Compressed Neural Network Implementation for Low Power Medical Imaging

Evangelos Ananiadis, eananiadis@uth.gr
Supervisor: Prof. Georgios Karakonstantis

***Abstract*—Breast cancer remains one of the leading causes of death among women worldwide. Diagnosis is possible through breast ultrasounds by medical experts, but what if this process could be automated and simplified? In this report we describe the process of porting a breast cancer diagnosis Convolutional Neural Network to an FPGA, with the aim of creating a low-power and accurate solution for breast cancer diagnosis.**

***Index Terms*—FPGA, Medical Imaging, CNN**

I. Introduction

Early detection of breast cancer is critical for its treatment. Breast ultrasound (BUS) is a widely used medical imaging technique for identifying abnormalities in breast tissue, particularly for distinguishing between malignant and benign tumors [4] [5].

Machine learning (ML) plays a crucial role in modern medical diagnostic tools, providing fast and accurate interpretation of medical images. In clinical settings ML has been applied to early cancer detection, automated fracture identification, diabetic eye disease screening, among other areas, often achieving diagnostic performance on par with or better than human experts [6]. Deep learning networks, especially Convolutional Neural Networks (CNNs), have shown strong performance in detecting breast cancer by learning to recognize intricate tumor patterns in BUS images.

Deploying ML models on Field Programmable Gate Arrays (FPGAs) offers several advantages over traditional CPU and GPU platforms. FPGAs provide low latency, high energy efficiency, optimized resource usage, and hardware reconfigurability. These benefits make FPGAs well-suited for integration into portable ultrasound systems capable of delivering real-time breast cancer diagnosis, making the diagnosis more accessible to patients. This report describes the process, challenges and benefits of porting a CNN for breast cancer detection on an FPGA.

II. Concept

A. CNNs in Breast Cancer detection

Balasubramaniam et. al [2] examined the performance of various CNN architectures that were trained on a public BUS image dataset [1]. The dataset contains 780 BUS images, categorized in three classes: normal, malignant and benign.

The best performing architecture is that of a modified LeNet, designed to better detect the abstract features of breast cancer tumors. The differences between traditional LeNet and modified LeNet are shown in Fig. 1.

TABLE I
Comparison of Proposed Model with Pre-trained Models on the BUS Dataset [2] [1]

Models	Val Acc.
AlexNet	0.7325
SegNet	0.7426
U-Net	0.7003
CE-Net	0.6982
SCAN	0.7169
Dense U-Net	0.7456
LeNet	0.7223
Modified LeNet	**0.7652**

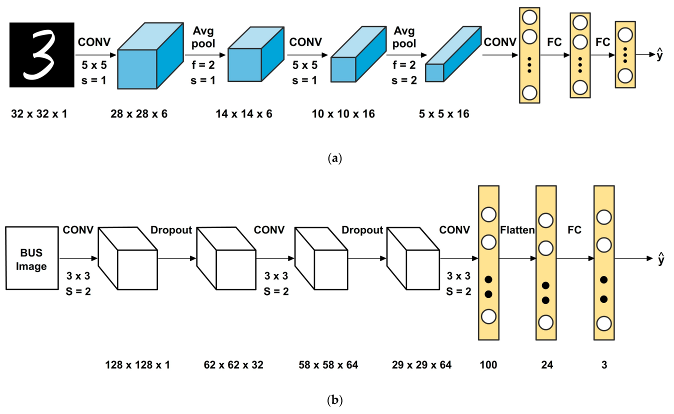


Fig. 1. a) Classical LeNet architecture. (b) Modified LeNet architecture [2]
.

Deploying a CNN on an FPGA is achieved by selecting a suitable model, compiling and optimizing the model to High Level Synthesis (HLS) language and after meeting FPGA constraints, deploying it on the target FPGA, as described in Fig. 2.

The modified LeNet described above is used as a case study, as it fits the above criteria: The initial accuracy is approximately 76%. Moreover, the Convolution, Dropout, Flatten and FC layers are supported by HLS compilation tools. The amount of weights is small, due to the few simple layers, and as such a low memory footprint can be achieved. Further optimizations are examined below.

B. HLS Compiler

hls4ml (High Level Synthesis for Machine Learning) is a highly versatile open-source High Level Synthesis compiler originally developed by CERN and collaborators. This tool

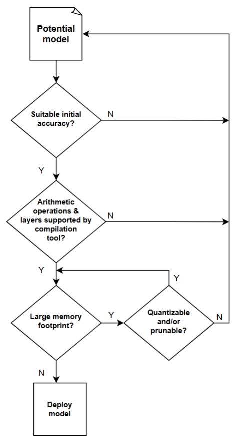

Potential model
Suitable initial accuracy?
N
Y
Arithmetic operations & layers supported by compilation tool?
N
Y
Y
Large memory footprint?
Y
Quantizable and/or prunable?
N
N
Deploy model

Fig. 2. Decision tree for choosing an architecture for FPGA inference, edited from [7]
.

essentially compiles pre-trained ML models into a High Level Synthesis language that can run on FPGAs [3]. hls4ml is designed for low-latency environments and supports simple, lightweight models. hls4ml provides front-end support for Keras, QKeras PyTorch, Brevitas, ONNX and QONNX models, and back-end support for Vivado/Vitis, VivadoAccelerator, oneAPI, Catapult, Quartus and SymbolicExpression.

C. FPGA Inference of the Modified LeNet

FPGA inference was achieved with a 70% accuracy. In order to fit an image processing CNN on an FPGA, a series of experiments must be done to determine the optimizations that need to be applied, under the constraints of the FPGA (available resources, specifically LUTs, BRAMs, FFs and DSPs) and the use-case constraints (accuracy, power, throughput). A medium scale Zynq UltraScale+ MPSoC FPGA part was selected: XCZU3EG-SBVA484-1-E. The reason for this was to have just enough resources to enable experimentation with techniques that increase accuracy, while also having to apply heavy optimizations - without them the model would not even compile. After implementation succeeded it was implemented on a ZedBoard.

Part	BRAM_18k	DSP	FF	LUT
Zynq UltraScale+ MPSoC	432	360	141120	70560
Zynq-7000 ZedBoard	140	220	106400	53200

TABLE II
FPGA RESOURCES

III. IMPLEMENTATION

A. Goals

This project examines FPGA inference of the CNN under Power, Accuracy and Latency constraints.

1) **Low-Power Usage**
The inferred model should work on low power settings. Thankfully, modern medium scale FPGAs work on less than a few Watts, our model specifically has a Total On-Chip Power of around 0.5Watts.
2) **High Diagnostic Accuracy**
The solution should retain high diagnostic accuracy. This is mainly achieved by training the model with the proper parameters and techniques and then correctly tuning FPGA parameters. The final design has an accuracy of 71%. Given the strict on-chip resource budget this is a reasonable accuracy level for FPGA inference.
3) **Low Latency/ High Throughput**
In theory, the FPGA would be connected to an ultrasound device, producing a live diagnosis. Live image processing is a highly demanding task and is generally not fit for FPGA applications. It should be noted that hls4ml is ideal for FPGA inference applications with low latency constraints. The final latency was calculated at 20us with a 200MHz clock.

B. Training the Modified LeNet

A large part of this project was training the model, in order to reach the best possible accuracy before lowering it to accommodate HLS optimizations. The original dataset [1] contained 780 breast ultrasound images. The breast ultrasounds were manually classified into normal, malign, benign. The dataset was then augmented by rotating, zooming, widening, lengthening and flipping the original images. This was possible due to the noisy and highly variable nature of the breast ultrasounds. Then a series of experiments took place to find the training parameters that were not described in Balasubramaniam et. al, or were better fit for our implementation. Batch size was set to 32 and learning rate was set to 0.01 due to the small-medium size of the dataset. The dropout of the dropout layers was set to 0.2. The kernel sizes and strides for the convolution layers were experimented with and the balance between model size and accuracy was achieved with kernel sizes 3, 2, 4 and strides 1, 2, 2 for the first, second and third convolution layers respectively. Epochs were set to 100 for the final model with callbacks implemented. With these techniques we successfully trained the model to an initial 80% accuracy, 5% higher than the accuracy of the proposed Modified LeNet.

C. HLS Conversion of the Modified LeNet

The modified LeNet was converted to HLS using hls4ml 1.1.0 and then implemented with Vivado 2023.2.

1) **Model Complexity**
Layers are the fundamental blocks of CNNs. The type, amount, and parameters of layers are directly responsible for the model size, as well as the data flow. CNNs can be compressed by removing layers or altering hyperparameters and retraining. Here we experimented with reducing the filter sizes of the 2D convolution layers. The first, second and third layer had filter sizes of 32, 64, 128 respectively, that were reduced down to 4, 8, 16, since it was determined that up until this point accuracy drops by less than 1%. Halving all the filters approximately halved all utilized blocks, LUTs, DSPs,

BRAMs and FFs. Additionally, halving the input size of the model significantly reduces its size, dropping utilization on the synthesized model by about 60% with a 5% accuracy penalty. This is the most aggressive optimization regarding the accuracy-resource utilization balance.

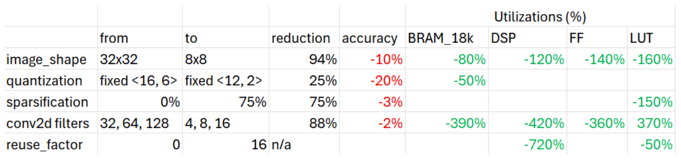
2) **Weight Size and Data Types**
By compressing weights, a lighter model is produced, reducing power usage at the cost of accuracy. Quantization reduces the numerical precision of the stored weights. In our case this means reducing BRAM utilization, since hls4ml stores all the weights on BRAMs. By default, weights are fixed 16,6, the number is stored as a 16-bit integer, but the lower 6 bits are interpreted as the fractional part. It was shown that the bitwidth of the weights significantly affects the model's accuracy. Even mild quantization (eg. fixed 12,2) drops the accuracy by as much as 20%. This optimization was not necessary in our case, since from the start the model weights easily fit in the BRAMs (less than 40% utilization).
Sparsification rounds the smallest weights to zero, effectively eliminating them. Constant sparsification was used for the training of the model, which begun after the first 5 epochs, in order to reinforce the model basis and was then implemented at every epoch afterwards. Increasing sparsification by 25% decreases LUT usage by about 50% with a minimal accuracy loss of less than 1-2%, and should be one of the first concerns when trying to infer a model.

3) **Dataflow & Resource Utilization**
Reuse factor essentially saves resources by reusing hardware blocks. In our case every time we double the reuse factor, the DSPs are halved. Model accuracy is not affected, but latency is increased. hls4ml generates fast solutions, so this is not a concern.

D. *Results*

Part	BRAM_18k (%)	DSP (%)	FF (%)	LUT (%)
Zynq UltraScale+	2.78	16.11	12.44	19.36
ZedBoard	2.78	20.28	15.4	26.91

TABLE III
FPGA UTILIZATION



					Utilizations (%)			
	from	to	reduction	accuracy	BRAM_18k	DSP	FF	LUT
image_shape	32x32	8x8	94%	-10%	-80%	-120%	-140%	-160%
quantization	fixed <16, 6>	fixed <12, 2>	25%	-20%	-50%			
sparsification	0%	75%	75%	-3%				-150%
conv2d filters	32, 64, 128	4, 8, 16	88%	-2%	-390%	-420%	-360%	370%
reuse_factor	0	16	n/a			-720%		-50%

Fig. 3. Summarized approximations of the effects of optimizations on resource utilization.

As mentioned before, the implementation on the Zynq Ultrascale+ MPSoC FGPA has a 71% accuracy and can run at almost 200MHz, with a latency of 20us, while consuming only 400mW on the Ultrascale FPGA. A modern CPU would require at least 20W for inference. The CPU holds higher accuracy, at 85%. However, it lacks in throughput, as it was calculated having a latency of 90ms, for a medium range 2023 Intel Core i5.

IV. Challenges & Future Prospects

A. *Implementation Challenges*

Some of the challenges that we faced during this project, are described in this section, for future reference. It should be noted that the following challenges were determined after extensive documentation hunting, searching for known issues in the hls4ml Github repository, and, of course, trial and error. hls4ml may silently fail for any number of reasons. Keeping in mind the following details should eliminate a large portion of bugs, or seemingly random silent compilation or synthesis fails. Firstly, one of the earliest issues one could face when using this tool is a g++ exception error when compiling or building the model. This error highly suggests that the model is too large to even compile. The solution for this should be drastic, the model size should be reduced by at least an order of magnitude, just for the model to compile. More options, include changing the io_stream argument of the build() method from the default io_parallel to io_serial. By changing this option the latency increases, but the utilizations significantly drop. io_parallel will only work with very large FPGAs, very light models or very aggressive optimizations, since when it is enabled LUT and DSP usages skyrocket. hls4ml provides support for Keras v2 since its early days, while support for PyTorch is relatively recent and even more so for Keras v3. Given the choice, building the model in Keras v2 would be the better choice. It should also be noted that a model may synthesize from the build() method, however the implementation may not run, with cryptic errors, such as "vivado returned an error", or "vivado loader was killed". This often means that Vivado tries to use too much memory and so the system the OOM-killer is called. A possible solution in systems or VMs with limited memory (less than 8GB), is to increase swap memory. Moreover, the hls4ml documentation lists the tools' dependencies. One should be careful when installing hls4ml on a preexisting environment, since an older version of hls4ml might install, instead of the current main build. For this reason, a clean conda environment is recommended. On the topic of version dependencies it is not entirely clear which versions of the Vitis suite work with the hls4ml tool. Fastmachinelearning recommends newer Vitis versions, however some of the legacy versions could work on some specific cases. Vitis suite 2022.2 should be avoided, since it usually does not compile, a fact which is not mentioned on the documentation pages. Finally, High Granularity Quantization (HGQ) is a framework for quantization developed by Fastmachinelearning, specifically for use with hls4ml. However, it is documented austerely, making it hard to understand and implement.

B. *Future Prospects*

Future work will focus on raising diagnostic accuracy beyond the current 71% by employing larger FPGAs that can host larger neural-networks. The FPGA pipeline can also be adapted to other medical-imaging tasks with only minor and straightforward changes. Advanced quantization techniques could be explored using frameworks such as HGQ, or QKeras. To streamline inference, a CNN-to-FPGA wrapper could be developed to auto-tune device parameters for any model, balancing latency, power, and resource budgets without extensive manual tweaking. Finally, hardware-aware neural-architecture search is a promising path to designing networks that are natively optimized for FPGA constraints.

References

[1] Walid Al-Dhabyani, Mohammed Gomaa, Hussien Khaled, and Aly Fahmy. Dataset of breast ultrasound images. *Data in brief*, 28:104863, 2020.

[2] Sathiyabhama Balasubramaniam, Yuvarajan Velmurugan, Dhayanithi Jaganathan, and Seshathiri Dhanasekaran. A modified lenet cnn for breast cancer diagnosis in ultrasound images. *Diagnostics*, 13(17):2746, 2023.

[3] FastML Team. fastmachinelearning/hls4ml, 2025.

[4] M Garcia, A Jemal, EM Ward, MM Center, Y Hao, RL Siegel, and MJ Thun. Global cancer facts & figures american cancer society. *Atlanta, GA, USA*, 2007.

[5] Paula B Gordon and S Larry Goldenberg. Malignant breast masses detected only by ultrasound. a retrospective review. *Cancer*, 76(4):626–630, 1995.

[6] Mengfang Li, Yuanyuan Jiang, Yanzhou Zhang, and Haisheng Zhu. Medical image analysis using deep learning algorithms. *Frontiers in public health*, 11:1273253, 2023.

[7] Jonas Sundseth. Acceleration of deep learning algorithms for cardiac ultrasound processing by use of xilinx fpga. Master's thesis, NTNU, 2021.