Tecnológico de Costa Rica

Computer Engineering Department

Licentiate Degree Program in Computer Engineering

Approximate kernel generation for convolutional neural networks using OpenCL

Final report submitted in partial fulfillment of the requirements for the degree of Licentiate in Computer Engineering

Esteban Calvo Vargas

Cartago, March 7th, 2018

a mis queridos padres

Agradecimientos

Gratitude pending

Esteban Calvo Vargas

San Jose, September 16, 2019

Resumen

Resumen en espera

Palabras clave: palabras, clave, ...

Abstract

The same as before, but in English.

Keywords: word 1, word 2,

Contents

List of Figures

Table index

List of abbreviations

Abbreviations

CES	Chair for Embedded Systems
CNN	Convolutional Neural Network
DNN	Deep neural Network
FPGA	Field-Programmable Gate Array
KIT	Karlsruher Institut für Technologie

Chapter 1

Introduction

Computers were invented to accelerate manual processes. In the beginning, computers offered speeds far superior than what a human could manage on specific activities, but their use was limited to relatively low data processing.

In the current era of increasingly advanced semiconductor technologies, a need for using computers with applications for high-level data processing has risen along with higher amounts of data to process. This has generated a race for maintaining high performance while keeping equal or even reduced energy, time and storage consumption. The main solution, for some years, has been to increase the computing power of the computers via mechanisms such as, for example, increasing the number of transistors per area. This solution, however, has brought with it a lot of considerations and problems, specially regarding energy consumption.

A new approach has surfaced as one of the solutions to the energy consumption problem is, approximate computing. This computing paradigm was born on the assumption that there are cases on which an exact result, with high precision, is not needed. A lot of data comes from inexact sources (sensors, readings) or do not require a precise processing algorithm (machine learning, user recommendation programs, statistics). This type of applications is known as error-tolerant. Approximate computing looks to use these types of data to create algorithms, languages, compilers, circuits and computer architectures that have the common objective of lowering the energy consumption and increasing the performance at the cost of having an approximate result.

Machine learning is a current area of research that can take advantage of the benefits of approximate computing. Its main property is decision making based on processing big amounts of data. This data can be written, visual (images, video) or audio information, as well as taking the feedback into account to improve the learning process. Approximate computing can take advantage of this fact and use it to reduce the computational effort required and, in this manner, improve significantly the investigation area of machine learning.

This work looks to explore methods and techniques for approximate hardware definition

on FPGA, such that it could be used on machine learning applications, specifically by generating approximate hardware kernels for CNNs through the use of OpenCL. The current project will deliver useful tools for any other developer that requires to accelerate their machine learning algorithms through the use of FPGAs.

1.1 Project background

1.1.1 Organization

The project is developed in the KIT, a university focused on the development of technology and science. KIT was created on 2009 after the convergence of the University of Karlsruhe, funded on 1825, and the Investigation Center of Karlsruhe. It is located in Karlsruhe, in the Baden-Württemberg state, to the southwest of Germany. Figure 1.1 shows a map with the location of Baden-Württemberg inside the european country and the location of Karlsruhe inside the aforementioned state.

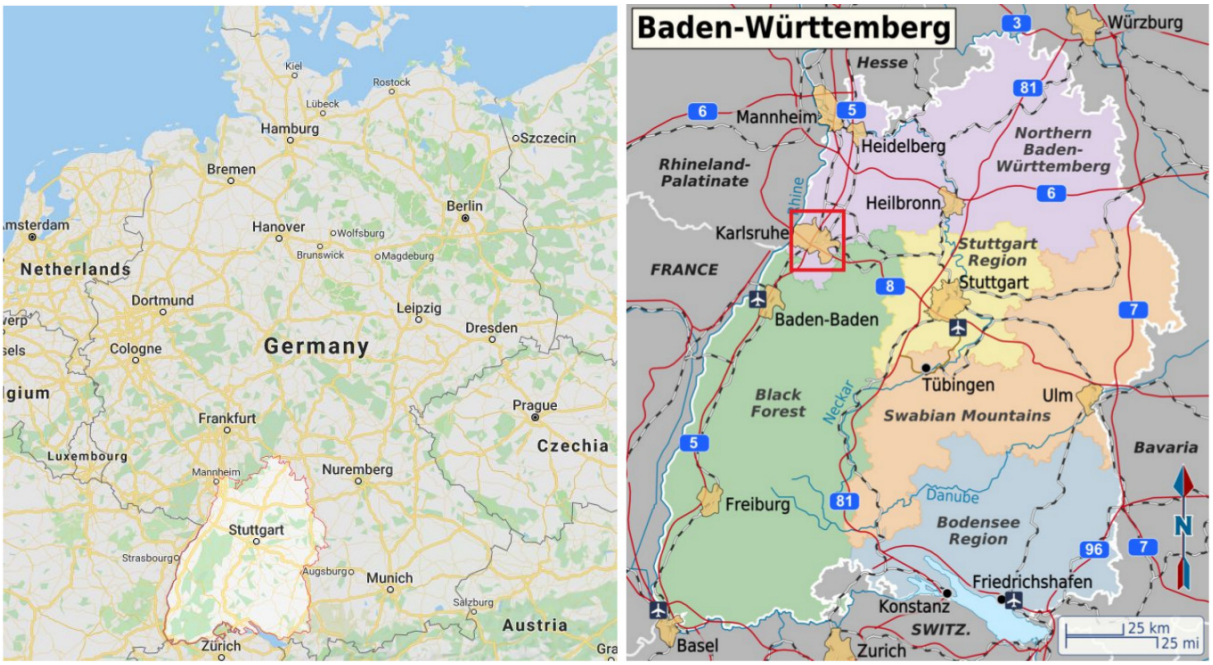

Figure 1.1: Map of Germany with Baden-Wüttemberg marked (left) [1] and location of Karlsruhe within the state (right) [22]

Nowadays, KIT is one of the most prestigious technical universities in Germany, specializing on engineering and science. Figure 1.2 shows the current logo of the university. KIT is conformed by a scientific organization separated by divisions:

- Division I: Biology, Chemistry and Process Engineering.
- Division II: Informatics, Economy and Society.
- Division III: Mechanical and Electrical Engineering.

Figure 1.2: Logo of the Karlsruher Institut für Technologie [20]

- Division IV: Natural and Built Environment.
- Division V: Physics and Mathematics.

These divisions are constituted by departments destined to investigational work, innovation and teaching. Each department has institutes responsible for university education. Aside from the institutes, there are KIT centers, on which topics related to investigation and innovation go beyond the divisions, supporting an interdisciplinary cooperation [21].

The Department of Informatics is one of the first to be established on Germany. This department is formed by different institutes focused on teaching and investigation of topics associated with informatics. As part of the Department of Information the CES comes as a center of investigation on aspects related with the design of embedded systems, from the reliability of electrical circuits to the management of electrical power on systems with multiple and many cores.

1.1.2 Knowledge area

The project is developed within the technical area of approximate computing, which is part of the areas of interest of computer engineering. Approximate computing looks to relax the numerical equivalence between the specification and implementation of such applications promising significant energy-efficiency improvements and it has gained significant traction over the past few years [49]. Within approximate computing, the project focuses on the utilization of Intel's® FPGA SDK for OpenCL™to develop approximate hardware kernels and their application on CNNs. The goal is to highlight the effect of these kernels on the improvement in performance and reduction on energy consumption in comparison to exact kernels.

This kernels must be designed using high-level synthesis tools. OpenCL is a tool that allows to define procedures on heterogeneous platforms through the use of a high-level programming language. The knowledge on hardware changes via the use of a software description is necessary to develop the project.

1.1.3 Similar works

In the area of neural network implementation on FPGA using OpenCL, notable previous works are:

- Suda et al. introduce a complete OpenCL-based accelerator design for CNNs based on matrix multiplication. They also explain algorithms to apply convolution by rearranging the components of the input image. This work is an exact CNN that could be used as a base for the current project. [41]
- Wang et al. propose a CNN implementation on FPGA specifically taking advantage of the ability to allow communication between layers through data channels [46].
- Zhang and Li present a model to analyze the resource usage on an FPGA, as well as implementing their own accelerator based on OpenCL [52].

Regarding the use of approximate computing for CNNs, we can find the following related works:

- Moons et al. show that using approximations on popular CNNs can increase the energy efficiency of the accelerators and show some of the techniques that can be applied to other CNNs [30].
- Kamel et al. compile a lot of techniques that can be used on CNNs and some other information regarding the application of CNNs on FPGAs [4].

Also, there is the work done by D. Spies [reference missing as the thesis has not been published yet] on which some CNN frameworks were implemented for low-power FPGAs, reducing the need for high-end FPGAs and allowing for further research on the area.

1.2 Problem statement

1.2.1 Problem context

In the last years, discussions on the physical (and economical) limits of the very-large-scale integration of transistors have surfaced [25][26], where statements like Moore's Law exert pressure the big chip manufaturers. Gordon Moore himself has assured that this trend cannot be maintained for a long time [10]. This leads to the search of new paradigms or techniques that allow to sustain the high requirements on performance and energy consumption of applications in the technological world, where a need to process even higher amounts of data is increasing. Some solutions to this increasing demand have appeared, such as multicore computers, multithreading architectures, large scale computers, GPU processing and more.

Despite these solutions, there exist other problems that cannot be solved just by improving on the architecture of the processor. Some of these problems are:

- The memory wall: Wulf and McKee [48] describe an imminent problem on which the superior speed increase of the processors is a lot higher that the improvement on memory technologies. This requires solutions that try to reduce the amount of memory accesses.
- The utilization wall: Taylor et al. [45] noticed a phenomenon that appears due to the increase on the amount of transistors per unit of area in a chip. The problem is related to an exponential reduction of the usable percentage of the chip depending on the scale of integration of the transistors.
- Problems with thermal dissipation: with the increase of frequency of the microprocessors, the level of heat dissipation has increased, forcing a reduction on the operational voltage. Some solutions have been proposed to this problem, like the utilization of multicore processors with cores that deactivate themselves to reduce the workload [15].

Approximate computing tries to solve this type of problems with the help of the recent increase of error-resilient applications (e.g. [42]). This paradigm tries to eliminate, or reduce, the need for precision during processing with the goal of obtaining gains in energy efficiency and processing speed.

Furthermore, one area of interest within the error-resilient application world is machine learning. The goal of this area is to allow a computational system to execute tasks without the need for a prior specific programming and, in some cases, using previous results as feedback to improve the execution. Algorithms used in machine learning look to build mathematical models based on "training" data to perform tasks without explicit programming [7]. Due to their nature, machine learning applications do not present an exact response immediately, or even never, instead they require multiple feedback cycles to achieve the expected response. This means that approximate computing is a potential paradigm to work with this type of applications [5].

1.2.2 Justification of the problem

Approximate computing is an area that is still in the upswing. There are diverse investigations and designs that look to take advantage of the existence of error-tolerant applications. However, it is necessary to continue advancing the paradigm and develop new techniques in order to observe its contributions in daily life. The importance of this paradigm resides on the fact that it does not depend on the current state of the technology to bring forth energetic and performance improvements.

The use of FPGA in the area of approximate computing is still under explored. At the same time, machine learning applications are of great interest to a lot of fields. An improvement on current machine learning implementations through the combination of both areas would generate new opportunities and solutions with low energetic consump-

tion and high adaptability levels. Following this thought, the project is important due to the following:

- It allows to advance the investigation on the area of approximate hardware definition using popular tools such as OpenCL, which could be used as the basis for future investigations with FPGA based applications.
- The use of a popular and easy to use tool could speed up the generation of results in a less explored area, such as FPGA implemented neural networks.
- The project generates new tools easily adaptable to machine learning applications based on neural networks. These tools can be individual kernels or sets of customizable kernels.
- New investigations on error tolerant applications and approximate computing can easily make use of the findings of this project.

1.2.3 Problem definition

Machine learning applications are based on different methods used to obtain results. One of the most used techniques are called neural networks, with the objective of imitating the work done by the human brain to obtain similar results. Furthermore, there is a branch of neural networks called deep learning. It consists in the increase of the amount of processing layer in order to obtain a bigger amount of details from an input of data [36]. Layers are represented by nodes (neurons) that process data from the input. This processing requires a high level of computation, but its results are mostly approximations. CNNs are part of the deep learning neural network implementations, with most of its usage focused on analyzing input images.

Now, FPGAs are devices that have been recently the object of investigations with the objective of accelerating the current processing being done. The interest in using FPGAs for high amounts of computation comes from the limitation of general use CPUs, which do not offer enough processing power (multiple operations with high level of complexity at the same time); and GPUs that, even with the capability of processing high amounts of data at the same time, are not specialized enough or even customizable to perform specific tasks. The use of a device that can be programmed at the hardware level for specific tasks (in this case, neural networks) offers possibilities on processing speed and energy efficiency that surpass GPUs [14].

The current project emerges with the objective of contributing to the ongoing investigation efforts in the area of FPGAs for machine learning applications through approximate hardware definition. Each neuron in a neural network is represented as a "hardware kernel" that gets defined the calculations and processes that it needs to do on the input data to analyze it. Through the use of software tools like OpenCL, it is possible to define kernels that perform machine learning tasks. Combining all these areas and tools, an improvement on performance and energy consumption must be possible, allowing machine learning applications to keep up with the global demand for higher amounts of data

processing.

1.3 Objectives

The project tries to find a solution to the ever increasing energy consumption problem of current world technologies. This is accomplished by using low energy platforms such as FPGAs applied to CNNs with the use of high-level definition tools like OpenCL.

1.3.1 Main objective

Design an approximate hardware implementation of CNNs through the use of Intel's® FPGA SDK for OpenCL™.

1.3.2 Specific objectives

- Describe the viable changes on the OpenCL tool for approximate hardware generation on FPGA.
- Create approximate and reusable hardware kernels for CNNs using OpenCL.
- Determine the error-tolerance level of CNNs when using approximate kernels instead of exact kernels.
- Ascertain the reduction of computational resources when using approximate kernels compared to the use of traditional exact kernels.

1.4 Scope, deliverables and limitations

1.4.1 Scope

The project's end goal is designing, developing and implementing approximate hardware kernels on FPGA. These kernels represent each of the layers of a CNN and will allow for an image processing algorithm to be performed on them. The end result must be more energy efficient and provide better performance over existing implementations of CNNs.

The project is subdivided in the following stages:

Theoretical investigation

A first step is to investigate on the following topics:

- Convolutional neural networks.

- OpenCL and hardware definition.
- Approximate computing on neural networks.
- CNN implementation on FPGAs.

This will provide the necessary information to use OpenCL to implement CNNs on FPGA. Futhermore, this information can also be used to generate mathematical models to measure the error and precision of the generated CNNs, be them exact of approximate, as well as the comparison between both implementations.

Exact CNN implementation on FPGA

An exact CNN implementation must be developed in order to have a baseline in order to develop the approximate kernels. This exact CNN implementation is also used to compare results in order to evidence the performance and energy efficiency improvements. Finally, the approximate CNN implementation must be a modification of the exact model.

Approximate CNN implementation on FPGA

This implementation must be completely based of the exact CNN implementation and will reflect the OpenCL changes that allow for a different hardware definition, one that is approximate and adds errors in the processing of the input data. Also, the approximate implementation must have a set maximum error, which should be measurable and used in the mathematical models.

Comparison and mathematical modeling

Using the results of both the exact and approximate implementation, a mathematical model should be created that reflects the changes in the error between the exact and approximate implementation of the CNN. This model will indicate which parameters could be changed in the approximate implementation in order to reduce or increase the error, as well as the repercussions in performance and energy consumption.

1.4.2 Deliverables

1. Theoretical investigation
    1.1. Compilation report of the use of neural networks on FPGA: this report will contain relevant information to be used in the rest of the project and will guide the decision making when designing neural networks on FPGA.
    1.2. Progress report with the viable modifications: report with the modification possibilities with OpenCL and that affect the hardware definition on FPGA.

2. Design stage

    2.1. Documentation on how to approximate neural networks: contains the necessary information on the principles of approximate computing that are applicable on neural networks, specifically for CNNs. It should contain mathematical models that can be compared against the results of the project.

    2.2. Design of the error calculation model: contains the mathematical formulations that allow to create a precise calculation of the results that will be obtained so they can be compared with the gain in performance.

3. Code development

    3.1. Source code: source code of the OpenCL application and any other code necessary to compile, execute and test the kernels.

    3.2. Exact kernels: source code of the exact kernels used as a base line for the approximate kernels.

    3.3. Approximate kernels: source code of the approximate hardware kernels that will be used on the testing phase.

4. Testing stage

    4.1. Results of the exact tests: measurements of performance, precision and energy consumption while testing the exact kernels.

    4.2. Results of the approximate tests: measurements of performance, precision and energy consumption while testing the approximate kernels.

    4.3. Results documentation: comparison of the results between exact and approximate tests.

5. Project administration

    5.1. Design document: contains the design of every tool developed during the project.

    5.2. Meeting minutes: contains all information obtained in the progress and validation meetings.

    5.3. Final report: final report of the project with all relevant information that was generated during its course.

    have less

1.4.3 Limitations

LIM-01: the student must mobilize to Germany in order to reduce conflicts with the project supervisor. Because of the lack of a visa, the time in the european country is limited to 3 months.

LIM-02: the student's budget is only 1000 euros during the stay in Germany. This money will only cover feeding, staying and commuting costs, limiting the possibility of buying new tools to develop the project. The student must use any tools that the supervisor or the university can give him.

LIM-03: the student must comply with the regulations and limitations of the KIT regarding international students.

LIM-04: no face to face meetings are possible with the advisor teacher due to the geographical location of the student during the project. Due to timezone differences, the available times for meetings is limited to 8 hours or less per day and the meetings depend on having good internet service and working audio equipment.

LIM-05: the project is based on a master thesis developed more than a year before the start of the current project. As such, compiling and executing the prior project limits the progress on the current project.

Chapter 2

Theoretical framework

2.1 Convolutional neural networks

A standard neural network consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons.

Learning is about finding specific weights that make the network exhibit desired behavior, such as driving a car. Depending on the problem and how the neurons are connected, such behavior may require long causal chains of computational stages, where each stage transforms the aggregate activation of the network. Deep Learning is an area of machine learning that tries to replicate this behavior by increasing the number of stages [37]. Figure 2.1 shows the difference between regular NNs and deep NNs.

CNNs are part of deep learning, a type of neural network based on the visual system

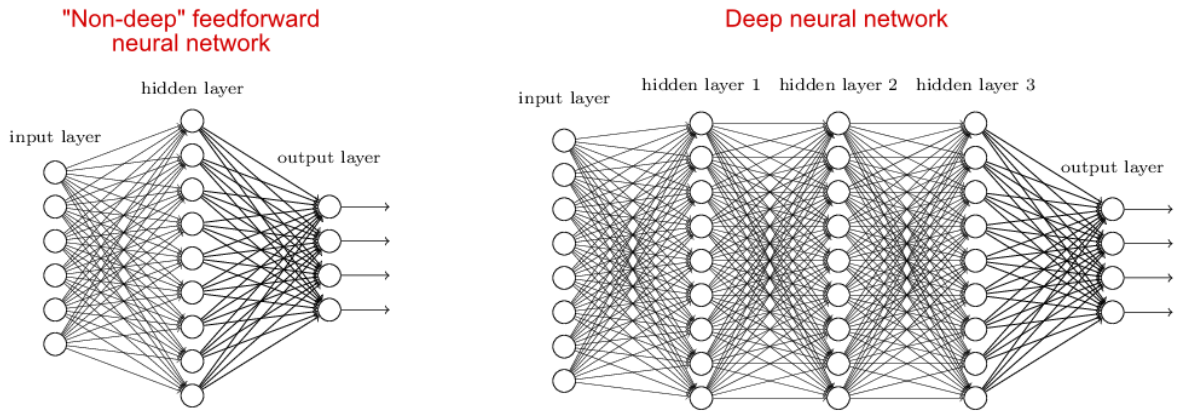

Figure 2.1: Illustration of the difference between a regular neural network and a deep neural network [32]

of mammals [11][16]. A CNN is usually comprised of three types of layers: convolution, pooling and fully-connected [18]. There could also be normalization and activation phases.

2.1.1 Convolution layer

The most important layer in CNNs, as it represents the main operation done throughout the layers of the network and makes it possible to highlight features of the input image. It is based on the convolution operation, defined as such:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.1}$$

However, in CNNs the input image and output result are typically multidimensional arrays with a defined size, which means that convolution must be implemented as a discrete function. Assuming a two-dimensional input I and an applied two-dimensional filter (kernel) the resulting operation is defined as following:

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i-m,i-n)K(m,n) \tag{2.2}$$

If the kernels are smaller than the input image, each of them will only filter a specific window of the image at a time while sliding this window to cover the full image, generating an activation map. Also, because the images are color images, the input will be a three-dimensional matrix which can be transformed (flattened and rearranged) and the convolution applied as a matrix multiplication following the process found on [41].

This sliding window will allow to cover the whole input image by moving a certain number of pixels after every operation, this number is called stride. Also, because the filter operates on every pixel of the window and to ensure a proper output size, a border of ceros could be added to the input image with a specific width, called zero-padding [18]. The last parameter to take into account is the amount of kernels to use per layer. Another relevant term is a feature map, which is any output of the layers conforming the CNN [12].

Suda et al. describes the equation that describes the output of a single neuron convolution operation applied to a specific (x,y) position on the input matrix as:

$$out(f_o,x,y) = \sum_{f_i=0}^{N_{if}} \sum_{k_x=0}^{K} \sum_{k_y}^{K} wt(f_o,f_i,k_x,k_y)in(f_i,x+k_x,y+k_y) \tag{2.3}$$

Where $out(f_o,x,y)$ and $in(fi,x,y)$ represent the neurons at location (x,y) in the feature maps f_o and f_i, respectively and $wt(f_o,f_i,k_x,k_y)$ is the weights at position (k_x,k_y) that gets convolved with input feature map f_i to get the output feature map f_o. This is done for all N_{if} input features [41].

2.1.2 Pooling Layer

A pooling function replaces the output of the NN at a certain location with a summary of the nearby outputs. For example, the max pooling [53] operation reports the maximum output within a rectangular neighborhood. The objective is to discard unnecessary details in the input matrix and preserving the important information.

Just like convolution, a sliding window is used and the pooling operation is performed for each movement of the window. This process reduces the output size (downsampling) [18].

2.1.3 Fully-connected Layer

A fully-connected layer is similar to convolution with the difference being that a convolution layer are connected only to a local region of the input matrix , while a fully-connected one operates over every activation in the previous layer. Their objective is to take every feature found by the other layers and classify them depending on the objects the CNN is trying to find, that is why they need all information gathered by the previous layers. For this reason, fully-connected layers are generally found at the end of the neural network architecture.

2.1.4 Examples of CNN architectures

One of the first successful CNN was LeNet, which was used for number detection and classification on zip codes, digits and more. It consisted of three convolution layers, two subsampling layers similar to pooling and one single fully-connected layer at the end [27].

VGG is another successful CNN which features up to 16 convolution layers, as well as using max-pooling and fully-connected layers [39]. It shows that increasing the depth of conventional CNNs is enough to achieve the necessary performance needed for large-scale image classification.

One of the most import CNNs developed in recent years is the one developed by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. Known as AlexNet, it innovated by stacking convolutional layers and increasing the depth of the CNN [23]. This project uses a partially modified version of AlexNet.

2.2 Approximate computing

Approximate computing is a paradigm that tries to take advantage of the fact that many applications can still work properly even with errors being introduced, be them manually or by accident. Chippa et al. [8] show that current world applications are error-resilient, meaning that they can withstand the introduction of errors while having meaningful

outputs.

Approximate computing can be applied to many fields of computer programming and organization stack. There can be approximate programming languages that introduce imprecise algorithms, to approximate compilers that generate imprecise but fast machine code. There are also approximate computer architectures, going down into approximate memory and hardware that allow for less energy consumption and better performance [49].

2.2.1 Approximate hardware definition

Using approximate hardware is a good approach when having access to transistor-level definition. There have been efforts that try to design approximate logical units [19][51] which could improve the energy levels of existing arithmetic units. But this approach is not viable when working with FPGAs as we can only change the functionality of the hardware instead of its electric properties or composition.

For high-level synthesis, introducing errors must be done partially, as not every part of the hardware can be imprecise. Some of the most important approaches have been:

- Finding the minimum area under a given error rate constraint. Shin and Gupta show that an error of at least 1% can lead to reductions of up to 9% in the area utilized by the design [38].
- Using an iterative process to improve the synthesis by taking into account the error magnitude [29].
- SALSA. A methodology to allow for better control of the quality constraints and transforming the approximate hardware synthesis problem into a regular synthesis problem [44]. There is also an extension to it called ASLAN that works for sequential circuits [33].
- Axilog is an extension to Verilog that allows the designer to choose which parts of the design can be approximate or exact [50].
- Working on fixed point arithmetic can be analyzed and bitwidth optimizations applied for precise control of the accuracy of the output [28].

2.2.2 Approximate neural networks

Agrawal et al. demonstrate that DNNs are resilient to numerical errors from approximate computing. By removing random computations on the convolution layers and using single-precision floating-point number representation, they achieved an error of about 18% [5].

Moreau et al. created a neural accelerator, called SNNAP, that approximates the filter functions and communications between each component within a FPGA-based application [31]. This can be used to generate approximate kernels adapting the SNNAP replacements to OpenCL and the neural network definition.

Another technique is the use of fixed-point arithmetic with less bitwidth for the data being computed inside the neural network. Wang et al. showed that a neural network can be trained using 8-bit floating point numbers without reduced accuracy on the final result [47]. Using fixed-point arithmetic on different layers could be used to change the precision of each of the layers, adjusting them and experimenting to obtain the best results energywise while trying to maintain an accepted output.

Most of the current CNNs in use have a lot of layers for fine tuning the output classification. Simonyan presents various CNN models with different number of layers, finally setting for 16 convolution layers and 3 fully-connected layers [39]. Looking at the results, the other configurations with less layers could be used in order to improve performance, reduce implementation area and reduce energy consumption, while the error is only reduced a little after many layers are included.

2.2.3 Approximate FPGA implementations

Lopes et al. explore the opportunities to explore on playing with the number of iterations on FPGA computations [34]. By using iterative solutions to linear systems, one can adjust the final precision of each calculation and the accuracy by executing less or more iterations. This can be used in conjunction with the operations defined by the neural network kernels to finely control the precision of each layer or the whole network.

As already mentioned, Moreau's neural accelerator can be used to approximate functions implemented on FPGAs. This approach allows to create an specific implementation that does not interfere with other techniques. There is also the framework introduced by Sampson [35], which enables programmers to manually adjust the approximation while providing automation for the process. Sampson showed good results using the framework on a SoC with an FPGA as an approximate accelerator.

There can also be optimization on the arithmetic operations defined on each of the functions to be run on the FPGA. Approximate multipliers can be used to accelerate the performance of matrix multiplications, convolution and other operations that use multiplications [43]. This can be improved upon by ways of memoization, a technique that stores a cache for expensive calculations and returns this cache whenever those calculations are repeated. Using memoization has already been proved to work on FPGA implementations [40] and, while it comes with a small cost on area, it can be used to speed up calculations by large amounts.

2.3 Intel® FPGA SDK for OpenCL™

The use of heterogeneous devices, from general-purpose GPUs to FPGAS, to execute machine learning algorithms and neural networks is of big interest in the current world and there are current efforts to bring all types of devices together [2].

OpenCL is a framework developed by Apple Inc. that allows for a seamless programming experience for heterogeneous devices. OpenCL is based on the C++ 14, including most constructs from C++ in order to work as a clone of it. This allows high-level programmers to develop applications that can be run on different platforms without having to create new code for any new platform.

Intel® FPGA SDK for OpenCL™is a development environment that allows the creation of synthesizable kernels, blocks of code that can be executed within the FPGA. Because of the high-level focus of OpenCL, it can be used to design in short time big application such as CNNs. As for its effectiveness, it has been demonstrated that using OpenCL instead of low-level languages such as Verilog does not yield bad performance results, at the cost of a little more memory usage [3].

Suda et al. [41] implement an FPGA accelerator for CNNs. They take advantage of the parallelism offered by OpenCL in order to set vectorization and loop unrolling factors as to have better control of the actual implementation on the FPGA. Also, Suda's implementation applies convolution using matrix multiplication, achieving good performance on these layers. Suda uses AlexNet and VGG to benchmark its results, finding that it can be significantly better than the CPU time.

Wang et al. [46] created a reconfigurable accelerator for CNNs based on OpenCL. This implementation features OpenCL's extended channel support on FPGAs enabling communication between layers without using onboard memory, allowing for reduced memory usage and faster communication between layers. This work demonstrates high performance and shows the low energy consumption of FPGAs. It also uses AlexNet and VGG to get its results.

Chapter 3

Methodology

The current project is a research on the effect of approximate computing techniques to the improvement of performance and energy consumption on FPGAs. First, the research will be classified based on different perspectives. The classification will be followed by the methods to used on creating the CNNs and validating the changes in execution time, energy consumption and accuracy degradation.

3.1 Type of research

The following classification is based on the different perspectives created by [24]

3.1.1 Application perspective

This work is categorized as an applied research. The objective is to apply concepts from theory, in this case approximate computing, into another field of work or active research such as the study of CNNs on FPGAs. These concepts will be mixed with other topics from computer architecture and hardware generation.

3.1.2 Objectives perspective

From an objectives standpoint, this is an exploratory research due to trying to explore the possibilities of performance and energy gain through the use of approximate hardware generation. The area of approximate computing on FPGAs, specifically for CNN implementations, is currently underexplored. This research will help grow the field and could enable future research or applications on using FPGAs to accelerate the field of machine learning.

3.1.3 Mode of enquiry

Finally, this is an unstructured research. The objective is to find how different techniques affect the final measurements and tests in relation to exact kernels. The specific techniques to apply are not predetermined, just as the expected accuracy after applying the different modifications to the exact kernels. This project tries to push performance to the maximum and energy consumption to the minimum while maintaining an acceptable accuracy, but there is no set number for any of these values.

3.2 Kernel creation

The research is based on modifications done to an exact CNN implementation on an FPGA. The next sections specify the methods used to generate the kernels and apply the approximate modifications.

3.2.1 Exact kernel generation

The exact kernel generation will be based on the CaffeNet implementation. This is an AlexNet implementation with the pooling and normalization layers reversed [9].

The kernels are implemented using OpenCL on a DE1-SoC board, a hardware board with an on-chip processor and a Cyclone V FPGA. Two codes must be generated, the host code to control the execution that will be executed on the embedded chip of the board and the device code that will perform the CNN calculations.

The kernels already contain some approximations due to limitations on the FPGA being used, mainly on using fixed-point values, as it does not contain enough resources to implement a non fixed-point implementation of the CaffeNet neural network.

3.2.2 Approximate kernel generation

The approximate kernels will be generated based on modifications done on the exact kernels. Different techniques will be used to apply these modifications. Any modification must be tested and different combinations of these modifications must be applied to show the change on accuracy, performance and energy consumption.

The modifications must be applied on varying levels of abstraction, measuring the output on different layers up to the output of the full CNN.

3.3 Validation

In this section, the validation to be used on the project is explained.

3.3.1 Accuracy

An algorithm based on the formulas found on [13] is created using the Python. This algorithm calculates accuracy of the approximate modifications against a Caffe [17] based AlexNet implementation, also written in Python.

The training weights to be used for evaluation are the ones found on [9]. The original network was trained using the Image-net Large Scale Visual Recognition Challenge 2012 image set [6]. The measurements will be done for top-1 and top-5 accuracy.

3.3.2 Energy consumption

For energy consumption, an initial measurement will be done with the FPGA doing no work, then after loading the program, a new measurement will be done. This will be compared to the same measurement when loading the configuration with no modifications done. This energy consumption will also be compared against the energy consumption of the computer while running the Python implementation of CaffeNet.

3.3.3 Performance

Performance will be measured by a simple subtraction of time measured at the end of the image classification and at the start. This will be compared to the performance of the Caffe implementation and the exact implementation with no modifications.

Chapter 4

Description of work

En este capítulo se exponen los diseños experimentales realizados para comprobar el funcionamiento correcto del sistema. Por ejemplo, si se realiza algún sistema con reconocimiento de patrones, usualmente esta sección involucra las llamadas *matrices de confusión* donde se compactan las estadísticas de reconocimiento alcanzadas. En circuitos de hardware, experimentos para determinar variaciones contra ruido, etc. También pueden ilustrarse algunos resultados concretos como ejemplo del funcionamiento de los algoritmos. Puede mostrar por medio de experimentos ventajas, desventajas, desempeño de su algoritmo, o comparaciones con otros algoritmos.

Recuerde que debe minimizar los "saltos" que el lector deba hacer en su documento. Por tanto, usualmente el análisis se coloca junto a tablas y figuras presentadas, y debe tener un orden de tal modo que se observe cómo los objetivos específicos y el objetivo general del proyecto de tesis se han cumplido.

Chapter 5

Conclusions and recommendations

Las conclusiones no son un resumen de lo realizado sino a lo que ha llevado el desarrollo de la tesis, no perdiendo de vista los objetivos planteados desde el principio y los resultados obtenidos. En otras palabras, qué se concluye o a qué se ha llegado después de realizado la tesis de maestría. Un error común es "concluir" aspectos que no se desarrollaron en la tesis, como observaciones o afirmaciones derivadas de la teoría directamente. Esto último debe evitarse.

Es fundamental en este capítulo hacer énfasis y puntualizar los aportes específicos del trabajo.

Es usual concluir con lo que queda por hacer, o sugerencias para mejorar los resultados.

Bibliography

[1] URL: https://www.google.com/maps/place/Baden-W%C3%BCrttemberg/%7B@%7D48.6618471,9.0036649,8z/data=!3m1!4b1!4m5!3m4!1s0x47911b62826406df:0xc68c48bf0d244860!8m2!3d48.6616037!4d9.3501336?hl=en.

[2] Martın Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[3] Mohamed S Abdelfattah, Andrei Hagiescu, and Deshanand Singh. "Gzip on a chip: High performance lossless data compression on fpgas using opencl". In: *Proceedings of the international workshop on openCL 2013 & 2014*. ACM. 2014, p. 4.

[4] Kamel Abdelouahab et al. "Accelerating cnn inference on fpgas: A survey". In: *arXiv preprint arXiv:1806.01683* (2018).

[5] Ankur Agrawal et al. "Approximate computing: Challenges and opportunities". In: *2016 IEEE International Conference on Rebooting Computing (ICRC)*. IEEE. 2016, pp. 1–8.

[6] Alex Berg, Jia Deng, and L Fei-Fei. *Large scale visual recognition challenge 2010*. 2010.

[7] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[8] Vinay K Chippa et al. "Analysis and characterization of inherent application resilience for approximate computing". In: *Proceedings of the 50th Annual Design Automation Conference*. ACM. 2013, p. 113.

[9] Jeff Donahue. *BVLC Reference CaffeNet*. 2012.

[10] Manek Dubash. "Moore's Law is dead, says Gordon Moore". In: *Techworld. com* 13 (2005).

[11] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[13] Google. *Classification: Precision and Recall*. URL: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall.

[14] Kaiyuan Guo et al. "A survey of fpga-based neural network accelerator". In: *arXiv preprint arXiv:1712.08934* (2017).

[15] Jim Held, Jerry Bautista, and Sean Koehl. "White paper from a few cores to many: A tera-scale computing research review". In: (2006).

[16] David H Hubel and Torsten N Wiesel. "Receptive fields and functional architecture of monkey striate cortex". In: *The Journal of physiology* 195.1 (1968), pp. 215–243.

[17] Yangqing Jia et al. "Caffe: Convolutional architecture for fast feature embedding". In: *Proceedings of the 22nd ACM international conference on Multimedia*. ACM. 2014, pp. 675–678.

[18] Andrej Karpathy et al. "Cs231n convolutional neural networks for visual recognition". In: *Neural networks* 1 (2016).

[19] Yongtae Kim, Yong Zhang, and Peng Li. "An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems". In: *Proceedings of the International Conference on Computer-Aided Design*. IEEE Press. 2013, pp. 130–137.

[20] KIT. *KIT logo*. URL: http://www.kit.edu/img/intern/10jahre_de.svg.

[21] KIT. *Organization and Governance*. URL: https://www.kit.edu/kit/english/organization.php.

[22] kjunix. *Map of Baden-Württemberg*. URL: https://en.wikivoyage.org/wiki/Baden-W%C3%BCrttemberg#/media/File:Baden-Wuerttemberg_travel_map_EN.png.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[24] Ranjit Kumar. *Research methodology: A step-by-step guide for beginners*. Sage Publications Limited, 2019.

[25] Suhas Kumar. "Fundamental limits to Moore's Law". In: *arXiv preprint arXiv:1511.05956* (2015).

[26] Suhas Kumar. "Fundamental limits to Moore's law". In: (Nov. 2015). DOI: arXiv:1511.05956.

[27] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[28] Chaofan Li et al. "Joint precision optimization and high level synthesis for approximate computing". In: *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2015, pp. 1–6.

[29] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. "Approximate logic synthesis under general error magnitude and frequency constraints". In: *Proceedings of the international conference on computer-aided design*. IEEE Press. 2013, pp. 779–786.

[30] Bert Moons et al. "Energy-efficient convnets through approximate computing". In: *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2016, pp. 1–8.

[31] Thierry Moreau et al. "SNNAP: Approximate computing on programmable socs via neural acceleration". In: *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2015, pp. 603–614.

[32] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA: 2015.

[33] Ashish Ranjan et al. "ASLAN: Synthesis of approximate sequential circuits". In: *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association. 2014, p. 364.

[34] Antonio Roldao-Lopes et al. "More flops or more precision? accuracy parameterizable linear equation solvers for model predictive control". In: *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE. 2009, pp. 209–216.

[35] Adrian Sampson et al. "Accept: A programmer-guided compiler framework for practical approximate computing". In: *University of Washington Technical Report UW-CSE-15-01* 1.2 (2015).

[36] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[37] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural networks* 61 (2015), pp. 85–117.

[38] Doochul Shin and Sandeep K Gupta. "Approximate logic synthesis for error tolerant applications". In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE. 2010, pp. 957–960.

[39] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[40] Sharad Sinha and Wei Zhang. "Low-power FPGA design using memoization-based approximate computing". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.8 (2016), pp. 2665–2678.

[41] Naveen Suda et al. "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks". In: *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2016, pp. 16–25.

[42] Darshan D Thaker et al. "Characterization of error-tolerant applications when protecting control data". In: *2006 IEEE International Symposium on Workload Characterization*. IEEE. 2006, pp. 142–149.

[43] Salim Ullah, Sanjeev Sripadraj Murthy, and Akash Kumar. "SMApproxlib: library of FPGA-based approximate multipliers". In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE. 2018, pp. 1–6.

[44] Swagath Venkataramani et al. "SALSA: systematic logic synthesis of approximate circuits". In: *DAC Design Automation Conference 2012*. IEEE. 2012, pp. 796–801.

[45] Ganesh Venkatesh et al. "Conservation cores: reducing the energy of mature computations". In: *ACM SIGARCH Computer Architecture News*. Vol. 38. 1. ACM. 2010, pp. 205–218.

[46] Dong Wang, Ke Xu, and Diankun Jiang. "PipeCNN: An OpenCL-based open-source FPGA accelerator for convolution neural networks". In: *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE. 2017, pp. 279–282.

[47] Naigang Wang et al. "Training deep neural networks with 8-bit floating point numbers". In: *Advances in neural information processing systems*. 2018, pp. 7675–7684.

[48] Wm A Wulf and Sally A McKee. "Hitting the memory wall: implications of the obvious". In: *ACM SIGARCH computer architecture news* 23.1 (1995), pp. 20–24.

[49] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. "Approximate computing: A survey". In: *IEEE Design & Test* 33.1 (2015), pp. 8–22.

[50] Amir Yazdanbakhsh et al. "Axilog: Language support for approximate hardware design". In: *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium. 2015, pp. 812–817.

[51] Rong Ye et al. "On reconfiguration-oriented approximate adder design and its application". In: *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE. 2013, pp. 48–54.

[52] Jialiang Zhang and Jing Li. "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM. 2017, pp. 25–34.

[53] Yi-Tong Zhou and Rama Chellappa. "Computation of optical flow using a neural network". In: *IEEE International Conference on Neural Networks*. Vol. 1998. 1988, pp. 71–78.

Appendix A

Demostración del teorema de Nyquist

El título anterior es solo un ejemplo ilustrativo. Éste teorema no ameritaría un apéndice pues es parte normal del currículum de Electrónica, pero apéndices usualmente involucran aspectos de esta índole, que se salen de la línea de la tesis, pero que es conveniente incluir por completitud.

Los anexos contienen toda información adicional que se considere pertinente agregar, como manuales de usuario, demostraciones matemáticas que se salen de la línea principal de la tesis, pero que pueden considerarse parte de los resultados del trabajo.