Implementation of MNIST Dataset using HLS4ML

TSELEPI ELENI

KAPAKOS GEORGIOS

PROJECT PRESENTATION 2024

Outline

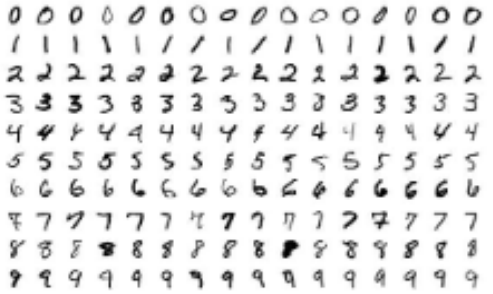
- Introduction
- Case Study
- Methodology
- Optimizations
- Experimental Results
- Conclusions

Introduction

- As new models keep growing in size and complexity, their number of operations and memory footprint pose significant challenges, especially under the strict latency and power consumption constraints typical of real-time edge systems
- The main obstacle to broader adoption of FPGAs in the Deep Learning community is the wide gap between software design and RTL implementation
- We used the hls4ml library, an open source software tool for deploying machine learning models on FPGAs to implemented a standard benchmark dataset like MNIST in order to examine the area optimizations

Case Study

- What is the MNIST dataset?
  - It is a dataset, that contains handwritten digits from 0 to 9.



- What is HLS4ML?
  - HLS4ML is an open source software library to convert Machine Learning Models to on FPGAs.
  - Taking a Neural Network as model it generates C/C++ code for the FPGA firmware.

Methodology

A. Set Dataset

B. Model

C. Training

D. Compilation

E. Synthesis

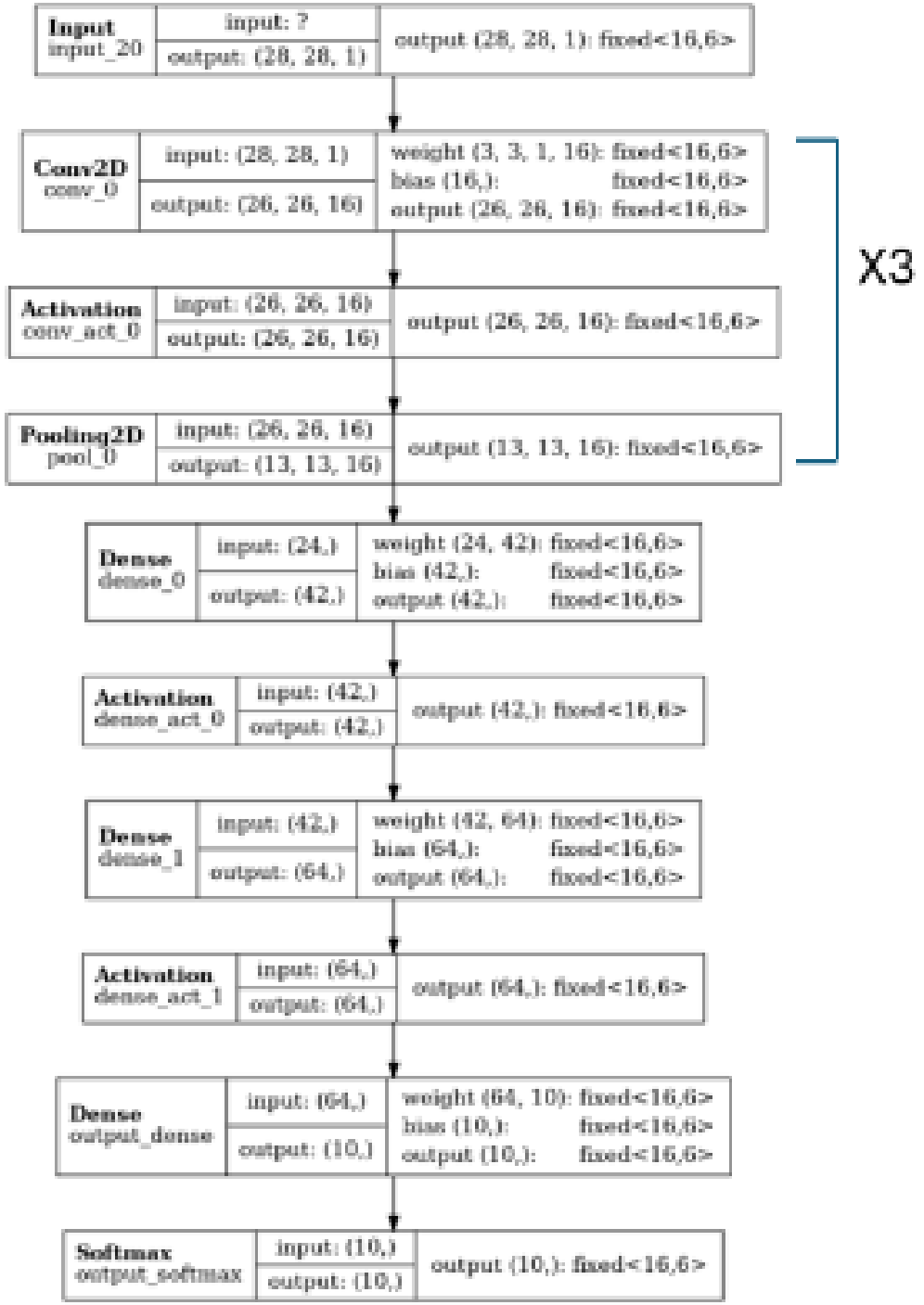
Model-Layers

- The model we created is composed by these types of layers:
  - Convolutional(Conv2D) layers: Convolutional layers apply a convolution operation to the input, passing the result to the next layer.
  - Max Pooling Layers: For downsampling spatial dimensions of an input volume.
  - Flatten Layer: To reshape the output into a 1-D vector.
  - Batch Normalization: To make the Neural Networks run faster as through adding extra layers in a deep Neural Network.
  - Dense Layers: To alter the dimensionality between fully-connected layers.
  - Activation Functions:
    - ReLU:(Between layers): Introducing the property of non-linearity into a deep neural network.
    - Softmax:(Output Layer): Handles multiclass classification problems, converting the logits into propabilities.

Our Model

The model has the following setup as seen in the shape:

The Input layer is followed by a convolutional layer(2-D), followed by an activation function(ReLU) and a Max-Pooling layer(2-D), this pattern is repeated for three times which is connected with Dense layer followed by an activation function(ReLU), repeated twice. Finally the later is connected
to the output Dense layer, classifying the result in the 10 different classes, followed by a softmax function



Compile

- **hls4ml.utils.config_from_keras_model** :This function serves as the initial step in creating the custom conversion configuration.

- **Hls4ml.converters.create_config(backend='Vivado')** : We set HLS4ML configurations like the type of input/output interface to io stream, which is necessary for CNNs.

- **hls4ml.converters.keras_to_hls()** and **hls_model.compile()** to convert the Keras model into an HLS project based on the configurations we defined previous and to compile the HLS project, transforming the model into C++ code ready for FPGA synthesis.

Synthesis

- **hls_model.build()** : Using the build method the model we created will continue with the synthesis on the XILINX VIVADO 2019.1 version.

- **Synthesizing** only the design and skipping the simulation part through our function settings.

Optimizations - Pruning

What is Pruning?

- Is a technique, that eliminates redundant components in the system, by removing the less important weights in the convolutional and dense layers of the Neural Network

How we pruned our model?

- We pruned the dense and convolutional layers gradually from 0 to 50% sparsity every 2 epochs ending by the 10th epoch using the TensorFlow Model Optimization Toolkit
- We applied pruning using the *sparsity.keras.prune _low_magnitude* command
- Specifically, it prunes the weights of the layer by setting the low-magnitude (i.e., small absolute value) weights to zero
- We used the class PolynomialDecay to define a schedule to gradually increase the sparsity of a model's weights over a specified period. We ensure that the model's performance does not degrade suddenly, but rather adapts to the increasing sparsity gradually

Optimizations - Quantization

What is Quantization?

- Is a technique, which is used to reduce the precision of the numbers that are used to represent data.

How we quantized our model?

- For better results, we created a pruned and quantized model using Qkeras
- We build the model sequentially
- With the command *quantized_bits(6,0,alpha=1)* we quantized the weights and biases to 6-bit precision from 16-bit precision.
- Then we added a quantized ReLU activation function using QActivation and a max-pooling layer to reduce the spatial dimensions.

Experimental Results - Resources

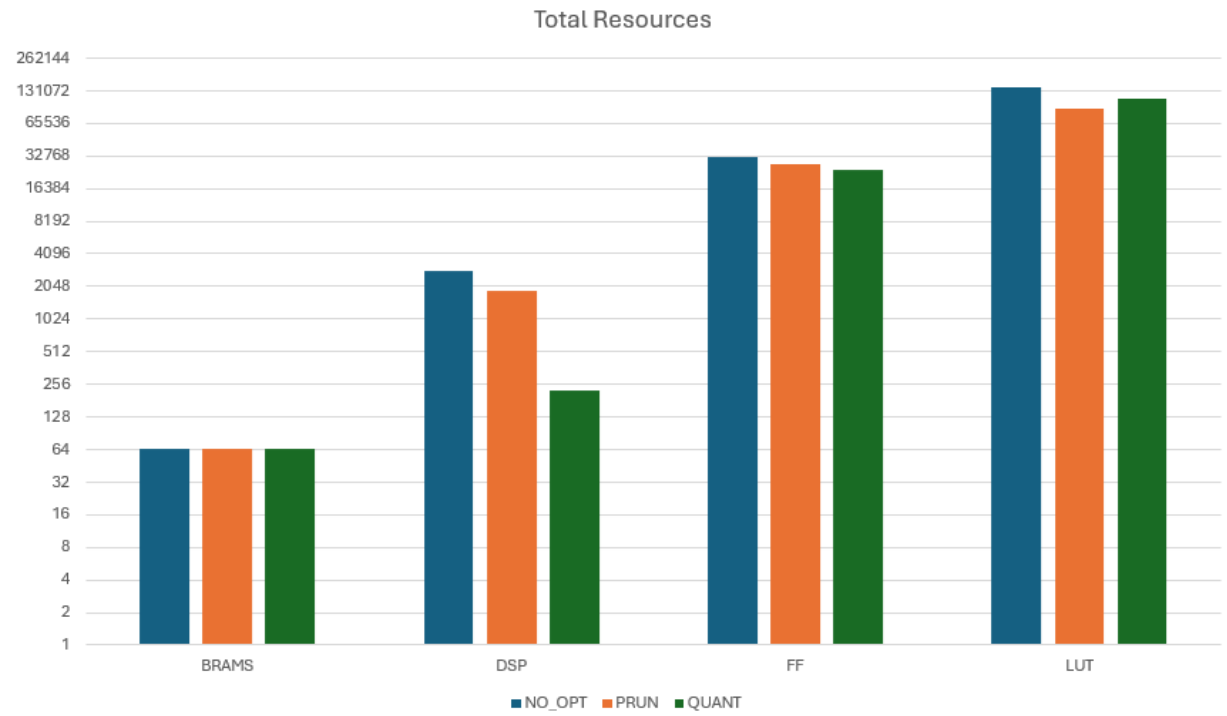
- We conducted a comparison between the unoptimized model and the two methods, pruning and quantization.
- The unoptimized model utilized a total of 64 BRAMs, 2850 DSPs, 31,765 Flip Flops, and 141,367 LUTs.
- The pruned model utilized a total of 64 BRAMs, 1840 DSPs, 27,260 Flip Flops and 88,507 LUTs.
- The quantized model utilized a total of 64 BRAMs, 222 DSPs, 24,617 Flip Flops and 110,807 LUTs

TABLE I
RESOURCES COMPARISON BETWEEN THE THREE MODELS

Model\Resources	BRAM	DSP	FF	LUT
No Optimization	64	2850	31765	141367
Pruned	64	1840	27260	**88507**
Quantized	64	**222**	**24617**	110807

Experimental Results - Resources

- The optimized models show a 35% reduction in DSP usage for the pruned model and an impressive 92% reduction for the quantized model compared to the unoptimized model.
- There is a reduction in Flip Flops by 14 % for the pruned model and 22.5% for the quantized model.
- The LUT usage is reduced by 37% for the pruned model and 21% for the quantized model



Experimental Results - Presicion

- The unoptimized model achieves a notable accuracy of 0.99
- The pruned model has a slightly lower precision at 98.79.
- The quantized model achieves an accuracy of 0.96.
- This means there is a 0.4% loss in accuracy between the unoptimized and pruned models, and a 3% loss in accuracy between the unoptimized and quantized models

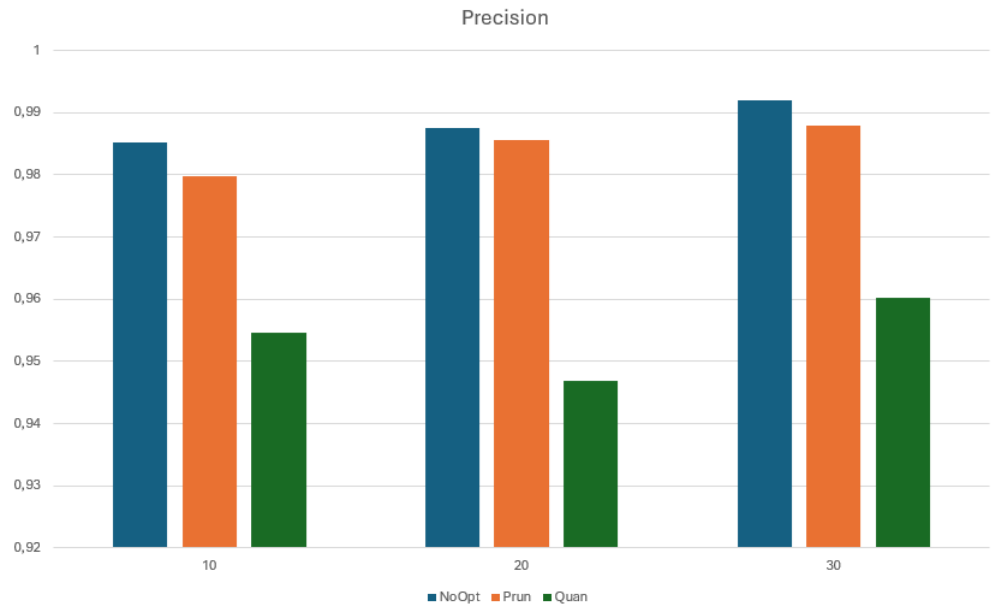


TABLE II
MODEL ACCURACY OVER THE EPOCHS

Model\Epochs	10	20	30
No Optimazation	0,9852	0,9876	0,9919
Pruned	0,9797	0,9855	0,9879
Quantized	0,9547	0,9468	0,9603

Conclusions

1. The HLS4ML model in NNs, is a useful method to convert Neural Networks into C/C++ code for firmware.
2. HLS4ML makes it easy to use optimization techniques into the hardware directly
from build in functions inside it's library.
3. Results from the model we implemented
show insignificant accuracy loss compared to the starting model's accuracy, making great optimizations in area.

Thank you for your attention !