FPGA Accelerated Deep Learning Radio Modulation Classification Using MATLAB System Objects & PYNQ

Andrew Macellan, Lewis McLaughlin, Louise Crockett, & Robert W. Stewart
Department of Electrical Engineering, University of Strathclyde, Glasgow, Scotland
*Email: a.maclellan@strath.ac.uk, lewis.mclaughlin@strath.ac.uk

Background and Motivation

Deep learning (DL) and Artificial Intelligence (AI) have proven to be exciting and powerful machine learning-based techniques that have solved many real world challenges. They have made their mark in the image and video processing and natural language processing fields and now seek to make an impact on radio communications. With the increasing demand of high quality wireless data processing for spectrum sensing; cognitive radio; and accurate channel estimation, DL techniques could be used as the new state of the art answers to these problems.

Traditionally, radio communications applications are deployed on SoC and FPGA devices because of their highly parallel architecture. Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) can take advantage of FPGA architecture and accelerate their functionality. This can result in a higher compute-to-power ratio compared to current GPU deployments. In addition, FPGAs offer more flexibility with regard to data types, making them better suited to heavily quantised networks than GPUs.

Previous research has shown that quantised networks even down to 1-bit weights and activations do not sacrifice significantly in classification accuracy as shown from the FINN framework by Umuroglu et al. [1]; and Ternary Neural Network by Alemdar et al. [2]; however, these frameworks targeted image and video processing applications.

Our paper proposes an architecture for deploying an Automatic Modulation Classification (AMC) CNN onto an FPGA with 2-bit quantised weights and activations.

Training Quantiﬁed Neural Network

The final aim for the architecture is to construct a hardware efﬁcient CNN that performs modulation classiﬁcation in real time. For our prototype architecture we trained a neural network on two modulation schemes - QPSK and QAM-16. The reason for this was to conﬁrm our algorithm performed well prior to increasing the range of classiﬁable modulation schemes.

The CNN is built up of 6 layers. Table 1 indicates the conﬁguration with the number of MACs needed for each layer:

<table>
<thead>
<tr>
<th>Layer #</th>
<th>Layer Type</th>
<th>Neurons</th>
<th>Activations</th>
<th>MACs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Input</td>
<td>256</td>
<td>ReLU</td>
<td>2*256</td>
</tr>
<tr>
<td>2</td>
<td>Conv</td>
<td>64<em>1</em>3</td>
<td>ReLU</td>
<td>48384</td>
</tr>
<tr>
<td>3</td>
<td>Conv</td>
<td>16<em>2</em>3</td>
<td>ReLU</td>
<td>761856</td>
</tr>
<tr>
<td>4</td>
<td>Dense</td>
<td>128</td>
<td>ReLU</td>
<td>293052</td>
</tr>
<tr>
<td>5</td>
<td>Dense</td>
<td>2</td>
<td>Softmax</td>
<td>256</td>
</tr>
<tr>
<td>6</td>
<td>Output</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The model was trained on 34K 128 long complex modulated data at varying SNRs. Each data input included a label with the modulation scheme. The training process was performed on a quantised version of a convolutional neural network. The forward pass would quantise the weights from the backprop, where the backpropagation would use a Straight Through Estimator (STE) and limit the gradient update to 2-bits signed with one fractional bit. This limitation allows for the network to produce weights only in a limited range of values and results in greater accuracy performance compared to quantising a set of floating point weights. Figure 2 illustrates this process.

Our achieved accuracy metric with 2-bit weights in test simulation was 97.2%.

Further Work

Going forward, we intend to develop a methodology for implementing CNNs on FPGAs. We aim to utilise external memory to ease the resource requirements on the programmable logic by loading in saved weights from the processing system.

Once this methodology has been established, we intend to develop an interface to model custom networks via an abstracted user input. This is intended to reduce the time taken from realising design into a functional application. A novel PYNQ framework makes use of the Python programming language to enable easier development of designs targeting a Zynq SoC. In combining this highly productive language with an FPGA, various components can be hardware accelerated for real-time deployment (neural network, modulation/demodulation, pulse shaping etc.) and controlled easily through a Jupyter Notebook, similar to calling functions from a software library [3].

- Python is currently a prevalent environment for developing DL algorithms. In addition, a PYNQ image exists for RFSoC – a SoC created specifically for interfacing with RF signals – making PYNQ a suitable framework for consolidating deep learning and radio communications processing.
- Jupyter provides the user with an interface for communicating with the hardware design. It can be used to send data to and receive data from the PL, allowing for a convenient means in which to both control the design and visualise data produced.

The architecture illustrated in Figure 3 proposes a single chip transmit and receive system. Data can be transmitted in BPSK, QPSK, 8PSK and 16QAM. The particular scheme can be selected from Jupyter and changed dynamically. The received data enters both the CNN and the receiver IP which performs the necessary filtering and demodulation for all schemes. The output of the neural network selects which demodulator output should be passed into the PS for analysis.

The neural network is designed using MATLAB System Objects as this provides a convenient method for simulation in fixed point as well as generating the necessary HDL code for deploying on an FPGA. System Objects also easily allow for aspects to parameterised (no. layers, weights, fixed point precision) meaning, once the initial framework has been established, further CNNs can be rapidly prototyped and deployed for various communications applications.

The PYNQ framework makes use of the Python programming language to enable easier development of designs targeting a Zynq SoC. In combining this highly productive language with an FPGA, various components can be hardware accelerated for real-time deployment (neural network, modulation/demodulation, pulse shaping etc.) and controlled easily through a `Jupyter Notebook`, similar to calling functions from a software library [3].

- Python is currently a prevalent environment for developing DL algorithms. In addition, a PYNQ image exists for RFSoC – a SoC created specifically for interfacing with RF signals – making PYNQ a suitable framework for consolidating deep learning and radio communications processing.
- Jupyter provides the user with an interface for communicating with the hardware design. It can be used to send data to and receive data from the PL, allowing for a convenient means in which to both control the design and visualise data produced.

The architecture illustrated in Figure 3 proposes a single chip transmit and receive system. Data can be transmitted in BPSK, QPSK, 8PSK and 16QAM. The particular scheme can be selected from Jupyter and changed dynamically. The received data enters both the CNN and the receiver IP which performs the necessary filtering and demodulation for all schemes. The output of the neural network selects which demodulator output should be passed into the PS for analysis.

The neural network is designed using MATLAB System Objects as this provides a convenient method for simulation in fixed point as well as generating the necessary HDL code for deploying on an FPGA. System Objects also easily allow for aspects to parameterised (no. layers, weights, fixed point precision) meaning, once the initial framework has been established, further CNNs can be rapidly prototyped and deployed for various communications applications.