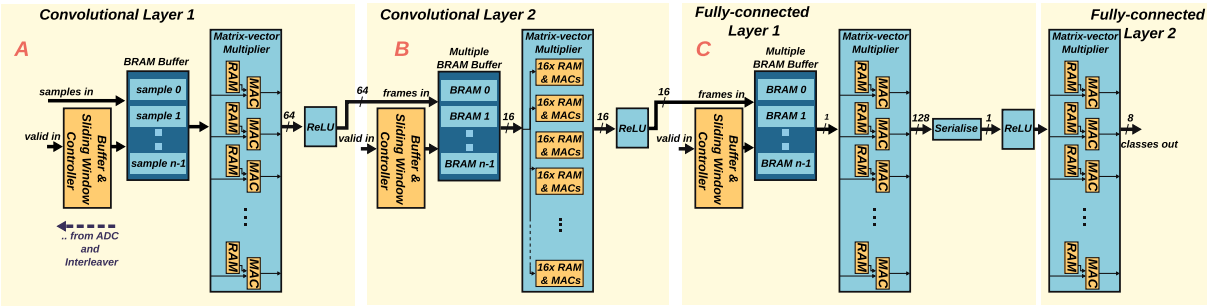

FIGURE 6. Overall CNN architecture for modulation classification.

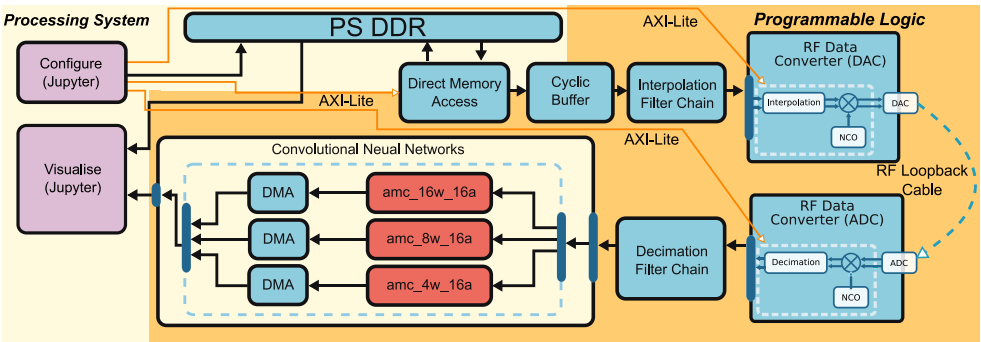

FIGURE 7. High-level system showing PS-PL communications.

TABLE 2. Parallelism of MACs in each layer.

Layer	Parallel MACs	Number of samples per output	Unrolled dimension
Conv 1	64	3	Conv filters
Conv 2	256	24	Conv filters
FC 1	128	1984	Output dim
FC 2	8	128	Output dim

an output. At point 'C', the vectors from the second layer are stored in another vector buffer, with the SWC extracting samples one at a time. These samples are passed through 128 parallel MAC units, operating over $1,984$ samples to generate the third layer output.

Up to this stage, samples pass through a ReLU function between layers, zeroing negative values. The final Fully-Connected (FC) layer computes the outputs using 8 MACs, without a transform or associated buffers. The layer's outputs, representing class predictions, are serialized and sent to a DMA IP for PS visualization. The parallelism of MAC units is summarized in Table 2.

IX. VALIDATION AND VISUALIZATION INFRASTRUCTURE

A validation and visualization infrastructure (Figure 7) was created in order to test the performance of the CNN models deployed on the PL. The CNN models were connected to the ADC of the Radio Frequency Data Converter (RFDC) through a decimation filter chain with the classification outputs of the models connected to DMA IPs to transfer to the classifications to the PS for visualization. We configured the RFDC to transmit modulation schemes at a center frequency (f_c) of 400 MHz. On the receiver side, the signal is received to the ADC via the RF loopback cable.

Figure 8 is a screenshot from the Jupyter Labs environment where a user interface was created to inspect the implementation through *ipywidgets* [24]. All three models were implemented into one FPGA bitstream to permit direct comparison of the performance of the quantised models (`amc_16w16a`, `amc_8w16a`, and `amc_4w16a`) which all operate on the same received signal. When a test signal is sent to the ADC and decimated through the DDC, the signal is split into three paths before it is passed into each model. The full system setup can be seen in Figure 7 showing the connections between the PS and PL. The models calculate their classification answers and output an 8-sample stream to a DMA IP block that passes data to the PS for plotting and analysis. The plots implemented in the user interface app are confusion matrices and classification confidence bar graphs. The confusion matrices compare the predicted label generated by the CNN models to the true label of the signal, and the confidence bar graphs illustrate the softmax result of each prediction.

The app in Figure 8 requires the user to choose the desired modulation scheme from a drop-down menu followed by a choice of SNR. Lastly a phase offset can be applied to the signal that is configured from within the RFDC. The 4,096-sample long transmitted signal is plotted on the top left. When the ADC receives the signal and captures the 128 complex samples, the time-series received signal is plotted on the top right of the app (this is the signal that is passed into each of the DL models). The models

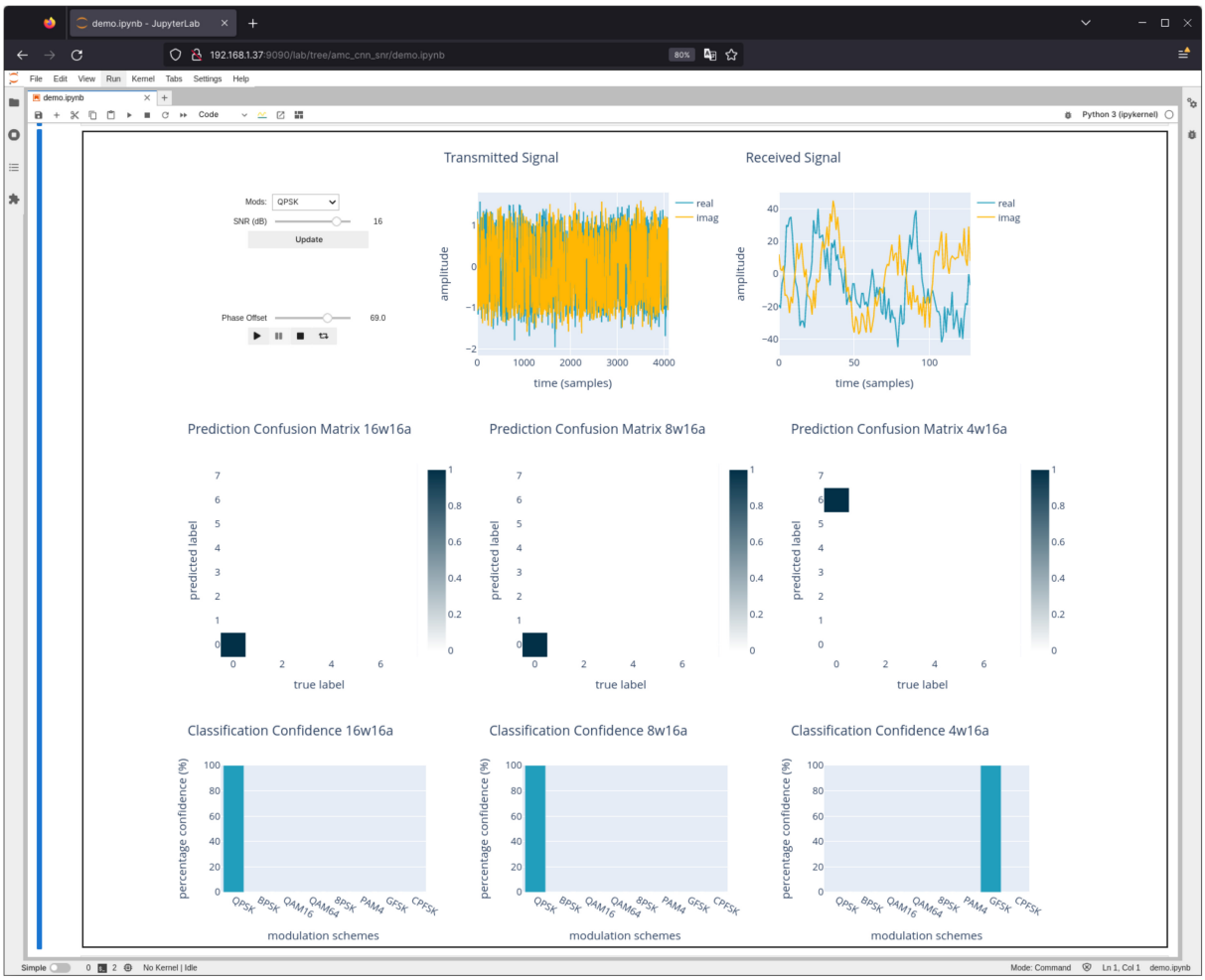

FIGURE 8. ipywidgets app in Jupyter Labs for testing three models live on the RFSoC.

automatically calculate the classification of the signal and show the prediction and confidence values in each of the plots. In the example shown in Figure 8 a QPSK modulated signal with a noise level of 12dB is sent to the receiver where all three models show a prediction. The models with 16-bit and 8-bit weights successfully predict the modulation scheme correctly, however, the 4-bit model predicts the signal incorrectly with high confidence. The ipywidgets app also provides a 'play' button that enables continuous transmission of the selected modulation scheme and periodically updates the prediction plots. The frequency at which predictions are visualized depends on how quickly data can be moved from the PL, rather than on the latency of the DL models.

X. RESOURCE UTILIZATION

The system was evaluated on the AMD RFSoC2x2 development board which features the Zynq UltraScale+ XCZU28DR RFSoC part. Table 3 presents the resource utilization for the DL models. Each model utilisers approximately 10% of the available DSP slices and 15% of the available BRAMs for each model. The DSP slices are used in the matrix-vector multiplication stages for each layer and the BRAMs store the samples entering the convolutional layers as well as storing the on-chip weights. The table shows the resources used by each layer of each model. The main difference lies in the utilization of BRAMs for storing on-chip weights and using an UltraRAM to store the

TABLE 3. FPGA resource utilization of each quantised CNN model.

Model *layer*	Slice (LUTs)	Slice (Register)	DSPs	BRAMs	UltraRAMs
16w16a	**26976**	**39791**	**456**	**169**	**1**
conv1	3245	4268	64	0	1
conv2	14822	20187	256	32	0
dense1	7826	14282	128	136	0
dense2	754	789	8	0	0
8w16a	**24045**	**35602**	**456**	**105**	**1**
conv1	4588	5374	64	0	1
conv2	11718	15931	256	32	0
dense1	6921	13372	128	72	0
dense2	564	659	8	0	0
4w16a	**21930**	**32803**	**456**	**105**	**1**
conv1	4233	4916	64	0	1
conv2	10436	14034	256	32	0
dense1	6531	12988	128	72	0
dense2	476	599	8	0	0

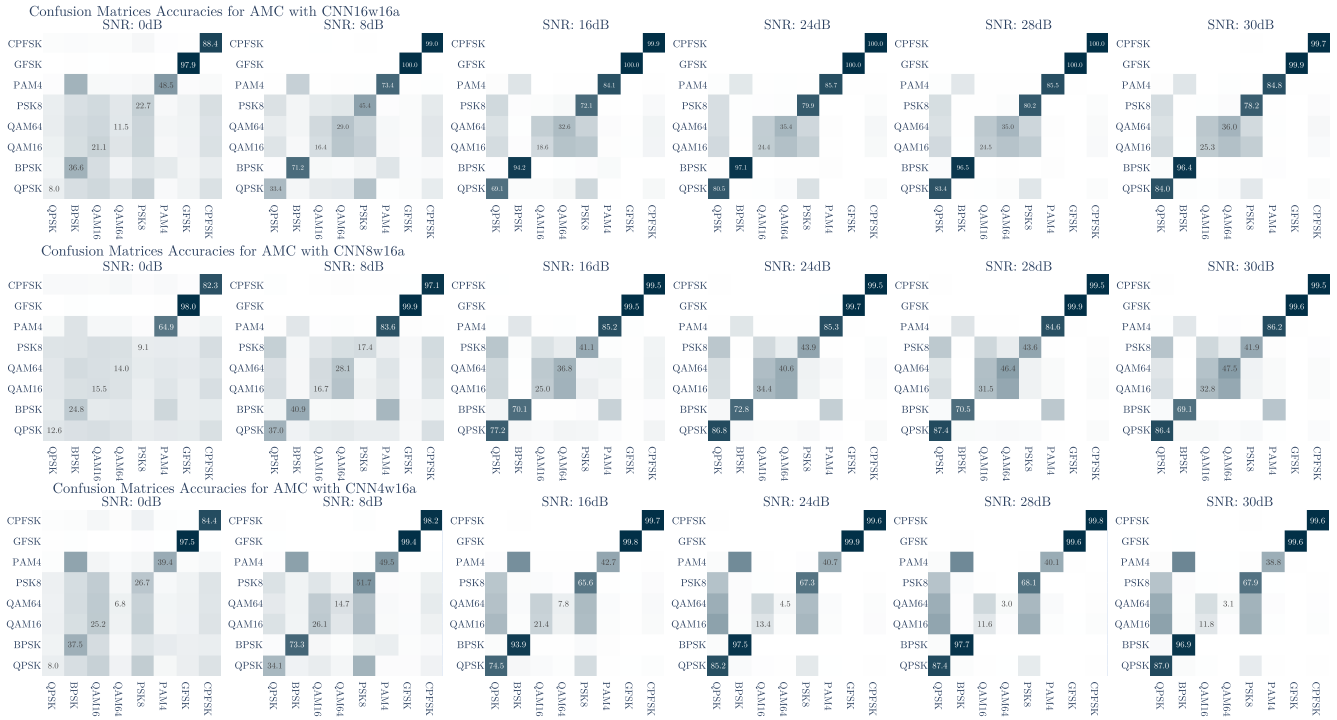

FIGURE 9. Confusion matrices of accuracies across various SNRs.

initially received samples. The 8-bit and 4-bit models have a reduced BRAM footprint compared to the 16-bit model due to their lower bit-width per weight. However, because HDL Coder [9] processes data in byte increments, the 4-bit model does not save additional memory over the 8-bit model, resulting in the same resource usage for both. Additionally, a noticeable difference arises in the consumption of LUTs across models, reducing as the weight bit-width decreases. This reduction stems from the diminished necessity for logical operations in accessing smaller weights from the BRAMs.

XI. PERFORMANCE RESULTS

The system's performance is evaluated by each quantised model's ability to predict modulation schemes across various noise levels, as shown in Figure 9 through a series of confusion matrices across different noise levels. Signals with SNR levels of 0dB, 8dB, 16dB, 20dB, 24dB, and 30dB were passed into the model and the overall accuracy for each noise level was recorded. The 16-bit model shows poor performance at low SNR but achieves 80-100% accuracy for each modulation scheme at the higher SNR levels, though it struggles to differentiate QAM16 and QAM64 due to their similarity. At 0dB, it quickly learns identify GFSK before all other modulation schemes. On average, the model achieves an accuracy of 76% at the highest SNRs.

In the 8-bit weight model, a similar performance to the 16-bit case can be seen, though with an overall decrease in accuracy. At lower SNRs (0dB) the model again identifies GFSK before the other modulation schemes, and at the highest SNR levels of 20dB and above, the model maintains an average 70% accuracy, approximately 7% reduction to the 16-bit model due to the reduction is dynamic range of the model weights. In the 4-bit model we can see further effects of the limited dynamic range, where the accuracy of the model plateaus at around 63% from 20dB SNR and onwards. Despite quantised-aware training, the 8-bit and 4-bit models do not achieve equal accuracy performance to the 16-bit weight model.

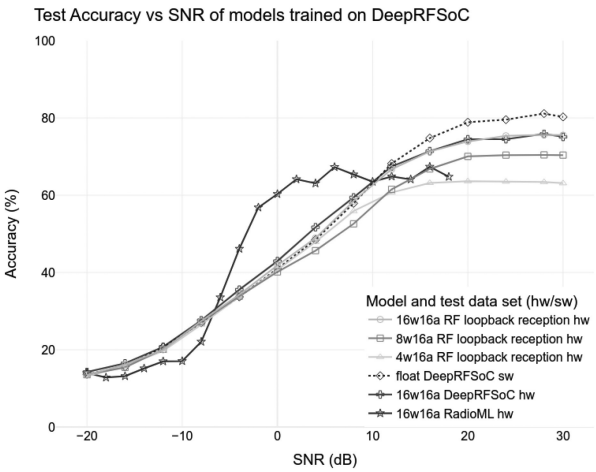

FIGURE 10. Accuracy plots for quantised models across SNRs, comparing real-time signals, a floating-point model tested with DeepRFSoC, and the 16w16a model tested on hardware with DeepRFSoC and RadioML data sets.

Figure 10 shows the average accuracy across SNRs for a series of trained and deployed models. Firstly, the average accuracy across all modulation schemes for each SNR point

TABLE 4. Comparison with CNN accelerators for modulation classification.

Accelerator	This work (16w16a)	Tridgell et al. [14]	FINN [16]	Hou et al. [11]	Jung et al. [12]
# Params	260k	636k	161k	216m	198k
Topology	2 conv/2 fc	VGG10-L (128, 512)	VGG10 (64, 128)	VGG-16	3 conv/2 fc
Quantisation (weight/act.)	16b/16b	2b/incr. prec.	4b/4b	Not Reported	16b/16b
Latency (μs)	29.6	8	11.7	234	12.4
Noise (dB)	-20–30	None	-20–30	-2–10	-20–30
Channel	Multipath + RFSoC	None	Multipath	None	Multipath
Clock rate	128 MHz	250 MHz	250 MHz	Not Reported	100 MHz
Accuracy (@ SNR dB)	76% @>20dB	80.2% @30dB	94.1% @30dB	92.36% @6dB	75% @>0dB
Throughput (cps)	34k	488k	120k	Not Reported	Not Reported
Results from live RF	Yes	Partial	No	No	No

is shown for each of the quantised models while it is receiving live data via RF loopback. It is shown that the best performance is that of the 16-bit weight model (16w16a). As a comparison, the DeepRFSoC data set, used to train the 16-bit quantised model, was evaluated with the deployed model by sending test frames via DMA transfer to the model. It can be seen that the 16w16a model performs equally when receiving data from the ADC or via DMA transfer. A floating-point equivalent model was trained in PyTorch, with accuracies plotted in Figure 10, to provide a baseline accuracy for the models. The floating-model model achieves an accuracy of 80% at the highest SNRs, showcasing that the deployed 16w16a model achieves a 4% accuracy reduction to the baseline. Finally, the 16w16a model trained on DeepRFSoC was sent test frames from the RadioML data set to evaluate the model's performance on other data sets that it was not trained on. As shown in Figure 10, the model reaches an average accuracy of 65% for SNRs above 4dB on the unseen RadioML data. Notably, model accuracy improves at lower SNRs when processing RadioML data compared to DeepRFSoC. This difference may be due to the additional channel effects introduced by the RFSoC transmission and reception path. Furthermore, the Rician channel model used in our work (as defined in Section V-A) differs from the one in [4], which could result in less signal disruption, making the RadioML channel easier for the model to learn.

Table 4 compares our work with other CNN accelerators designed for modulation scheme classification. Unlike other studies, our results are based on real-time classification of signals received live from the ADC. We believe it is crucial to benchmark CNN models in their intended operational environments. Other works have typically used test sets from the same distribution as the training set, with only [14] partially achieving live operation with four modulation schemes. Our model's throughput is lower compared to others, as it is designed to match the speed of samples received from the ADC+DUC stages. However, this throughput was sufficient for our experiment, as no samples were dropped. To improve latency, scheduling layer computations, as seen in [10], [14], could reduce idle times and enhance MAC parallelism.

XII. CONCLUSION

In this paper, we presented a novel streaming CNN architecture designed to operate with RFSoC radio receivers. The model processes samples received from the ADC stages seamlessly without dropping samples, making it very suitable for integration into wireless communications pipelines. We also presented a method for creating a data set for modulation classification where the AMD RFSoC was used as part of the training data set generation process. Three models were trained on the data set using quantised-aware training and the results of each model were compared to show the performance difference of storing weights with different dynamic ranges. Experimental results showed that the 16-bit, 8-bit, and 4-bit models achieved 76%, 70% and 63% accuracy while receiving data in real-time, respectively. The results showcase the effectiveness of quantised-aware training for deployment of CNNs on FPGA receivers. The results demonstrate the practicality of the proposed CNN architecture and provide insight into how the dynamic range of the stored weights affects the trained network. This work is available on GitHub[1] and associated data set [36].

In summary, this paper introduced three key contributions: a novel streaming-based CNN implementation for RFSoC receivers, a method for crafting data sets suited for real-world RFSoC applications, and insights from quantised-aware training of modulation classification models. These advancements offer promising avenues for improving the efficiency and effectiveness of RFSoC systems in PHY layer wireless communications settings.

We would like to extend our gratitude to AMD for their software and hardware support of the project.

REFERENCES

[1] H. Ye, G. Y. Li, and B.-H. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 1, pp. 114–117, Feb. 2018.

[2] R. Q. Shaddad, E. M. Saif, H. M. Saif, Z. Y. Mohammed, and A. H. Farhan, "Channel estimation for intelligent reflecting surface in 6G wireless network via deep learning technique," in *Proc. 1st Int. Conf. Emerg. Smart Technol. Appl. (eSmarTA)*, 2021, pp. 1–5.

[3] M. H. Rahman, M. Shahjalal, M. O. Ali, S. Yoon, and Y. M. Jang, "Deep learning based pilot assisted channel estimation for Rician fading massive MIMO uplink communication system," in *Proc. 12th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, 2021, pp. 470–472.

[4] T. J. O'Shea, J. Corgan, and T. C. Clancy, "Convolutional radio modulation recognition networks," 2016, *arXiv:1602.04105*.

[5] B. Zhang, Y. Sui, L. Huang, S. Liao, C. Deng, and B. Yuan, "Algorithm and hardware co-design for deep learning-powered channel decoder: A case study," in *IEEE/ACM Int. Conf. Comput.-Aided Design Tech. Dig.*, 2021, pp. 1–6.

[6] H. Wu, Z. Sun, and X. Zhou, "Deep learning-based frame and timing synchronization for end-to-end communications," *J. Phys. Conf. Ser.*, vol. 1169, Nov. 2018, Art. no. 12060.

[7] "RFSoC-PYNQ." AMD. Accessed: May 2024. [Online]. Available: http://www.rfsoc-pynq.io

[1] https://github.com/axdy/rfsoc_quant_amc

[8] "RFSoC 2x2 kit." AMD. Accessed: May 2024. [Online]. Available: https://www.amd.com/en/corporate/university-program/aup-boards/rfsoc2x2.html
[9] "HDL coder." MathWorks. Accessed: May 2024. [Online]. Available: https://uk.mathworks.com/products/hdl-coder.html
[10] Y. Umuroglu et al., "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programm. Gate Arrays (FPGA)*, 2016, pp. 65–74.
[11] C. Hou, C. Fang, Y. Lin, Y. Li, and J. Zhang, "Implementation of a CNN identifing modulation signals on an embedded SoC," in *Proc. IEEE 63rd Int. Midwest Symp. Circuits Syst. (MWSCAS)*, 2020, pp. 490–493.
[12] K. Jung, J. Woo, and S. Mukhopadhyay, "On-chip acceleration of RF signal modulation classification with short-time fourier transform and convolutional neural network," *IEEE Access*, vol. 11, pp. 144051–144063, 2023.
[13] C. Horne, N. J. Peters, and M. A. Ritchie, "Classification of LoRa signals with real-time validation using the Xilinx radio frequency system-on-chip," *IEEE Access*, vol. 11, pp. 26211–26223, 2023.
[14] S. Tridgell, D. Boland, P. H. W. Leong, R. Kastner, A. Khodamoradi, and Siddhartha, "Real-time automatic modulation classification using RFSoC," in *Proc. IEEE 34th Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2020, pp. 82–89. [Online]. Available: https://ieeexplore.ieee.org/document/9150443
[15] "Datasets RadioML," Dataset, DeepSig. Accessed: May 2024. [Online]. Available: https://www.deepsig.ai/datasets
[16] F. Jentzsch, Y. Umuroglu, A. Pappalardo, M. Blott, and M. Platzner, "RadioML meets FINN: Enabling future RF applications with FPGA streaming architectures," *IEEE Micro*, vol. 42, no. 6, pp. 125–133, Nov./Dec. 2022.
[17] A. Maclellan, L. H. Crockett, and R. W. Stewart, "Streaming convolutional neural network FPGA architecture for RFSoC data converters," in *Proc. 21st IEEE Interreg. NEWCAS Conf. (NEWCAS)*, 2023, pp. 1–5.
[18] "Brevitas: Neural network quantization in PyTorch." AMD. Accessed: May 2024. [Online]. Available: https://github.com/Xilinx/brevitas
[19] T. S. Rappaport et al., "Wireless communications and applications above 100 GHz: Opportunities and challenges for 6G and beyond," *IEEE Access*, vol. 7, pp. 78729–78757, 2019.
[20] (AMD, Santa Clara, CA, USA). *PYNQ—Python Productivity for Zynq*. Accessed: May 2024. [Online]. Available: http://www.pynq.io
[21] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.
[22] P. Virtanen et al., "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nat. Methods*, vol. 17, pp. 261–272, Feb. 2020.
[23] (Plotly, Montreal, QC, Canada). *Plotly: Low-Code Data App Development*. Accessed: May 2024. [Online]. Available: https://plotly.com
[24] "Jupyter widgets 8.1.2 documentation." Jupyter. Accessed: May 2024. [Online]. Available: https://ipywidgets.readthedocs.io/en/latest
[25] (MathWorks, Natick, MA, USA). *Communications Toolbox*. Accessed: May 2024. [Online]. Available: https://uk.mathworks.com/products/communications.html
[26] "PyTorch." Accessed: May 2024. [Online]. Available: https://pytorch.org
[27] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, 2017, pp. 1921–1925.
[28] "Quantization aware training with TensorFlow model optimization toolkit—Performance with accuracy." TensorFlow. Accessed: May 2024. [Online]. Available: https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html
[29] J. Chen, Y. Gai, Z. Yao, M. W. Mahoney, and J. E. Gonzalez, "A statistical framework for low-bitwidth training of deep neural networks," 2020, *arXiv:2010.14298*.
[30] A. Maclellan, L. McLaughlin, L. Crockett, and R. Stewart, "FPGA accelerated deep learning radio modulation classification using MATLAB system objects & PYNQ," in *Proc. 29th Int. Conf. Field Programm. Logic Appl. (FPL)*, 2019, pp. 246–247.
[31] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.
[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017, *arXiv:1412.6980*.
[33] J. Bottleson, S. Kim, J. Andrews, P. Bindu, D. N. Murthy, and J. Jin, "clCaffe: OpenCL accelerated Caffe for convolutional neural networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, 2016, pp. 50–57.
[34] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biol. Cybern.*, vol. 20, no. 3, pp. 121–136, Sep. 1975.
[35] J. S. Bridle, "Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 211–217.
[36] A. MacLellan, 2023, "Training dataset for RFSoC modulation classification," Dataset, University of Strathclyde Glasgow. [Online]. Available: https://pureportal.strath.ac.uk/en/datasets/95f907fb-4cb2-4365-93ac-c36165053999

ANDREW MACLELLAN received the M.Eng. degree (Distinction) in electronic and electrical engineering from the University of Strathclyde, Glasgow, U.K., in 2018, where he is currently pursuing the Ph.D. degree with StrathSDR Research Group, and also a Research Assistant. His research interests include deep learning (DL) for PHY layer wireless communications and DL inference on FPGAs. He has interned in the Wireless HDL Toolbox team at MathWorks Glasgow for three separate internships. From 2020 to 2021 he also interned with the PYNQ development team at AMD (formerly Xilinx).

LOUISE H. CROCKETT received the M.Eng. (Distinction) and Ph.D. degrees in electronic and electrical engineering from the University of Strathclyde, in 2003 and 2008, respectively, where she is currently a Senior Teaching Fellow and a Senior Member with StrathSDR Research Team and she supervises and manages researchers and key sponsored projects. She has previously co-authored three books on Xilinx/AMD Technology. Her teaching focuses on digital systems design targeting FPGAs and SoCs, and builds practical skills to equip graduates for roles in industry. Her core research interests are in the implementation of DSP systems, FPGAs and SoCs, wireless communications, and SDR.

ROBERT W. STEWART is a Professor with the Department of Electronic Engineering, University of Strathclyde, where he leads the 'StrathSDR' team working on software defined radio and next generation radio access networks using shared spectrum with dynamic spectrum access. He has led a number of 5G Testbed and Trials projects and in recent years he has been working on solutions for the media and broadcast industry and private 5G SA networks working alongside a number of international broadcasters. Over a 30 year career he has published 4 books and more than 200 papers. He is also a Director of the University startup company Neutral Wireless Ltd.