# Real-time Automatic Modulation Classification using RFSoC

Stephen Tridgell[1], David Boland[1], Philip H.W. Leong[1], Ryan Kastner[2], Alireza Khodamoradi[2], and Siddhartha[1]
[1]The University of Sydney, [2] University of California, San Diego
Email: {stephen.tridgell, david.boland, philip.leong, siddhartha.siddhartha}@sydney.edu.au
{kastner@ucsd.edu, alirezak@eng.ucsd.edu}

***Abstract*—The computational complexity of deep learning has led to research efforts to reduce the computation required. The use of low precision is particularly effective on FPGAs as they are not restricted to byte addressable operations. Very low precision activations and weights can have a significant impact on the accuracy however. This work demonstrates by exploiting throughput matching that higher precision on certain layers can be used to recover this accuracy. This is applied to the domain of automatic modulation classification for radio signals leveraging the RF capabilities offered by the Xilinx ZCU111 RFSoC platform. The implemented networks achieve high-speed real-time performance with a classification latency of ≈8$\mu$s, and an operational throughput of 488k classifications per second. On the open-source RadioML dataset, we demonstrate how to recover 4.3% in accuracy with the same hardware usage with our technique.**

## I. Introduction

Automatic Modulation Classification (AMC) is an important method that can be used in applications such as cognitive radios. The high level goal is to monitor the RF spectrum and determine the different modulations being used. A cognitive radio subsequently uses this information to more efficiently transmit its own data. Ideally, modulation classification is done with low latency, so that decisions can be quickly made based upon the current situation.

Recent results show that neural networks are effective in performing modulation classification with high accuracy [9], [14], [23]. However, these results ignored any real-time demands and used networks that are computationally expensive. In this work, we design, develop and prototype a real-time AMC using the Xilinx RFSoC platform - a new FPGA-based radio platform.

In order to achieve a real-time implementation, we must perform classifications at a high rate. Figure 1 shows the peak classification rate of three different networks running on a GPU and the RFSoC platform. The modern NVIDIA RTX 2080 Ti GPU tops out at a peak throughput of ≈ 30k classifications per second on the smaller models. In addition, a large batch-size plays a vital role in sustaining this throughput, which may not be feasible for a real-time implementation due to latency constraints. For RF applications, the data-rate can be much higher on the order of 100s of millions of samples per second, and hence, both the latency and throughput of the AMC design are critical for sensing and responding to changes in the RF channel. Using the techniques described in this paper, the RFSoC provides the computational power to perform real-time modulation classifications.

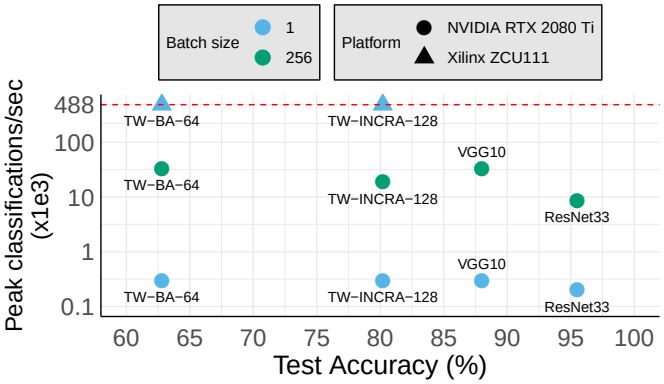



Figure 1: Peak classification throughput vs model accuracy on 24 modulation classes, measured on two modern hardware platforms. Note that increasing batch size also increases the response latency of the model.

The contributions of this work are as follows:

- A fully-pipelined ternary-weights convolution neural network implementation that is capable of producing high-speed real-time classifications on the signal modulation classification problem.
- Training CNN's by incrementally increasing the precision of the activations (INCRA) to exploit properties of the hardware implementation.
- To the best of our knowledge, the fastest, open source, FPGA-based AMC implementation.

## II. Background

### A. Automatic Modulation Classification

Automatic Modulation Classification (AMC) is a common requirement in cognitive radio networks (CRN) for both military and civilian applications. For example, AMC plays a pivotal role in dynamic spectrum management where cognitive radios are expected to detect idle frequency bands and transmit data in these bands such that the primary users of these channels are not affected [13].

AMC is a challenging task for two reasons: (1) often, there is no a priori information known about the transmission signal/channel (*e.g.* signal-to-noise ratio, carrier frequency, etc.),

especially in military contexts, and (2) latency/throughput requirements to achieve real-time processing limit the complexity of the implementation and its runtime effectiveness.

The classic approach to AMC is built on statistical/stochastic methods that require high domain expertise and frequent manual tuning to achieve reliable performance in each specific environment [11], [1], [13]. Recent efforts use deep learning techniques for the classification [9], [14], [23], which demonstrate an improvement to classification accuracy and reliability. However, developing real-time machine-learning based AMC implementation is left unexplored. Our work explores the practicalities of realizing a high throughput and low latency system implementation for doing AMC. It examines the tradeoffs in a number of networks for their hardware size and achievable throughput balanced against accuracy. This allows us to develop a system implementation on the RFSoC board that is truly real-time while maintaining reasonable accuracy.

### B. Deep Neural Networks on FPGAs

Convolutional neural networks (CNNs) are deep neural networks that comprise of convolution, pooling, and dense layers (see Sze et al. [15] for a comprehensive overview of modern CNN implementation). Our goal is to use a CNN to classify the modulation scheme of raw I/Q data, i.e., the data that is sampled directly from the analog-to-digital converter (ADC).

While CNNs are typically trained with single/double floating-point precision, inference can be done with reduced precision and/or pruning without a significant loss in accuracy [20]. Quantized CNNs are particularly useful for hardware implementations as the precision directly affects resource usage and performance [18], [21], [3], [19]. The ternarization of weights [6], [2] is a popular quantization choice as it delivers a significant reduction in the memory footprint of the CNN model while preserving the model accuracy to a large degree.

Ternary weight networks (TWNs) have parameters that are quantized during training as follows:

$$W_i^l = \begin{cases} +1, & \text{if } W_i^l > \Delta \\ 0, & \text{if } \left|W_i^l\right| \leq \Delta \\ -1, & \text{if } W_i^l < -\Delta \end{cases} \quad (1)$$

where $W_i^l$ are the parameters in each layer of the network, and $\Delta$ is a positive threshold parameter used for quantization. The activations can also be quantized to reduce the computation required.

### C. RadioML Dataset

We train and evaluate all our network designs on an open-source dataset created by O'Shea et. al. [9]. The RadioML 2018.01A dataset[1] is a collection of raw I/Q samples that have been captured over-the-air using USRP devices [9]. Each training sample is a time-series of 1024 I/Q sample pairs, accommpanied by a label that identifies its modulation class. There are a total of 24 modulation classes recorded at 26 signal-to-noise ratio (SNR) levels, ranging from $-20$ dB to $+30$ dB in increments of 2 dB. Each {*modulation class, SNR*} pair has 4096 training examples. The dataset has 2.56M labeled I/Q time-series examples. The 24 modulation classes of the signals are: OOK, 4ASK, 8ASK, BPSK, QPSK, 8PSK, 16PSK, 32PSK, 16APSK, 32APSK, 64APSK, 128APSK, 16QAM, 32QAM, 64QAM, 128QAM, 256QAM, AM-SSB-WC, AM-SSB-SC, AM-DSB-WC, AM-DSB-SC, FM, GMSK and OQPSK. We divide this dataset into training and validation sets as described in [9].

[1]https://www.deepsig.io/datasets, *Accessed: July 2019*

### D. Related Work

Deep learning for RF applications is a relatively new field, and in particular, existing work on AMC has been restricted to model design and evaluation purely in software. Mendis et. al [8] compute the spectral correlation function (*i.e.* a cyclo-stationary feature extraction step) and implement a deep-belief network for classifying five modulation classes. In [12], [7], the authors report the effectiveness of various deep neural networks on ten modulation classes, and demonstrate several strategies that help reduce training time. Zhang et. al [22] propose a heterogeneous CNN / LSTM (long short-term memory) model that delivers high classification accuracy on eleven modulation classes. While inference runtime is reported for CNN networks (on the order of a few seconds), the authors do not report runtime of the proposed heterogeneous models, which is likely to be larger due to the increased complexity in the models. O'Shea et al. [10], [9] design and evaluate two CNN models (VGG10 and ResNet33) and demonstrate competitive accuracy performance on 24 modulation classes. This paper is an expanded version of our previous work [16], with the new INCRA technique, expanded descriptions of the design, more detailed comparison with other implementations, and a full system implementation.

## III. CNN-based Automatic Modulation Classification

We use the models proposed in [9] – VGG10 and ResNet33 – as our baseline, and explore design strategies to achieve a high-speed and resource-efficient FPGA implementation. In order to achieve this, we experiment with low precision variations of the VGG10 network and explore the tradeoff of computation with accuracy. Due to the size of the ResNet33 model, implementing it on an FPGA with high throughput is difficult for two reasons: (1) the model size is too large to be spatially mapped to the FPGA fabric, which limits the achievable classification throughput significantly, and (2) the residual connections create an unbalanced design which can result in bottlenecks if large amounts of on-chip memory are not available to store intermediate activations. Hence, we elected to use the smaller and simpler VGG10 model instead for our real-time FPGA-based AMC implementation.

The VGG10 model has seven 1D convolutional layers followed by three dense layers. All of the convolutional layers have a kernel size of three and a stride of one. The

convolutions are followed by maxpool, batch normalization [4] and the ReLU activation layers. In [9], the first two dense layers use alpha dropout [5] followed by the SELU activation function. We use this training method for networks trained with floating-point precision, but swap alpha dropout layers with batch normalization layers when training low-precision networks for improved training stability.

### A. Training Method

All models are trained on the RadioML 2018.01A dataset. We set the batch size to 128, the initial learning rate to $10^{-3}$, and train for 250k steps. The learning rate was set to smoothly decay exponentially at a rate of 0.5 every 100k steps. The dataset was partitioned into a 90% – 10% split to create the train and test sets respectively, such that each {SNR, modulation class} pair had 3686 train and 410 test examples. For training, we use samples captured at $\geq$ +6dB SNR, which is typically the minimum signal strength observed in most wireless communication systems. This improves the training time, and ensures that the CNN is able to achieve the best classification accuracy for typical real-time use cases. In addition, we also use the teacher-student [2] training methodology to further improve accuracy. In this case, for all VGG10 networks, including floating-point implementations, a trained ResNet33 model was used as the teacher.

Ternary weight networks were quantized using the method described in [6], which results in a network with ternary weights and floating-point activations. The authors in [6] choose a threshold using a value of $\Delta = \nu \cdot E(|W|)$, where they recommend $\nu = 0.7$. As discussed in [17], $\nu$ can be used to control the sparsity of the weights, and hence, it has a direct impact on the implementation of the network. For networks with quantized activations we first clip the floating-point activation values between 0 and 1, and then compute

$$x' = round(x * (2^k - 1))/(2^k - 1)$$

where x is an activation, and k is the number of bits to use for the quantization. This activation quantization is done after the ReLU of each layer.

### B. Model Design

We leverage model design techniques described in [17] to implement and evaluate low precision CNNs. We explore several different model sizes, and our goal is to maximize classification accuracy as much as possible while fully utilizing the available FPGA resources. Table I details all the models and their properties explored in this paper.

The first four networks in Table I are trained and tested with floating-point weights and activations. All the networks with the prefix `TW-` are trained with ternary weights (*i.e.* {-1, 0, 1}). For all networks, we use $\nu = 0.7$ for the first layer, $\nu = 1.2$ for the remaining convolutional layers, and $\nu = 0.7$ for the two dense layers with ternary weights. Networks with `-BA-` in the model name refers to binary activations, where all activations from the layers are binary including the first two dense layers. Finally, `-INCRA-` refers to a network that was trained with incrementally increasing precision of activations, starting with 1b activations after the first layer. The incremental precision network, `TW-INRCA-128`, was trained with quantized activations for the first four layers to the following number of bits: 1, 2, 4, and 8. The next two convolutional layers do not have quantized activations and the final convolutional layer was quantized back to binary acitvations along with the first two dense layers. All networks have floating-point batch normalization variables. Neither the weights and activations of the final dense layer, nor the input data in all networks is ever trained with quantization.

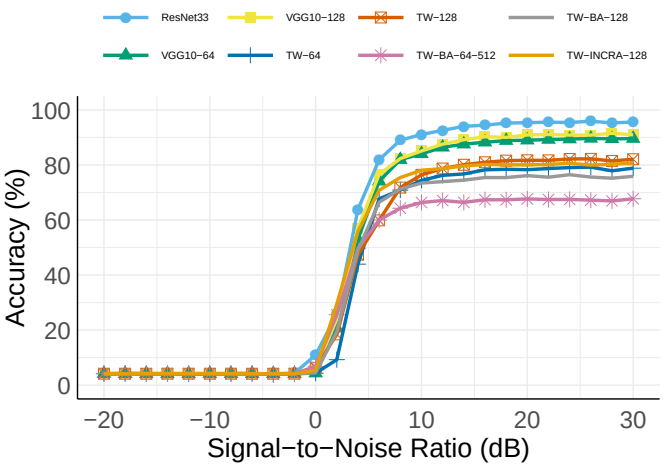



Figure 2: Accuracy (%) vs SNR (dB) with different models

### C. Model vs Accuracy tradeoff

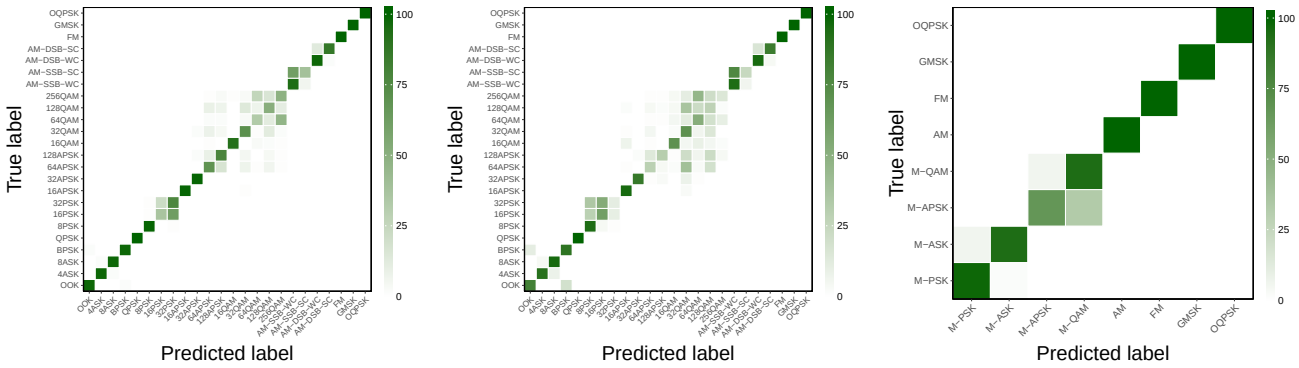
Figure 2 shows the performance of different models at different SNR levels. ResNet33 and VGG10 have the largest number of parameters and operations, and outperform the less-expressive models. Unfortunately, it is not feasible to implement a high-throughput ResNet33 model on the target RFSoC ZCU111 platform. Hence, we focus on evaluating models that have ternary weights and fixed-point activations with varying number of filters.

The confusion matrices in Figure 3 show the distribution of incorrect classifications when tested with signals at +6dB SNR. The distribution illustrates that, intuitively, the difficulty in classification lies in similar higher-order modulation schemes. For example, a valid 256-QAM signal could appear identical to a 16-QAM signal given the right encoding, especially over varying SNR, which results in significant classication errors. This is apparent when comparing Figures 3a (ResNet33) and 3b (TW-INCRA-128), where most of the misclassifications in the hardware-friendly model are due to the high-order modulation schemes. If a radio application does not require fine-grained classification between modulation classes, we can achieve much better overall classification performance. In Figure 3c, we create "super classes" of different modulation schemes, which shows that the classification accuracy across these "super classes" is almost 100% with a hardware-friendly

Table I: Properties of various models designed and explored in this paper.

| Model Name | Architecture | Precision[1] (Weights/Activations) | # parameters | # MACs | Accuracy[2] |
|---|---|---|---|---|---|
| ResNet33 [9] | {ResBlock}×6, (FC, 128)×2, (FC/Softmax, 24) | 32b/32b (FP) | 507k (2.03Mb) | 111m | 95.5 |
| VGG10 [9] | {(Conv, K3, 64), (MaxPool, S2)}×7, (FC, 128)×2, (FC/Softmax, 24) | 32b/32b (FP) | 102k (407Kb) | 12.8m | 88.0 |
| VGG10-64 | {(Conv, K3, 64), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 32b/32b (FP) | 381k (1.5Mb) | 13.3m | 89.6 |
| VGG10-128 | {(Conv, K3, 128), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 32b/32b (FP) | 636k (2.5Mb) | 51.1m | 90.9 |
| TW-64 | {(Conv, K3, 64), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/32b (FP) | 381k (95kb) | 13.3m | 78.8 |
| TW-96 | {(Conv, K3, 96), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/32b (FP) | 490k (123kb) | 29.1m | 82.4 |
| TW-128 | {(Conv, K3, 128), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/32b (FP) | 636k (159kb) | 51.1m | 82.1 |
| TW-BA-64 | {(Conv, K3, 64), (MaxPool, S2)}×7, (FC, 128)×2, (FC/Softmax, 24) | 2b/1b | 102k (25kb) | 12.8m | 62.8 |
| TW-BA-64-512 | {(Conv, K3, 64), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/1b | 381k (95kb) | 13.3m | 67.7 |
| TW-BA-128 | {(Conv, K3, 128), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/1b | 636k (159kb) | 51.1m | 75.9 |
| TW-INCRA-128 | {(Conv, K3, 128), (MaxPool, S2)}×7, (FC, 512)×2, (FC/Softmax, 24) | 2b/(variable) | 636k (159kb) | 51.1m | 80.6 |

[1]FP = 32b floating-point; [2]Best accuracy at +30dB SNR




(a) ResNet33 (b) TW-INCRA-128 (c) TW-INCRA-128 with "super-classes"

Figure 3: Confusion matrix that demonstrates the effectiveness of signal modulation classification with ResNet33 and `TW-INCRA-128` models at +6dB SNR.

model like TW-INCRA-128. The misclassification between M-ASK and M-PSK is due to OOK and BPSK modulation classes, which are very similar binary modulation schemes.

### *D. Quantization error*

The networks trained in Table I use 32b floating-point for the batch normalization variables and the final dense layer. During inference, the batch normalization variables can be multiplied into the scaling factors for the convolution to give an equation $bn(x) = ax + b$, where $a$ and $b$ are known constants that can be pre-loaded into the hardware design. Note that some networks (*e.g.* TW-64) also use floating-point activations. In order to avoid implementing floating-point blocks for portions of the computation, the calculations are done in fixed-point instead. We empirically determine that these models deliver the best accuracy when 6 fractional bits in the activation layers, and 8 fractional bits for the batch normalization variables are used. We observe minimal numerical difference at the output, typically around 2% for the class with the largest output activation value. Table II demonstrates that this proposed precision design strategy provides enough bits for stable implementations, such that any difference in

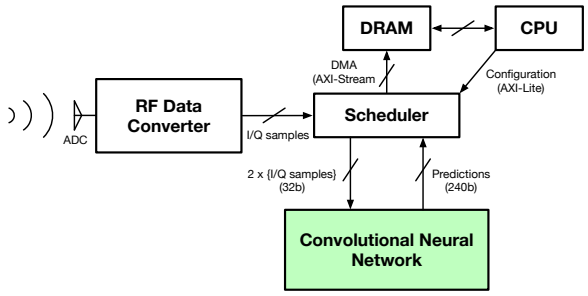



Figure 4: High-level view of the final system implementation on the ZCU111 RFSoC platform.

accuracy is essentially training noise.

Table II: Model accuracies on test set before/after quantization measured at +30dB SNR.

| Model | Accuracy (%) | Quantized Accuracy (%) |
|---|---|---|
| TW-64 | 78.8 | 78.7 |
| TW-96 | 82.4 | 81.1 |
| TW-128 | 82.1 | 81.7 |
| TW-BA-64 | 62.8 | 63.1 |
| TW-BA-64-512 | 67.7 | 67.4 |
| TW-BA-128 | 75.9 | 75.9 |
| TW-INCRA-128 | 80.6 | 80.2 |

## IV. Implementation on ZCU111

Figure 4 shows a high-level block diagram of our final implementation on the RFSoC ZCU111 platform, which we detail in this section.

### A. Scheduler

The scheduler orchestrates any data movement between the ADC, the CNN-based AMC core, and the Zynq Ultrascale host CPU. Data is communicated using the lightweight AXI4 stream protocol, where I/Q samples and prediction outcomes are sent as packets. We design the scheduler as a configurable Xilinx IP core, allowing a user to easily reconfigure data-widths of signals if needed.

### B. Auto-generator

In the course of this work, a Python3 package was developed, which has since been published as an open-source package in the pip3 packages repository[2]. It is a Python implementation of the work in [17] with additional features that improve the hardware-design productivity significantly.

Before generating the Verilog, the package also performs common subexpression elimination (CSE) on the input matrix in order to encourage result-sharing and to alleviate computational redundancy in the network. In our experiments, we observed that this CSE step can reduce the computational requirements by as much as 55% of the original network, which is crucial for deriving a highly pipelined architecture proposed in this paper. The CSE outputs an irregular adder/subtractor tree that computes a single output pixel of the convolution. The CSE is independent of the activation precision or the throughput. The output tree from the CSE, along with the specified precision and method of adding the inputs, is then used to generate the RTL implementation of the computation.

Once the trained weights from a network are conditioned into a CSV file, generating the Verilog to compute the convolutions is less than 40 lines of Python code. The tree can also be fed into a C generator for quick verification of the test set results, which offered a substantial runtime boost to the evaluation phase in our experiments.

### C. Implementation of Batch Normalization

During inference, the batch normalization layer computes $Y = ax + b$, which requires a fixed-point multiplication for each output channel of the convolutional layer and is frequently mapped to a DSP block on the FPGA. We merge the batch normalization layer with the ReLU function, where we compute $relu(x) = x \times (x > 0)$. For quantized activations with k bits, the batch normalization and ReLU are implemented as:

$$Y_i = \begin{cases} 1, & \text{if } a_i x_i + b_i > 1 \\ \frac{round((a_i x_i + b_i)(2^k - 1))}{2^k - 1}, & \text{if } 0 \leq a_i x_i + b_i \leq 1 \\ 0, & \text{if } a_i x_i + b_i < 0 \end{cases} \quad (2)$$

On the FPGA, to compute the results with integers requires multiplying with $2^k - 1$ where necessary. We incorporate the factor of $2^k - 1$ into the batch normalization variables, $a$ and $b$, prior to transforming them to fixed-point numbers.

### D. Implementation of Dense layers

The dense layers are implemented by storing the ternary weights in read-only BRAMs on the FPGA. The weights are then streamed in over multiple cycles, while the soft-logic (LUTs/FFs) manages the data movement and issues the necessary multiply-accummulate operations. This is then followed by a batch normalization and ReLU as described in the previous section.

### E. Throughput matching

As discussed in [17], the throughput of a pipelined CNN implementation is affected drastically by each maxpool layer. In a VGG10 model with 1D convolutions, the throughput drops by a factor of two after each maxpool layer, since each subsequent layer expects a smaller signal length at the input due to the maxpool operation. This drop in throughput would result in idle cycles for fully pipelined designs, where the hardware is simply waiting for new data to arrive before it is able to compute the next outputs. The number of idle cycles is exacerbated deeper down the network, and without throughput matching, the last convolution layer in VGG10 will

[2]https://github.com/da-steve101/radio_modulation

be sitting idle for ≈97% of the time, which is a significant under utilization of the available hardware resources.

We can exploit this property of the network by doing careful throughput matching. Since we know the architecture of the CNN model from the beginning, we also have apriori information available on the throughputs produced by each layer in the network. We can then allocate additional precision to the activations between layers in such a way that the number of idle cycles is minimized, or completely eliminated. For example, if a {convolution + maxpool} layer produces 2b activations, instead of using 2b adders in the subsequent layer, we can use bit-serial adders that compute each new output over 2 cycles. We can keep increasing precision in this fashion after each maxpool layer, which gives us two positive outcomes: (1) the increased precision improves the accuracy of the network, giving us higher-quality classifications at the same throughput, and (2) we can persist with the bit-serial arithmetic units in our implementation, which lowers the hardware utilization cost to the equivalent of binary activation networks. This technique essentially delivers an accuracy boost without any significant hardware overheads on the RFSoC platform, as observed in Table III and Table II when comparing TW-BA-128 and TW-INCRA-128 (≈5% accuracy boost with <1% hardware overheads). Incrementally increasing precision in this way is a core contribution of this paper.

We also do throughput matching in the dense layers. However, instead of increasing the precision, we unroll the datapath in the dense layers such that its throughput now matches the arriving data rate at its input. On the RFSoC platform, we found that we were able to increase the size of the dense layers from 128 in the baseline VGG10 model to 512 in our proposed models without suffering from over-utilization. When mapped naïvely, the last two dense layers in the baseline implementation sat idle for 75% of the time, which is reduced to zero in the proposed models. In addition, the final resource utilization is overall more balanced, as more DSPs and BRAMs are utilized to compute activations and store weights of dense layers respectively.

### *F. Functional implementation on the ZCU111*

While we train and report accuracies for our proposed models on the RadioML dataset for benchmarking purposes, a fully-functional implementation on the ZCU111 would require an end-to-end design where we train on data captured from the on-chip ADCs. In order to work towards this goal, we also create a signal generator IP core that is capable of sending modulated signals over the ZCU111's DACs. On the receiver end, we implement an I/Q sample collector IP block that designed to help do the data capture. Both IP cores are run-time configurable using the a lightweight AXI4-Lite interface (*e.g.* modulation class to transmit, number of consecutive I/Q samples to collect, etc). As of now, the modulator IP core is capable of transmitting four different modulation classes: BPSK, QPSK, 8PSK, and QAM16.

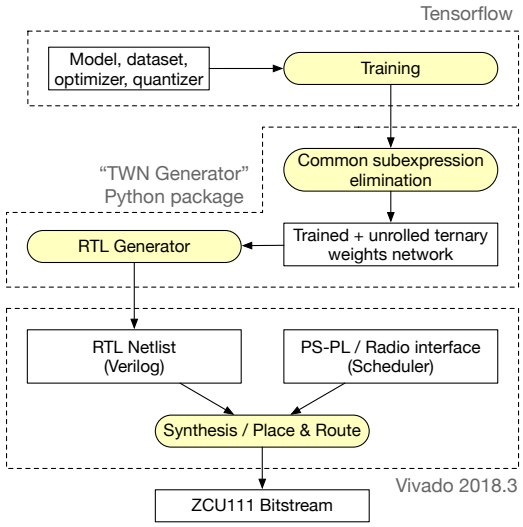



Figure 5: Completely automated toolflow for implementing a high-speed unrolled convolutional neural network for doing automatic modulation classification on the ZCU111 platform.

Table III: Out-of-context resource utilization. Target device: xczu28dr-ffvg1517-2-e at 250MHz. [1] Failed to Route

| Model | CLBs | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|---|
| TW-64 | 28k (53.5%) | 124k (29.1%) | 217k (25.5%) | 524 (48.5%) | 1496 (35%) |
| TW-96 | 47k (89.3%) | 232k (54.7%) | 369k (43.4%) | 524 (48.5%) | 1207 (28.3%) |
| TW-128[1] | 51k (96.7%) | 320k (75.3%) | 506k (59.5%) | 524 (48.5%) | 1431 (33.5%) |
| TW-BA-64 | 11k (21.2%) | 58k (13.7%) | 92k (10.8%) | 76 (7.0%) | 704 (16.5%) |
| TW-BA-64-512 | 15k (27.6%) | 80k (18.8%) | 108k (12.7%) | 521 (48.2%) | 1471 (34.4%) |
| TW-BA-128 | 43k (80.7%) | 234k (55.1%) | 333k (39.2%) | 523 (48.4%) | 1408 (33.0%) |
| TW-INCRA-128 | 42k (80.2%) | 211k (49.6%) | 324k (38.1%) | 512.2 (48.3%) | 1407 (32.9%) |

## V. Methodology

We use Vivado 2018.3 to synthesize designs, the AXI4 communication framework to interface with the design through AXI4-Lite and AXI4-stream protocols, and the PYNQ framework to manage, visualize, and verify functional correctness on Jupyter notebooks. For training, we use the Tensorflow framework to train and test various models, using the Ternary Weight Networks [6] approach for quantizing the weights.

In order to support our claim for real-time modulation classification, we choose an I/Q sample rate of 500MHz and design the CNN to accept 2×I/Q samples each cycle with a 250MHz clock.

## VI. Results

### A. Resource Utilization

Table III shows the resource utilization for the various models compared in this paper. The DSP usage is relatively constant as the convolutions do not use any DSPs. TW-BA-64 shows significantly lower DSP usage as the dense layers have 128 outputs instead of 512 like in all the other models. For binary-activation networks, the DSPs are utilized by the batch normalization layer, which is fused at the output of the dense layers. For both dense layers with 512 outputs, this is 1024 multiplications, whereas for 128 outputs only 256 multiplications need to be performed. It should be noted that this is typically multiplying a 16 bit number with a 2 bit weight so mapping it to a DSP is optional for Vivado as this could be computed with CLBs. The largest network, TW-128, fails to complete routing. All other designs met timing constraints with a 250 MHz clock.

The most interesting result in Table III is comparing TW-BA-128 and TW-INCRA-128. The model TW-BA-128 has binary activations throughout the network. This leads to lower hardware usage than TW-128 but comes at the cost of a 5.8% accuracy drop to 75.9% as seen in Table II. The model TW-INCRA-128 differs from TW-BA-128 as it has activations with increasing precision after each convolution layer (increasing from 1b to 16b, in powers of 2, and capped at 16b). This restores most of the accuracy, and is only 1.5% less accurate on the test set than TW-128 (80.2%). However, looking at the hardware utilization, both TW-BA-128 and TW-INCRA-128 have essentially the same resource usage. This is because of the careful throughput matching carried out during implementation, as described in Section IV-E earlier. By designing the precision to increase with decreasing throughput requirements, the accuracy is restored almost for free by using hardware that would have otherwise been idle.

### B. System Performance

To demonstrate the effectiveness of our method in practice, we generate a simple dataset with four modulations on the RFSoC board. A DAC and ADC on the RFSoC board were used in loopback fashion to generate BPSK, 8PSK, QAM16 and QPSK signals. A dataset was generated with 500k signals for each modulation type, each 1024 I/Q samples in length as recommended by O'Shea et. al. [9]. No noise was added to the signals. The dataset was then randomly partitioned into 90% train and 10% test resulting in 180K train examples and 20k test examples. As the dataset was small and simple, we chose the baseline VGG10 network with ternary weights and fixed point activations. The network achieved zero test set error rate in less than an epoch. This model was then put onto the board with the entire system as: Modulator $\rightarrow$ DAC $\rightarrow$ Coaxial Wiring $\rightarrow$ ADC $\rightarrow$ CNN $\rightarrow$ DMA $\rightarrow$ CPU.

Our design accepts 2$\times$I/Q samples from the ADC in each clock cycle and computes the prediction until 1024 I/Q samples have been provided before immediately starting the next one. A sample rate of 500MHz generates 488K signals of length 1024, each capturing 2.048 $\mu s$ of data. Our design is fully pipelined and matches this throughput. According to Xilinx documentation, the ADC is zero latency. With minor delays for some management logic, the entire network computes a classification in less than 2000 clock cycles from the first sample input or, with a 250MHz clock, $8\mu s$. At a throughput of 488K classifications/second and a latency of $8\mu s$, we achieve real time modulation classification with high accuracy. There is additional latency to the CPU through the DMA, which is larger than the latency for the classification but still on the order of $\mu$s. With the 12.8 MMACs required to compute a single classification on the GPU, our network computes the equivalent of 6.28 TMACs on the FPGA.

Table IV shows the utilization of different components of the system design. This shows there are significant portions of the FPGA unused in this design. Only 121k (30%) of the available LUTs are used by the entire design. Most of the resources are used by CNN model implementation (TW-VGG. TW-VGG-C$\mathrm{x}$ show the resources needed to compute the convolutional adder trees generated by our Python package where $\mathrm{x}$ refers to the layer number. Layer 2 using 16 bit adders has the highest resource utilization. Layer 3 uses 8 bit adders, and only requires half the resources of layer 2 but has the same number of output channels. Subsequent layers use lower bitwidth adders, hence use fewer resources each time, until layer 6 and 7 which both use bit-serial adders. The first layer is the exception as it has far fewer inputs – only $2 \times 3 = 6$ – compared to the $64 \times 3 = 192$ for layer 2-7. TW-VGG-Dx show the usage needed by the dense layers. D1 and D2 use ternary weights and do not need DSPs to perform the multiplication, whereas D3 uses 16b fixed-point weights to compute, which is mapped using DSPs. The resources used by the batch normalization layers are all the same after each of the 7 convolutions and the first two dense layers. They compute a 16b fixed-point multiplication, and hence require 1 DSP block for each channel.

## VII. Conclusion

In this paper we demonstrate real-time automatic modulation classification with CNNs on an FPGA. We explore the effect of hardware design parameters on model accuracy and on the performance/resource efficiency. We demonstrate the benefits of incrementally increasing the precision to match the throughput required. This improves accuracy with barely any change in hardware utilization. We also create a freely available Python package for easily generating convolutional neural networks in Verilog and C. This paper also presents the first real-time implementation, to the best of our knowledge, of a high-speed machine learning model on a radio frequency application that takes advantage of the Xilinx ZCU111 SoC achieving a throughput of 488K classifications/s and a classification latency of $8\mu$s. Finally, we also open-source IP cores for transmitting and receiving modulated radio signals in order to promote FPGA-based research and development into radio applications on the Xilinx ZCU111 platform.

Table IV: Resource utilizations for ZCU111 design

| Element | LUTs | FFs | BRAMs | DSPs |
|---|---|---|---|---|
| Top | 121k | 198k | 162 | 710 |
| TW-VGG | 109k | 194k | 152 | 708 |
| TW-VGG-C1 | 3k | 4k | 0 | 0 |
| TWVGG-C2 | 31k | 45k | 0 | 0 |
| TW-VGG-C3 | 17k | 26k | 0 | 0 |
| TW-VGG-C4 | 7k | 14k | 0 | 0 |
| TW-VGG-C5 | 5k | 8k | 0 | 0 |
| TW-VGG-C6 | 4k | 5k | 0 | 0 |
| TW-VGG-C7 | 4k | 5k | 0 | 0 |
| TW-VGG-D1 | 2k | 4k | 0 | 0 |
| TW-VGG-D2 | 2k | 4k | 0 | 0 |
| TW-VGG-D3 | 0k | 0k | 0 | 4 |
| BN-CLYR $\times 7$ | 0k | 2k | 0 | 64 |
| BN-DLYR $\times 2$ | 0k | 4k | 0 | 128 |
| AXI-DMA | 7k | 5k | 0 | 0 |
| MODULATION-GEN | 2k | 2k | 7 | 2 |
| RF-CORE | 3k | 2k | 0 | 0 |

## ACKNOWLEDGMENT

The authors gratefully acknowledge support from the University of Sydney - University of California San Diego Partnership Collaboration Awards, "Real-time machine learning for advanced wireless communications" project.

## REFERENCES

[1] Ameen Abdelmutalab, Khaled Assaleh, and Mohamed El-Tarhuni. Automatic modulation classification based on high order cumulants and hierarchical polynomial classifiers. *Phys. Commun.*, 21(C):10–18, December 2016.

[2] Hande Alemdar, Vincent Leroy, Adrien Prost-Boucle, and Frédéric Pétrot. Ternary neural networks for resource-efficient ai applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2547–2554. IEEE, 2017.

[3] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. ReBNet: Residual binarized neural network. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 57–64. IEEE, 2018.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[5] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[6] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

[7] Xiaoyu Liu, Diyu Yang, and Aly El Gamal. Deep neural network architectures for modulation classification. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 915–919. IEEE, 2017.

[8] Gihan J Mendis, Jin Wei, and Arjuna Madanayake. Deep learning-based automated modulation classification for cognitive radio. In *2016 IEEE International Conference on Communication Systems (ICCS)*, pages 1–6. IEEE, 2016.

[9] T. J. O'Shea, T. Roy, and T. C. Clancy. Over-the-air deep learning based radio signal classification. *IEEE Journal of Selected Topics in Signal Processing*, 12(1):168–179, Feb 2018.

[10] Timothy J O'Shea, Johnathan Corgan, and T Charles Clancy. Convolutional radio modulation recognition networks. In *International conference on engineering applications of neural networks*, pages 213–226. Springer, 2016.

[11] Prokopios Panagiotou, Achilleas Anastasopoulos, and A Polydoros. Likelihood ratio tests for modulation classification. In *MILCOM 2000 Proceedings. 21st Century Military Communications. Architectures and Technologies for Information Superiority (Cat. No. 00CH37155)*, volume 2, pages 670–674. IEEE, 2000.

[12] Sharan Ramjee, Shengtai Ju, Diyu Yang, Xiaoyu Liu, Aly El Gamal, and Yonina C Eldar. Fast deep learning for automatic modulation classification. *arXiv preprint arXiv:1901.05850*, 2019.

[13] B. Ramkumar. Automatic modulation classification for cognitive radios using cyclic feature detection. *IEEE Circuits and Systems Magazine*, 9(2):27–45, Second 2009.

[14] S. Riyaz, K. Sankhe, S. Ioannidis, and K. Chowdhury. Deep learning convolutional neural networks for radio identification. *IEEE Communications Magazine*, 56(9):146–152, Sep. 2018.

[15] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[16] Stephen Tridgell, David Boland, Philip H.W. Leong, and Siddhartha. Real-time automatic modulation classification. In *Proc. International Conference on Field Programmable Technology (FPT)*, pages 299–302, 2019.

[17] Stephen Tridgell, Martin Kumm, Martin Hardieck, David Boland, Duncan Moss, Peter Zipf, and Philip H. W. Leong. Unrolling ternary neural networks. *ACM Trans. Reconfigurable Technol. Syst.*, 12(4):22:1–22:23, October 2019.

[18] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[19] Stylianos I Venieris and Christos-Savvas Bouganis. fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47. IEEE, 2016.

[20] Erwei Wang, James J Davis, Ruizhe Zhao, Ho-Cheung Ng, Xinyu Niu, Wayne Luk, Peter YK Cheung, and George A Constantinides. Deep neural network approximation for custom hardware: Where we've been, where we're going. *ACM Computing Surveys (CSUR)*, 52(2):1–39, 2019.

[21] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, et al. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded FPGAs. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 23–32. ACM, 2019.

[22] Duona Zhang, Wenrui Ding, Baochang Zhang, Chunyu Xie, Hongguang Li, Chunhui Liu, and Jungong Han. Automatic modulation classification based on deep learning for unmanned aerial vehicles. *Sensors*, 18(3):924, 2018.

[23] Siyang Zhou, Zhendong Yin, Zhilu Wu, Yunfei Chen, Nan Zhao, and Zhutian Yang. A robust modulation classification method using convolutional neural networks. *EURASIP Journal on Advances in Signal Processing*, 2019(1):21, Mar 2019.