USING COMPUTING FIRST PRINCIPLES TO IMPROVE THE SYMBIOTIC PERFORMANCE IN ALGORITHMS AND PROCESSORS USED IN LOW-POWERED MACHINE LEARNING

A Dissertation Presented

by

Robert Nsinga

Submitted to the Graduate School of
Advanced Technology and Science in partial fulfillment
of the requirements for the degree of

DOCTOR OF ENGINEERING

September 2022

Systems Innovation Engineering

DEDICATION

To God Almighty.

To my mother, who wanted more for me.

ACKNOWLEDGMENTS

This work would not be completed without the support of numerous people and organizations I herein wish to recognize.

I thank my advisor, Asst. Prof. Stephen Karungaru, for his wealth of experience and for giving me the freedom to pursue my own research interests. I wish to recognize Prof. Kenji Terada, the staff of the Terada lab (B1 lab), and the Dean office of the faculty of engineering for their unwavering support.

I am grateful to the staff and management of CIS Corporation Research and Development department for their support in conducting my experiments, and for the timely advice on technical matters.

I owe a great deal to the people of Japan who made my stay memorable. Witnessing your values and traditions have shaped me into a better person, and for that I am grateful. To the friends I made along the way, thank you.

ABSTRACT

Using less electric power or speeding up processing is catching the interests of researchers in deep learning. Models have grown in complexity and size using as much precision depth as can be computationally supported regardless of how expensive the minimum required cooling system might cost. Quantization has offered ease of deployment to small devices lacking floating precision capability, but little has been suggested about the floating numbers themselves. This thesis evaluates hardware acceleration for embedded devices that cannot support the energy requirements of floating numbers and proposes solutions to challenge the limits of power consumption and apply them to measure their effectiveness in terms of energy demand and speed capacity.

Experts have declared the end of Moore's law with the current state of nanotechnology coming to terms with its inability to increase the performance per transistor density ratio. Accelerators, although providing a countering measure, have also increased their power needs to unsustainable levels. At the same time there has been sufficient increase in knowledge, such as distributed computing, to branch-off into possibilities that could reduce power demands while maintaining, or possibly increase microprocessor performance. This thesis highlights some important challenges that were born out of the rapid rise of deep learning.

We present experimental results showing that low-powered devices can serve as powerful tools in low cost deep learning research. In doing so we are interested in slowing down the ongoing trend that favors expensive investment in deep learning computers. Using known properties in computer architecture, hardware acceleration, and digital arithmetic we implement ways to design algorithms that symbiotically match their performance in accordance with the theoretical limits afforded by the hardware components that run them.

Computer processors are utilized based on their ability to execute instructions defined in code or machine-readable format. Some processors are multi-purpose, others are domain-specific, the former being good at a wide range of tasks and the latter only focused for specific tasks. While executing any task an ideal processor should engage all its transistors to ensure that no part is left underutilized. However, in practice it is not always the case, which is why domain-specific processors are optimized to carry only the instructions for which they would fully commit their components.

It is considered good practice when algorithms are designed to encourage the maximum use of available capacity for any execution. Our proposed method improves the symbiotic complementarity in peak algorithm performance and theoretical hardware capacity.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Since as far back as historical records show, humans have always attempted to recreate intelligence, either through artificial means, or by providing cognitive training to animals. Around the middle of the 20th century the general academic consensus revolved around giving scientific super-calculators, the predecessors to modern computers, the ability to reason. New discoveries would then be tested against mathematics theorems such as the *Principia Mathematica* that were exclusively used to reason about new mathematical proofs.

In recent years deep learning has emerged considerably and cemented itself as a part of the daily discourse on scientific progress and cutting-edge technology. Multiple advances were registered in many other fields that chose to use deep learning methods in new or existing computer-assisted practices. Deep learning is an umbrella term under the machine learning field, another umbrella term. Machine learning comprises theories, tasks, algorithms, devices, etc. used to develop techniques, software and hardware used in solving a number of problems, generally using computers. Along with the rapid emergence of deep learning, expensive computer configurations and intimidating verbiage, artificial intelligence has grown into complexity and made it hard to get started in the field. At the same time there has been a mounting cost in terms of time, money, and knowledge. This research is meant to address another cost associated with the measure of performance in using deep learning.

The true potential of computer systems lies within two essential capabilities: to automate given tasks and to help make informed decisions. From the beginning of the modern computing era these two capabilities have been harnessed, improved, and applied to the workplace, to transportation, to businesses and industries, to science and space, to personal and home needs and to many other lifestyles.

Most machine learning systems are data-based, i.e., they use data that has been prepared in advance. This data scheme has grown tremendously in ways that pushed the costs highlighted above even higher. Most machine learning systems are made of parameters that are tuned by computations in a process called training. Training may take weeks or days depending on the performance of the computing platform in use. Any machine learning practitioner is then responsible for making the right, cost-effective choices that would afford them the best solutions. A priori there is the initial cost associated with the time and effort invested in acquiring adequate knowledge and expertise which directly correlates with the capacity to choose the techniques, methods, algorithms, tasks, software, and hardware needed for a particular study. It is at this juncture that the scientific community is divided based on access to knowledge and tools. On one side there is heavy industry investments in super-computers capable of covering for most if not all needs above. On the other side there is everyone else trying to make do with available means, either a laptop or a PCB (printed circuit board) computer. This divide has pushed the community to invest capacity in understanding and improving the performance of machine learning systems with minimal cost.

Machine learning systems help uncover patterns from large amounts of data. The generated patterns constitute their understanding of the problem, be it in self-driving cars, healthcare, language synthesis, or recommender systems. Large data takes large computations to process as their “understanding” of the data is largely built from primitive operations such as accumulation and multiplication of floating-point numbers, known to be error prone [1]. Human interference is meant to provide feedback for the computer to adjust its accuracy, which improves the computer’s representation of the information, resulting in better predictions. In this context the “understanding” of the data and the representation of information could be used interchangeably.

1.1 High-performance computing

Scientific research is best explained with data in mind. It is a collection of techniques and rigorous measures, depending on the field, ordered in a manner which collects, processes then transforms raw data into useful information that advances knowledge in said field. Contextually, in machine learning as is the subject of this thesis, the transformation is not meant to generate useful information but rules that a computer can use to understand the data, or new, unseen data. Over the years computers have evolved a tool of choice for scientific researchers (Figure 1). In fact, as computer performance and computer networking evolved, scientific research made greater strides to tap into the vast capabilities presented by these evolutions. Suddenly, problems that could take months or years to solve could be answered much sooner by assigning as much computing power as necessary.

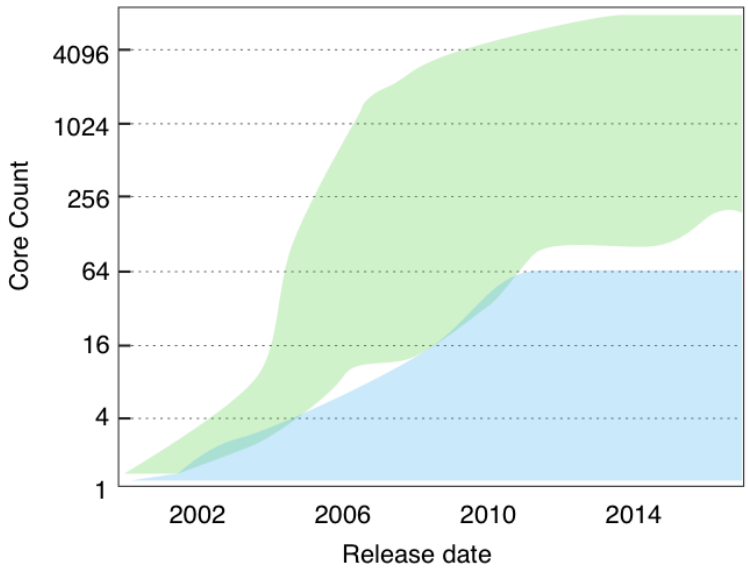


Figure 1: Evolution spaces in the number of processing cores for GPU (green color) and CPU (blue color) for computing architectures

With the possibility to ask new questions came the need to gather more data, at the expense of infrastructure capacity, or to avoid the risk of stalled research or diminishing utility. The core count for CPU stalled at about 64 cores since around 2010, which meant that it wasn't possible to increase the

performance simply by increasing the number of cores, unlike GPUs which maintained a steep climb to date. Writing applications that are sequential in their execution style became harder which gave rise to parallel execution.

To prevent any delay or slowdown, resource pooling became a way to bypass certain issues. High-performance computing is a resource pooling mechanism that aggregates many computers to act as one unit, with the underlying objective that the more computers are clustered the faster research could accelerate, turning months into mere hours or minutes. But another objective is the possibility to answer multiple questions at the same time. Parallel computing is a paradigm that essentially breaks down pooled resources into independent or semi-independent operators for simultaneous execution. Each operator can expand or shrink its resources according to its needs. The main advantage of resource pooling is to increase capacity in terms of processing, storage, or access (networking).

Let's consider the traditional machine learning research steps:

- *Training*: Assumes data collection, model definition and data learning
- *Validation*: Uses a sample of data to evaluate the learning process
- *Testing*: Applies new, unseen data to assert the learning effectiveness

The first step is usually the longest because it is hard to automate completely. Data collection requires human participation to annotate or label the data correctly. Human involvement is often referred to as human-in-the-loop and is a bottleneck area that is extremely difficult to assign to a computer. This thesis proposes to maintain human involvement as a crucial practice by improving the tools that would instead accelerate human effort or save it. That would enable cost-savings on several aspects of scientific research.

In a machine learning research activity, experimentation is halted as soon as the testing phase achieves acceptable results based on the collected data. However, in real-world environments data is continuous, which may drag research into a continuous loop where the steps above are repeated to take into

consideration the latest data. Instead of repeating the training from scratch every time, a method referred to as “fine-tuning” is used.

High-performance computing is also an amalgamation of several computing fields including systems administration, parallel programming as briefly stated above, digital electronics, computer architecture, system software, programming languages, algorithms, and computational techniques [2]. And its use in machine learning is an unfortunate series of choices that begin with the need for more. This has propelled research into assuming the best results should be associated with costly choices, therefore making machine learning an expensive and exclusive research direction reserved for the wealthy and resourceful organizations and individuals. Along the evolution of computing, and following Moore’s law, it became clear around the turn of the 21st century (1999-2001) that it was not possible to increase the performance of transistors without negatively affecting voltage or other sensible circuitry parameters. It was then that the idea of having transistors grouped together in smaller units for the purpose of carrying computations independently from another group of transistors took off under the now common multi-core and/or multi-threaded processing. Table 1 shows the constraints associated with increasing electrical power per millimeter-squared of processing area. Soon, a limit forced by inadequate or expensive cooling, or both inadequate and expensive cooling.

Table 1: Scaling results for circuit performance showing electrical properties of transistor scaling, for $k \in \{1,..., 10\}$ (Source: Robert Dennard, IEEE)

Device or circuit parameter	Scaling factor
Device dimension t_{ox}, L, W	$1/k$
Doping concentration N_a	k
Voltage V	$1/k$
Current I	$1/k$
Capacitance eA/t	$1/k$
Delay time/circuit VC/I	$1/k$
Power dissipation/circuit VI	$1/k^2$
Power density VI/A	1

HPC is not to be confused with super-computing. It used to be the case that pooling was targeted from a single computing infrastructure the size of several rooms, but nowadays it is associated with clusters, cloud, or virtualization because of the internet. HPC is generally associated with research, but engineers and businesses also use it for computer-aided design (CAD), big data mining, or transaction processing in finance.

High-performance computing (HPC) remains the de facto environment for many deep learning projects. Apart from the architectural complexity that goes into designing, operating, and maintaining HPC, this research also argues against its relatively elevated costs, financial or otherwise. Notwithstanding the continued investments from both the academia and industry to achieve high results with minimal consumption. To tap into low resources computing alternatives it is important to discuss HPC's appeal in the first place.

Deep learning is a subset of machine learning that deals with repetitive, layered computations that exploit patterns in data by associating them with weights. This is made possible in part because of the sheer amount of data, and on the other part because of the flexibility and composability of the mathematics principles used. As one could easily conclude, the more capable a computer, the more expensive it is. Not only that, but also the more environmental waste it expels, which introduces environmental considerations in the practice of machine learning. The nascent field of deep learning is already regarded as a powerful problem-solving technique that any individual should consider. Associated with this growing sentiment is the resurgence of domain-specific architectures (DSA) that attempt to leap over the barrier caused by the slowing of Moore's law without further aggravations to environmental conditions.

As highlighted in Table 1, domain-specific or application-specific processing came by after it was increasingly difficult to maintain acceptable power consumption levels at the same level as the cooling technology state-of-the-art. Instead of having transistors used for multi-purpose processing,

accelerators were devised to focused on specific types of processing that later favored and nurtured the sudden growth realized in deep learning.

Recent computer designs have pushed towards specializations that maximizes computational resources such as storage, memory, and processing, to push performance to higher limits while keeping costs manageable. When it comes to the manufacturing of computers the standards have not been as homogeneous as expected to provide a means for cross-compatibility across varying use-cases, form factors, or demands. The personal computer, or PC, is the most striking proof of this homogeneity, especially in the design of its processing architecture. When it comes to machine learning and deep learning in particular, these architectural designs underperform and therefore the need to branch off in the direction of the demands of machine learning create a breach in the pursuit for homogeneous computing. New types of processing architectures are required, and old methods are revisited to maintain cross-compatibility. For example, industry-leading efforts have explored quantum computing for large machine learning computations and others have revisited analog computing to replace digital computing.

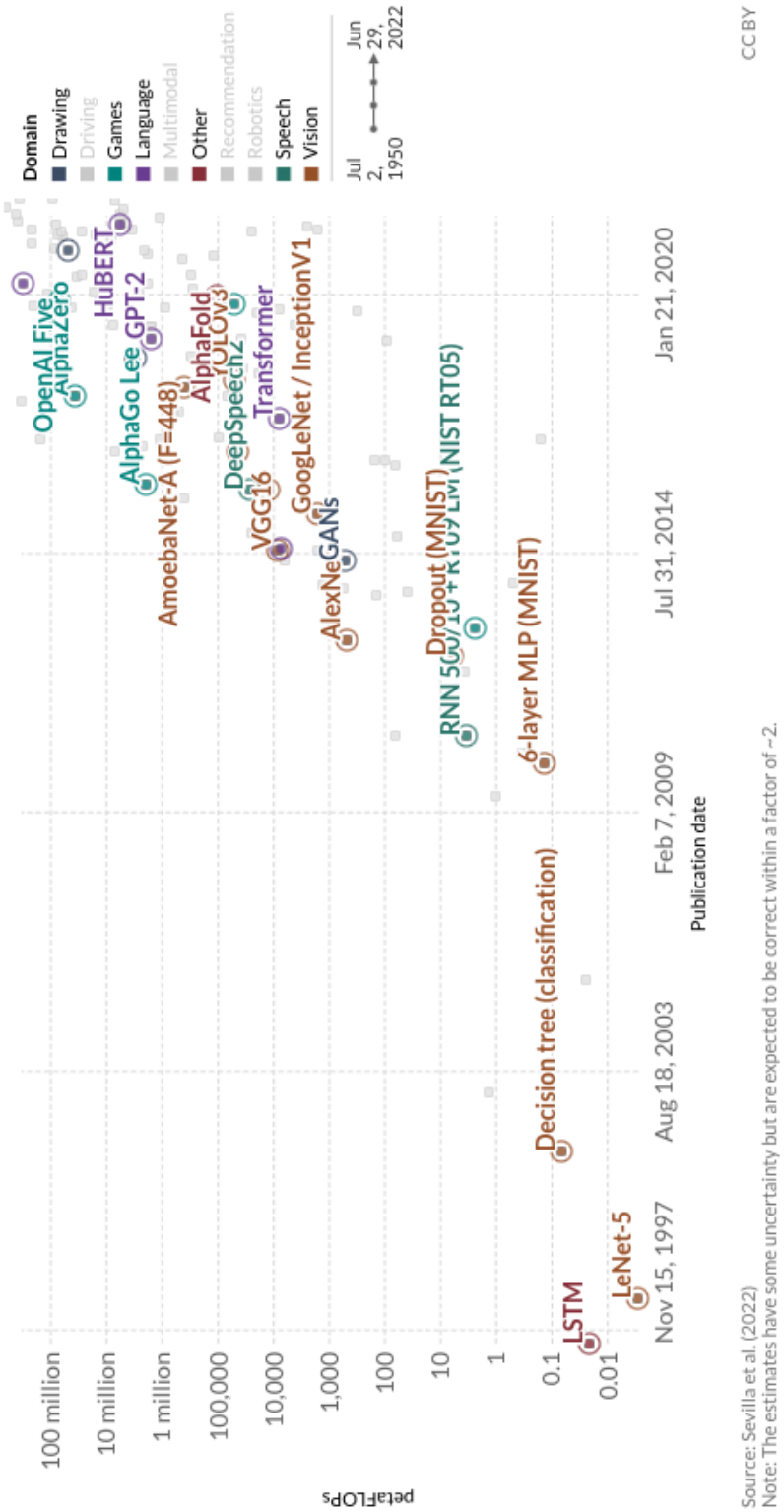


Figure 2: Evolution of model parameters with estimated computation (in petaFLOPS) required in large AI training runs. Source: ourworldindata.org

Several considerations, other than market trends such as shown in Figure 2, come into focus when building high-performance computers, including:

- *Parallelization*: Clusters of DSA arranged in a specific manner can mitigate the slowdown of computation that arises from the sequential nature of processing steps. Parallel resources can therefore split up and share computations in a simultaneous orchestration. This has immediate effect on the total time it takes to process. It also requires lots of electrical power.
- *Memory*: Weights generated from learning patterns need to be accessed multiple times by multiple processes. Disc reads and writes cannot sustain this traffic alone. Temporary storage such as RAM needs to be considerably vast to accommodate this demand.
- *Computation*: Although machine learning computations are merely accumulations and multiplications, the amount of data to process can quickly overwhelm the most performant CPU. More cores need to be activated at a time, sometimes in the order of several hundreds. Figures 1 and 2 show a correlation in core count increase and the explosion in large models over the past two decades.
- *Flexibility*: In a repetitive layered approach such as is expected in deep learning, algorithms switch quickly and are overturn by new methods or best-practices that are not bound to one specific type of architecture. Therefore, architectural designs need to account for flexibility and expect shifting configurations, workloads, or techniques.
- *More data*: Along with storage considerations, there is also concerns about the format in which data is stored. 32-bit floating point format is the industry format for its capacity to capture precision. However, the same format is known to be error-prone and take up lots of resources such as memory and electrical energy.

A modern super-computer is critically measured against all the above listed features. However, the defining factor is its interconnect design, or how the components communicate together. The way signals move from one component

to another, say from memory to processing, is an area of engineering prowess that large companies invest in without hesitation. Designing a super-computer involves arranging its components to improve interconnect speeds, most termed as reducing latency.

Parallelization is implemented at the hardware level and configured to interface with software through firmware or other proprietary software or libraries. Multi-core processing can be generalized or specialized, and market trends continue to show upward movement in both capabilities.

1.2 Embedded systems

Taken at face value, an embedded system is nothing more than some processing chip, some internal memory, and some peripherals to connect to the outside, put together to perform some computations. The market perception associated with this understanding has contributed more details such as the size, shape, and form factor of an embedded system. Generally, an embedded system has a domain-specific purpose that involves controlling a larger mechanical or electronic system in which it is embedded. Embedded systems are typically small and require very little power to function. One other important aspect of embedded systems is that they are perpetually in use i.e., there is no on/off switch or some sort of booting routine. It should therefore come to no surprise that HPC may include embedded systems.

A single unit of embedded system may also include an accelerator, or several accelerators, that offer parallel computing capacity within the low-powered limitations. Their size presents a versatility that is boundless. Embedded systems can be found everywhere, in more utilities than humanly imaginable such as cables, watches, rings, earphones, light bulbs, etc. Miniaturization in the computing environment has become just as vast and complex as the parent field itself, bridging many gaps while at the same time improving the symbiotic

relationship between humans and computers. As this thesis demonstrates, a common class of dedicated accelerators is the digital signal processor or DSP. It has a particular property that incidentally makes machine learning possible.

Given the market forces governing where efforts should focus, the machine learning community has devised standards and categories. The pursuit of high performance has given rise to sophisticated manufacturing technologies that compete to cram as much processing power as possible onto as little surface as possible. On one hand there are the maximalists who opt into high performance along with energy requirements such as cooling systems and expensive manufacturing, and on the other hand the minimalists who are keen on low-powered, low-resource computing devices that maintain acceptable performance. However, research has shown that embedded computing has evolved based on similar principles to produce powerful devices that compete with personal computers such as laptops or smartphones.

Depending on which side one stands on, there are 2 distinct types of processing influenced by the type of machine learning task:

- Inference only: this is a type of processing that is concerned with running predictions only. That is, the host or processor is pre-loaded with all weights and algorithms necessary to deliver expected estimates when queried.
- Training and inference: this type of processing can learn to generate its own weights and algorithms before it provides predictions.

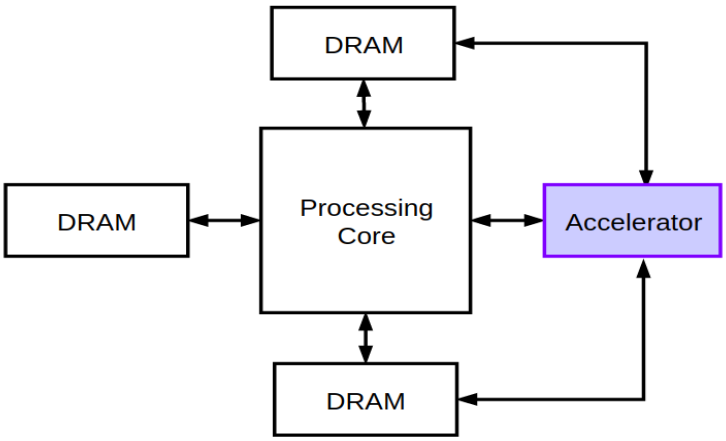


Figure 3: Diagram showing how the central processing unit shares resources with the accelerator (in light purple) in a parallel computing architecture

Despite the many benefits low-resource computing present, the machine learning has slowly adopted its use compared to HPC. Figure 3 presents the basic arrangement of a computing architecture that HPC or embedded systems emulate, often called the von Neumann architecture. We go through the main reasons why this is the case and present experimental analysis for promising solutions We propose targeting embedded systems for various social-economic reasons, and our experiments focus on addressing some of the limitations caused by IEEE 754-2008 floating points computations.

The motivation behind targeting low-powered devices has long been argued even though the most cited applications of deep learning are utilitarian in nature, and thus demand to be deployed closer to their intended users, on their personal computers, cell phones, or cameras, for instance. Deep learning has hindered the motivation for low-powered devices by increasing the computation complexity with parallel computing such as is shown in Figure 1 or giving rise to alternative options like distillation, model pruning and quantization [2] [3].

Floating numbers are a good choice to avoid precision decay over multiple rounds of operations. For this reason, the deep learning community has given preference to expensive computer systems for training with floating numbers, and the resulting models are converted to integers format for smaller form factors like

embedded systems. However, research has shown that replacing floating numbers with integers does not deal a heavy toll on the accuracy of a model but instead improves its storage and access speed [4].

Deep learning has shown potential in other areas like healthcare, home consumer products, and the creative industry. There is growing interest that is pushed back by the relatively high barrier of entry caused by expensive computing requirements and complex verbiage. Transfer learning has provided some improvement, but more efforts are required to allow anyone to contribute new ideas from scratch, free from any prior limitations or cumbersome requirements.

1.3 Overview of experiments

We formulate and analyze a configuration based on a mix-and-match of existing solutions in deep learning. We are interested in testing embedded systems capabilities when applying these solutions. Our approach applies to the mathematical operations at the core of computations.

Specifically, we modify arithmetic operations in existing software algorithms to manipulate a different type of number notation to assess whether models can be trained without precision loss on operations like convolution or automatic differentiation and challenge the boundaries of power and latency for embedded systems.

In general, ongoing contributions can be split into the hardware and software distinctions where each proposes specialized methods. A third distinction that is a hybrid software-hardware mix tries to challenge the shortfalls in linking innovations in hardware and the rapidly changing software. This work is on the latter distinction, using domain-specific accelerators and software. For our experimentations, hardware and software considerations include existing proprietary and open-source devices and libraries.

1.4 Scope and key assumptions

The paradigm shift introduced with the popularization of machine learning, especially deep learning, has reaffirmed the importance of human supervision, especially in the early stages of experiments. It brought to prominence human-in-the-loop (HITL) systems. In AI, HITL enables human intervention at all stages of the workload to verify, correct, or abort processes. It can present itself as a dashboard UI giving broad view of all resources engaged in the workload, or it can be a set of standard checklists and kill switches that are always within reach. HITL is beneficial to critical applications such as healthcare, financial services, governance, or others. Benefits such as risk mitigation, exception handling, productivity monitoring, cost control, or data completeness are not only helpful at the beginning, or at the end, but throughout the entire workload.

Let's consider healthcare with its data-dependent and extremely risky industry. Not only does the approximations have to be always spot-on, but they must also remain reliable for future decisions, not only present actions. Machine learning sounds like the perfect medicine to handle this situation, but recent years have shown mounting claims that AI is a black box of unexplainable processes and algorithms, sometimes rendering human experience in the field baseless while achieving results that cannot be traced or assessed. Although dealing in approximations, machine learning is also making good progress in robotics and self-driving cars, two fields that rely on precision. The only way to always be precise is to fully understand how that precision was approximated, or at the very least always maintain human presence. If automation's objective is to limit human presence, then HITL is a contrarian endeavor.

The role of AI in healthcare is to support personnel in decision-making, and HITL is a feedback loop that preserves human experience and incorporates it in the approximation workload.

Because we are interested in the limitations of floating numbers on small form factors like embedded systems, we outline key heuristics or conditions. First and foremost, we are not interested in any type of dataset, nor are we measuring the accuracy or overall effectiveness of a deep learning model. Instead, we solely consider computations which manipulate floating numbers and analyze their performance. In other words, we are discussing the engineering side of deep learning computations. Rooted in scientifical computations, machine learning is as deep as it is wide, touching almost every corner of engineering disciplines, lifestyle, or experimental research such as deep space exploration. Simply because machine learning research takes place on a computer does not automatically make it a subset of software engineering, nor does computing become a subset or artificial intelligence for more or less the same reasons. Research in machine learning happens in stages that must be organized in a manner that each stage takes from the one prior, does some processing, and passes its results to the next stage, like a conveyor belt.

Machine learning pipelines can be multi-disciplinary, with computing holding a limited responsibility to the overall flow. One thing that is universally the same is human input and control of the entire flow. And therefore, our proposed method fits into the human-in-the-loop (HITL) systems. Its advantages include mobility, low-power, and other attributes associated with embedded systems. From an engineering standpoint this research is more concerned about the relevance of software code used to solve a given deep learning task and the symbiosis or complementarity with host hardware.

Posit arithmetic also falls out of scope of this research simply because the energy requirements do not improve on IEEE 754-2008 floating point number arithmetic [5].

For deep learning algorithms to be effective, useful features must be extracted from large amounts of data. For this reason, there is a cost, and perhaps a well-founded argument, associated with the performance of the host system

used. Distillation is a mechanism used to reduce the resources needed by a model without hurting its performance. The same distillation can be applied to the data as well, by reducing its size or replacing some samples with synthetic ones. This research commends all these methods but considers them out of scope as well. Metrics like accuracy or precision are used with regards to digital arithmetic operations, or computations in general. Energy and latency are measured on the computational instructions carried by a host processing unit. Energy represents the computational effort (in terms of energy, measured in Joules) of performing a given task, while latency (delay) refers to the time it takes to complete a task.

One of the key theoretical insights of deep learning is that in recent years the general consensus agrees that increasing the depth of deep layers results in better model performance. The argument is so far proving to be correct to the detriment of host systems that cannot counter-argue for lack of capacity or compatibility with deep-layered models. The fundamental basis on which tensor calculus is implemented digitally remains open to experimentation. Automatic differentiation is also applied at the code level to take advantage of the benefits this approach may have on small form factors.

One of the main components in a deep-layered model is the fully connected layer, a collection of computation nodes each carrying the exact operation on a chunk of the input. The hurdle with convolutional neural networks, as the deep-layered model will be referred as henceforth, is that the computation of each node scales as the square of number of inputs. Very quickly, what looks like mere addition and multiplication can explode in complexity. Research has progressed towards the use of distributed computing as a means of allocating portions of the operations to other host systems to ease the complexity of the task, but also to arrive at the solution quicker. Given that host systems can have relatively expensive power and maintenance costs, it makes sense to consider ways to reduce or at most alleviate any cost associated with the distribution of tasks. The aim of this work is to contribute efforts on the use of low-powered devices for

deep learning research, that also includes the possibility of using distributed embedded devices to maintain the cost to relatively affordable levels for individual researchers as well as small teams that have financial constraints towards the realization of their endeavors. Lastly, another scope limiting factor is the use of a digital signal processing (DSP) accelerator for domain-specific computations, namely convolutions.

Research in general is an exercise of approximation, always edging closer to some optimal objective without precisely hitting it. The use of floating-point arithmetic in deep learning is both an enabler and a limiting factor for common tasks. On one hand, it enables precision depth that ensure optimal approximations, and on the other hand it requires resource-intensive computations that accrue the cost of conducting research very fast. AI accelerators are a stark example of this dilemma, precisely in the design of inexact arithmetic circuits. This paper uses 3 types of hardware accelerations to critically consider their singular contributions in deep learning computations. Each locks the user in a set of dependent libraries that are either open-source such as OpenCL, or proprietary (closed-source) such as NVIDIA's CUDA. Each has specific instructions on how to best utilize the maximum processing units for any task, or how to engage in parallel processing by ensuring all available processing units participate equitably.

In recent years, AI accelerators have grown considerably with ever demanding energy requirements considering that their design favors parallel processing, with each parallel unit adding to the energy requirements. Parallel computing is only beneficial in specialized conditions, most of which can also be handled by sequential (general) computing. However, each parallel annexation is potentially an increase in costs (financial and otherwise) that must be critically considered beforehand. This research experiments provide a pathway into lowering these costs by using parallel computing with cheaper alternatives. That is one of the objectives we set out to reach with the use of embedded systems.

1.5 Outline of the report

This paper is organized into 4 parts. The introduction gives an overview of the ecosystem and context necessary to this research. The research problem, justification, scope, and key assumptions are also presented in the first part.

The second part presents the literature for the research problem. Here we review similar works and their key contributions, paying attention to the best results.

The third part describes the methodology used. The tools necessary for our experiments are highlighted before the fourth and last part closes with our results, conclusion, and recommendations. In the conclusion section we go in depth to discuss our method execution showing how multi-faceted the proposal is. Knowledge and best practices are gathered from a wide range of disciplines, including cardiology, in attempting to present the results of this thesis.

CHAPTER 2

LITERATURE REVIEW

In recent years mobile computing has grown to become an essential component of life and work. With the recent attention around deep learning since 2012 and later the enormous investments in high-performance computing (HPC), deep learning and mobile computing have continued to attract each other in what continues to be a tug-of-war between deep learning at the edge or in the cloud. The mixing of mobile computing and deep learning have given a new definition to smart devices. Deep learning is poised to enable new frontiers of discoveries and innovations in both the mobile computing and HPC spaces.

To provide a guide for this thesis it is important to scope our research. First and foremost, we do not discuss any specific outcome from model training such as accuracy or errors. We are strictly limiting our explorations to computational performance. Secondly, although we introduced HPC this thesis focuses on low-powered computing, including acceleration.

Several algorithms are discussed for their prevalence in the deep learning field and used to demonstrate our proposal via experiments. This section introduces notions in computing and machine learning that are worth keeping in mind as we contribute further to the growth of deep learning for constrained computation.

Computing platforms and form-factors present advantages and inconveniences that simply do not favor one over the other, and when it comes to small devices the choices remain ambiguous. However, the vendors provide tools and libraries to further accelerate the commercial adoption in HPC as well as embedded computing, so it is no surprise that leading manufacturers have offers covering the entire range of platforms and form-factors, from cheap low-powered micro-controllers to geographically distributed super-computers.

By nature, computers have limited capacity. There is only so much they can do. However, the limitations are less and less of a concern nowadays. In the early days of modern computing, algorithms were programmed with the same procedural logic used in manual calculations. Floating numbers are arranged in large multi-dimensional matrices called tensors. Tensor operations are computationally expensive, and more so on embedded devices.

In digital signal processing, a multiply-accumulate operation (MAC) is an instruction execution step that computes the product of two scalars and adds the result to a summation, or accumulator. A single DSP core can handle up to 50,000 multiply-accumulate operations per second (MACS) due to its relatively low wattage. Typically, this is done fast for integers but not for floating point numbers.

The 2008 revision of IEEE-754 that improves digital floating-point arithmetic, or IEEE 754-2008, specifies a novel instruction, called FMA for fused multiply-add, that fixes the inherent properties of digital floating-point arithmetic of non-associativity and non-distributivity by a single rounding rather than two as is the case for common DSPs [6]. Certain operations with higher order polynomials could lead to errors unless careful considerations are made with regards to the memory space allotted to the result of the operations [7]. FMA is also great for the software implementation of arithmetic division and square root, but it is relatively inadequate for low-powered devices [8].

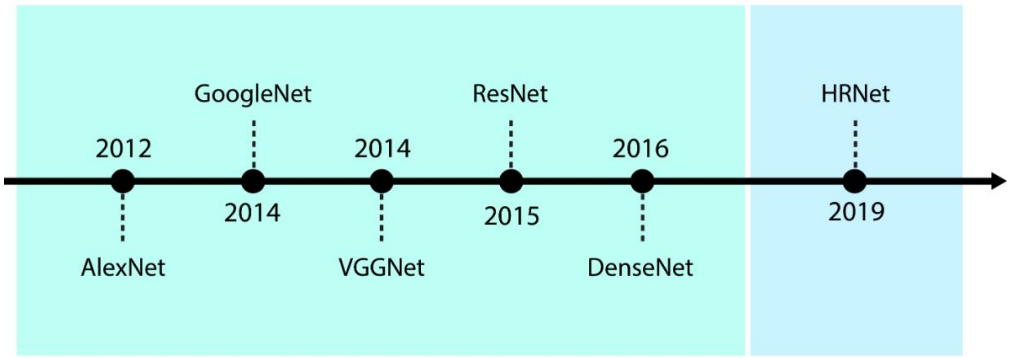


Figure 4: A timeline of other breakthroughs in deep learning image classification since the 2012 win of the AlexNet model, the blue zone on the right shows the beginning of the large-scale training era. Source: microsoft.com

2.1 Low-powered computing accelerators

In recent years, hardware accelerated neural networks have been proposed in lieu of their traditional CPU focused alternatives for improved processing speeds [9] dictated in large by breakthroughs as shown in Figure 4. Different types of existing hardware were repurposed, and new types of hardware were designed specifically to support acceleration for the most critical computations found on many neural networks [10]. GPUs are the most common and consumer-ready, plug-and-play type of accelerator [11]. Given a supporting architecture, and a pre-installed special purpose kernel library like the proprietary CUDA or the open-source OpenCL, a GPU can significantly improve computational speeds.

This is not exclusive to GPUs, however, as industry efforts continue to provide drivers for other types of processors like DSP. A notable innovation in high-performance computing is the TPU, a type of ASIC (application-specific integrated circuit) accelerator designed purposefully for neural networks on large data centers. The motivation for TPUs grew out of the need to address limitations on consumer devices like smartphones that could not process computations fast enough, therefore posting requests to a server and returning the results. For some of these special-purpose hardware, a dedicated matrix calculator is implemented at the hardware level instead of the common software level. TPUs, for instance, have a systolic array configuration to reduce delay.

Matrix multiplication continues to keep researchers busy to date. It helps a lot with tensor calculus. By defining a matrix with coordinates corresponding to rows and columns in a grid-like pattern, each element of the matrix can be found wherever a row and column intersect. It would take at least n^3 operations to multiply a n*n square matrix. Over the years researchers have attempted to optimize this task by rewriting the algorithm to computer specifications. In 1969

the best algorithm used $n^{2.8074}$ operations. In 2014 François Le Gall achieved $n^{2.3728639}$, and as of 2020 $n^{2.3728596}$ was achieved [12].

2.2 Execution steps in accelerated computing

Once a given deep learning task has been compiled and scheduled for execution it must be sent to the correct hardware accelerator that will carry said execution and return its results [13]. This is also the time to parallelize. Specific libraries help in this regard; like CUDA from NVIDIA, or OpenCL from the Khronos Group, an open-source consortium of industry and academia leaders, of which NVIDIA is a part of. The reason NVIDIA is mentioned is because of their monopoly in the deep learning space.

Vendors make different hardware accelerators, and OpenCL is meant to be compatible with as many as possible. A GPU may require a specific workflow between its components, which is different from how a CPU implements a similar workflow. A CPU may require vectorization to execute vector instructions but doesn't have components that are specialized for vectorization. A platform may have one or more of different types of accelerators, therefore achieving parallelism requires partitioning the task amongst a heterogeneous ecosystem to appropriately distribute the execution workload.

2.3 Floating versus fixed point numbers

Generating deep learning models with fixed-precision parameters as opposed to floating points has shown advantages [14] in terms of accuracy [15]. Fig. 1 shows why digital floating-point arithmetic is non-associative and non-distributive. Floating point precision has long been a standard in computers [16], but the popularization and consumer adoption of deep learning applications has pushed scientists and engineers to rethink this standard because of major concerns

on embedded systems [17]. Floating point scalars consume more energy compared to fixed point scalars.

Fixed-point arithmetic is a more efficient alternative, but the programmer must manually control and accurately determine the range and flexibility of the variables as shown in Table 1. Digital signal processing falls between two computationally distinct categories:

- Fixed point
- Floating point

These categories refer to the format used to persist and manipulate numeric data. Integers are represented with 16 bits and floats, as they are commonly called, with 32 bits. This way, floating point is fit for scientific purposes for its wider range, from very large numbers to extremely small ones. When it comes to operations that can quickly grow in complexity like exponentiation, a dynamic format like floating point is, again, preferred over fixed point.

In every DSP execution a quantization step is applied which yields errors. This error is a gap between the actual manual computed result and the digital result. Quantization here is a process of rounding and or truncating a result to the nearest value that satisfies a given format [18]. In fixed point format the gap is noticeably bigger, hence floating-point format is, once again, preferred.

Without disregarding the many great benefits of fixed-point and floating-point DSP alike, generally the latter is meant for computationally intensive applications where precision is paramount. The former works great for general purpose, high-volume applications. However, in the next section we reveal ways to manually manage the gaps derived from fixed-point operations in DSP, and even include differential steps in the execution.

2.4 Q-format number notation

The Q format is a proprietary, logical representation, or notation, of fractional bits and optional integer, non-fractional bits, that is fixed in advance; in other words, with Q-format floating numbers are treated like integers. This notation is intended for use in programming implementations of routines that manipulate numbers on hardware that does not support floating precision. Under strict precautions, Q format numbers can be used in automatic differentiation, although the memory capacity of the resulting gradient must be known in advance to avoid memory overflow errors.

Q-format is an abstraction that allows to represent rational numbers that behave like integers by pre-defining the space required for the fixed part and the space required for the fractional part. Using this abstraction, it becomes easy to bring together the benefits of floating-point format into the fixed-point format and maintain a reasonably low rounding gap error. One of the caveats is to consider saturations, or scenarios that may lead to an output of a higher format range than the inputs.

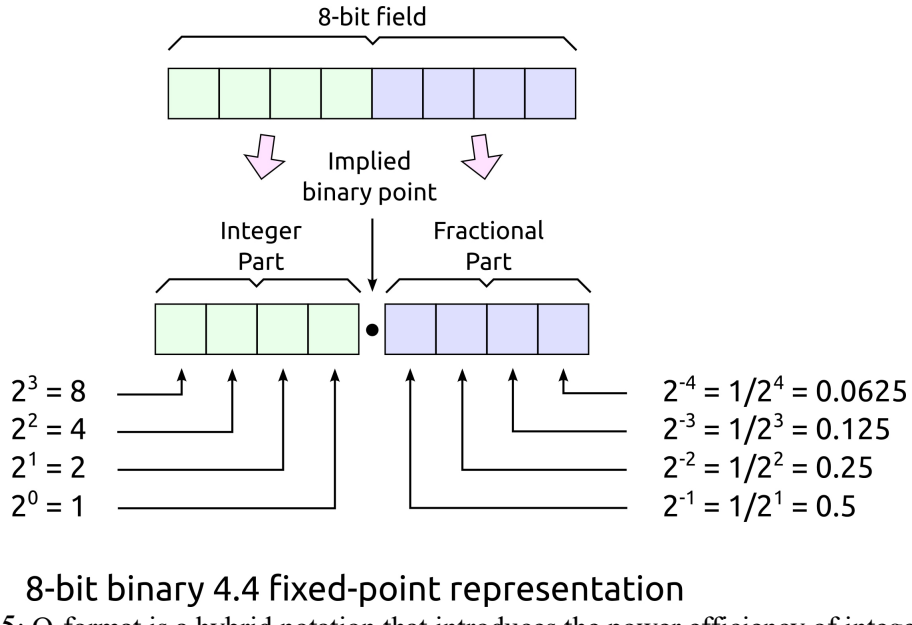


Figure 5: Q-format is a hybrid notation that introduces the power efficiency of integers while maintaining the dynamic range of floating-points

Consider A=64 and B=64, 2 scalars of Q-format type with a fixed part of size 8 and a fractional part of size 24, noted Q8.24. This is a signed format that can hold numbers up to 127 (one of the eight fixed part bits is used to hold the sign). The addition of the values yields 128, a number that this format cannot support. This saturation can be handled by either forcing the result to 127, therefore creating a gap of 1, or by choosing a larger Q-format type.

2.5 Convolution operations in deep learning models

Multiply-accumulate operations are among the most common in machine learning computation. Table 2 shows a few algorithms along with the energy and speed requirements for each based on MAC units. The basic idea is to utilize the hardware capacity to its maximum by leveraging algorithms. Other consideration including I/O bus speeds, pipelines, off-chip memory access or on-chip buffering, whichever composition of methods is best to maximally execute any given task.

Table 2: Multiply accumulate readings for some algorithms evaluated on a DSP accelerated embedded ARM architecture. Energy savings (picojoules per MAC), Latency savings (MAC per second), and % Gains (comparison to IEEE754-2008)

Algorithm	Energy savings (pJ/MAC)	Latency savings (GMAC/sec)	% Gains
ABS (Absolute)	1.33	3	11
FFT	1.9	1.8	24
BatchNorm	1.67	2	27
2D Conv	1.9	2	21
Softmax	1.2	2	16

2D convolution is among the most repeated tasks in deep learning [19]. It is implemented in 2 major ways: the sliding-window technique or a large matrix multiplication. The former has come to be known simply as 2D convolution and is focused around a smaller 2-dimensional matrix "sliding" over the larger matrix, such as the pixel values of an image. Each step is a Hadamard product between the small matrix, called a kernel filter, and the corresponding portion of the large matrix. The second method converts the input map into a matrix and simply

multiplies it by the kernel. Either method presents advantages and setbacks that fall outside the scope of this work, but we experiment on differentiation applied to the following algorithms:

1) Fast Fourier Transform (FFT): research showed that a 2D convolution solution is the same as the Hadamard product of two FFT transformations followed by an inverse FFT [20]. This discovery helped reduce the complexity from $O(n^2k^2)$ to $O\ (n^2logn)\ +O(n^2)$ when assuming a square matrix. The approach is not optimal for small kernel sizes, which most convolutional neural networks use [21].

2) Winograd Minimal Filtering was proposed to reduce the complexity of convolutions with 3x3 kernels by slicing 4x4 tiles out of the input image [22]. Although this method is limited to a 3x3 size it is in response to a growing trend in the scientific community. See Figure 5 for details.

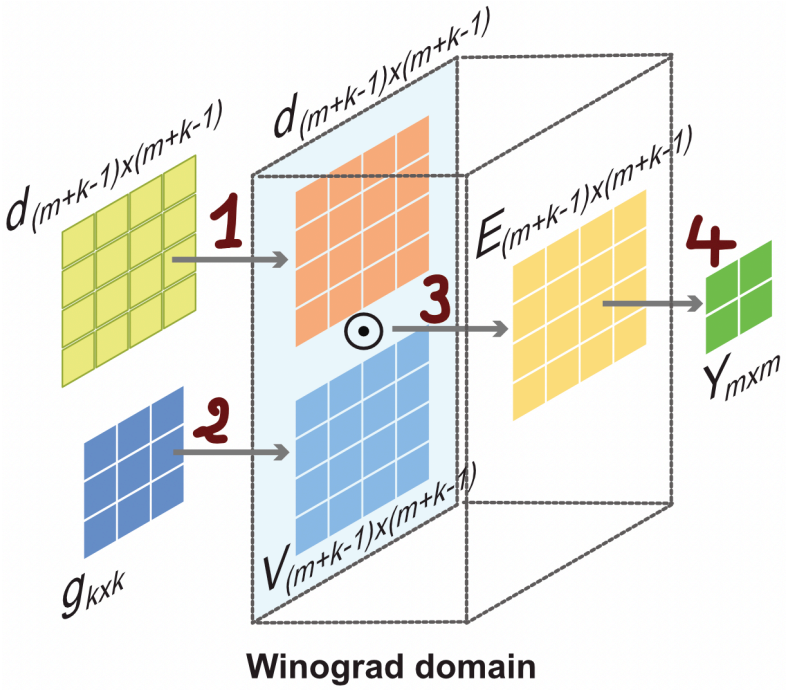


Figure 6: Depiction of a Winograd convolution showing 4 matrix-to-matrix operations. All operations are multiplications except (3) which is element-wise multiplication. It means all except (3) can be computed by add/shifting operations such that the total number of operations is equal to the number of element-wise multiplications. This total is significantly less than conventional convolution operations.

Models used in deep learning are highly structured with particularly few control dependencies, sometimes borrowing from other structured concepts such as graph theory. FFT consists of multiply-accumulate operations that are easy to track in partial or complete executions [23][24]. This, in turn, enable elaborate synergies between hardware and software in ways that could greatly benefit embedded systems. FFT is also highly differentiable.

2.6 Automatic differentiation

Automatic differentiation (AD) is a technique for computers to automatically return certain values for tracking an execution in the same manner that differentiation in mathematic can let us track the rate of change during a set of calculations. AD evaluates a solution in a non-symbolic manner to avoid infinite differences by providing a trace that can be fully pieced together using the chain rule. It is powerful enough to trace even control flows, on top of the easier closed-form computations.

Traces can be generated by logically finding the derivatives of a function with respect to some given variable set, one variable at a time. The forward mode AD is one algorithm that computes the primal and tangent traces, though we are interested in the former only. These intermediate variables are the pieces needed to constitute a trace of execution, which allows for “time-stamp” computation and provides a powerful means of arbitrarily or randomly walking back or resuming at whatever period of execution. The adjoint mode AD, commonly called reverse mode generates a trace by starting from the end of an execution path, which means that it is useful only when gradients are provided, most likely from a forward mode operation.

Automatic differentiation is increasingly proposed as the default method of computing derivatives in deep learning [25]. Differentiation is a mathematical algorithm (see Algorithm 1) used in tensor calculus since most problems in deep

learning are expressed in terms multi-dimensional matrix multiplications and summations [26]. Software programming frameworks like TensorFlow [27], and PyTorch [28] are efficient in part due to their automatic computation of gradients. Scalar summation and multiplication are the basis of convolution operations in deep learning. While digital integer is associative, that is notoriously not the case with floating point, causing problems with parallelization and reproducibility. There exist solutions that unfortunately are not implemented on modern computers.

One of the main contributions of this work is the use of differentiation [25] on the Q-format notation. Automatic differentiation is the computation of gradients during execution without explicit request from the user [29]. During model definition, the user typically defines the forward path of execution, and the automatic differentiation generates a backward path.

2.7 Multiply-accumulate (MAC)

The concept of multiply-accumulate is closely associated with digital signal processors, and a major hardware concept in convolutional neural networks. In fact, the same principles are found in systolic arrays because the elements of said array, formally called processing elements (PEs) are multiply-accumulate units. Systolic arrays are hard-wired nodes of computation that are optimized for a unique operation that work to achieve a bigger operation. Each node receives data upstream, performs the operation, saves the result then pass the same result downstream. This sort of collaborative arrangement can be encoded at the transistor level, but it is also abstracted at the software level using existing mathematical rules such as the fast Fourier transform or the Winograd convolution, each offering its advantages and inconveniences based on the problem definition, the available computing architecture, and the requirements of

the users that may include environmental parameters such as temperature, electrical power, etc.

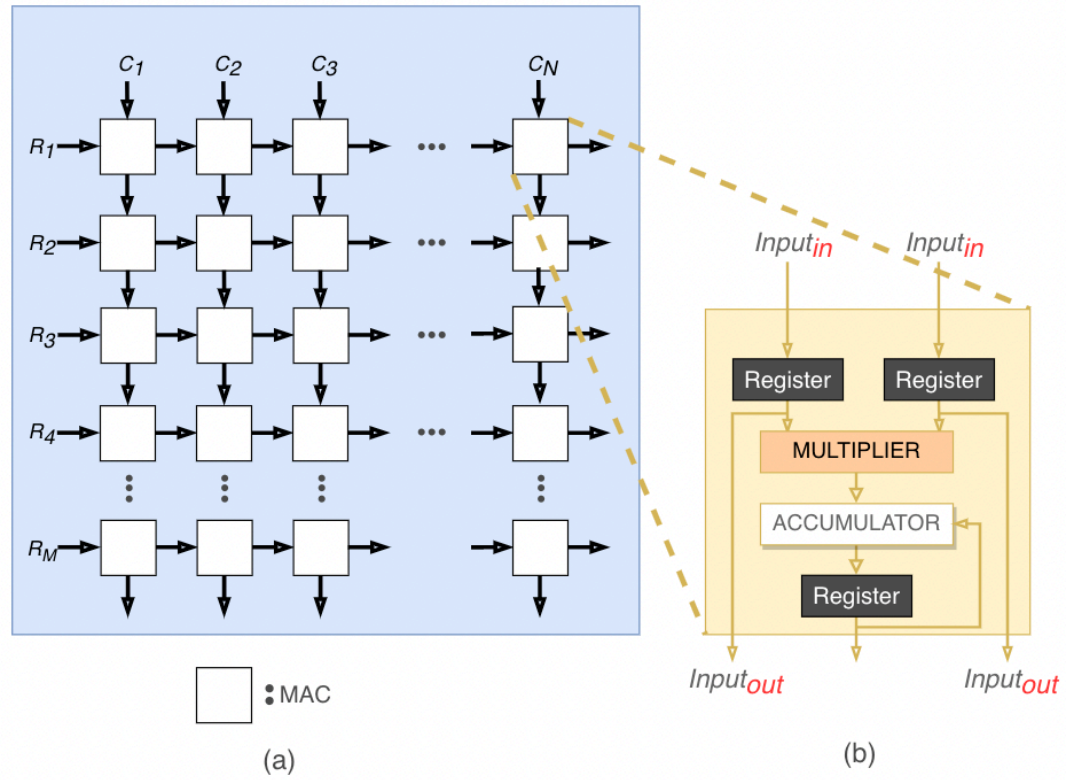


Figure 7: Logical organization of transistors in an accelerator or CPU that shows how instructions are carried by multiply-accumulate units. This logical representation (a) is very similar to a 2-dimentional array structure, which they are primarily designed to manipulate. Each MAC is a set of transistors and capacitors (b) that provide gates and registers respectively to multiply 2 inputs and add the result to registry before passing all values, including the input, downstream to the next MAC.

MAC, as shown in Figure 6 (b), is inspired by a mathematical rule used to evaluate polynomial functions, also called the Lagrange method or Horner's rule, and has been used since the early days of computing as a method to solve said functions efficiently. As a reminder, this rule essentially reorganizes the function into a series of additions and multiplications. But because of the recursive nature of the rule, the number of additions and multiplications can grow exponentially and present a bottleneck in processing. The nodes of transistors in a MAC unit are meant to split these recursive steps amongst themselves and evaluate the function simultaneously in a tree-like structure, like a systole that expertly pumps a load of blood equally in all arteries during a cardiac cycle. Systolic arrays,

which we discuss in this thesis were named and inspired by this ingenious feat of nature.

Given a degree-n polynomial, and by contrast to the monomial form or rule which requires at most n additions and $(n^2 + n) / 2$ multiplications, the Lagrange method reduces its steps to n additions and n multiplications.

Parallel processing has lots of approaches, but we devote the next paragraphs to diving deeper into systolic processing. A systolic array is a network of processing elements (PEs) that rhythmically compute and pass data during a given cycle of execution. The name is an analogy to how blood flows through arteries to capillaries in one heart contraction and how it is sucked out of the capillaries and into the veins when the heart expands. The contraction and expansion are also called systole and diastole respectively. In the same manner, data flows into the PEs in a rhythmic way before it is returned to memory.

In a systolic array, there are many identical processors arranged in a clear linear or 2-dimensional structure. Each PE is connected to others and has its own private storage, or registry. This structure can be manufactured physically on hardware, or virtually as written or compiled computer code. This research uses computer code to design a systolic array as a 2D structure of PEs that are mapped to physical MAC units on the target processor.

Regularity, reconfigurability, and scalability are some of the features of systolic design. The objective is to maximize throughput capacity. Multi-dimensional image processing, video streaming, non-linear optimization, convolution operations or decision-based algorithms are some of the many that are computationally demanding and therefore can benefit from systolic design.

Hardware and software must go hand-in-hand to remove the memory hierarchy bottleneck and improve performance. Using a dependence graph as a mapping between software and hardware makes it accessible to achieve a space-time transformation. Mapping is

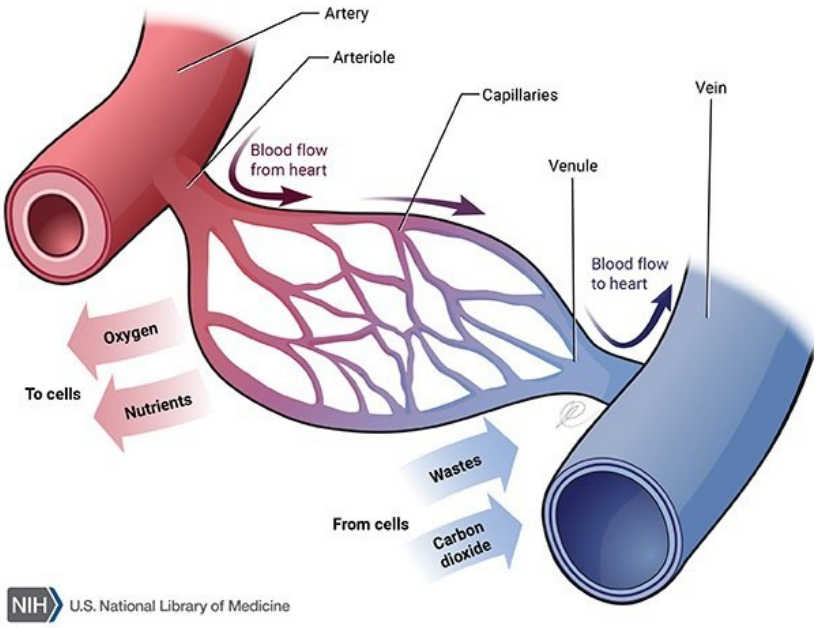


Figure 8: Capillary network showing the systole and diastole. The design is called systolic because data flows through the chip in waves, like the way the heart pumps blood

Systolic design methodology maps N-dimensional dependence graph to a lower dimensional architecture using transformations. Matrix multiplication can be considered a systolic structure.

In terms of performance research has shown that systolic design can boost the performance-per-watt capacity of its host, depending on the instruction set the system host is based on. The topic of instruction set architecture falls outside the scope of this thesis, but the reader is encouraged to explore more on their own. However, typical RISC processors rely on scalar processing to perform simple calculations such as addition or multiplication. At high clock speeds this should not be an issue, but for matrix operations a better option is needed. One such option is called vector processing and is implemented on modern CPUs as an extension to the instruction set such as SSE or AVX, and on modern GPUs as streaming multiprocessors (SMs). Vector processing makes it possible to execute multiple operations in one cycle. It means that intermediate results are retained on the processor until the matrices are traversed in their entirety, which significantly reduces power demand.

Target host resources can be a complex endeavor which requires complete mastery of the hardware instruction set architecture as well as other companion specifications and software libraries used to access these resources. As an example, let's consider GPUs processing with its proprietary CUDA library for manipulating the accelerators. When writing algorithms for operations that we want to execute on such processors, we must use the proposed way CUDA implements them. Like simple addition or multiplication, the library has a big library and even bigger documentation on the way to target their hardware. For general-purpose processor such as CPU, instruction set extensions such as SSE or AVX present standard techniques to implement vector processing, but this is limited because the architecture of a CPU is primarily sequential in nature.

CHAPTER 3

PROPOSED METHOD

To increase computing resources savings and reduce latency, we propose to use the Q-format notation. The Fast Fourier Transform and Winograd minimal filtering algorithms are evaluated on this notation. Human-in-the-loop systems built on the assistive intelligence concept are conditioned on fast processing and portability, the former being the missing feature for embedded systems. Our contribution provides a proof of concept for deep learning tasks on embedded systems by applying automatic differentiation (Figure 7) on the Q-format notation.

Algorithm 1 Automatic Differentiation

$(q_1, q_2 \ldots, q_m) \leftarrow$Input in Q-format
$d_i \leftarrow (d_{ij})_{j=1}^{|previous(i)|} \in D_i(q_{previous(i)})$
$i \leftarrow p+1 \ldots q$
$e \leftarrow$ Epsilon (host approximate)
procedure (FORWARD MODE)
  Initialize $\frac{\delta q_k}{\delta q} = \mathbf{e}_k, k = 1, \ldots, p$
  for $k = p+1, \ldots, m$ **do**
    $\frac{\delta q_k}{\delta q} \leftarrow \sum_{j \in previous(k)} \frac{\delta q_j}{\delta q} d_{kj}$
    where $q = (q_1, \ldots, q_p)$
  end for
  return $\frac{\delta q_m}{\delta q_{1,\ldots,m}}$
end procedure
procedure (BACKWARD MODE)
  map$\{previous(t)\} \leftarrow t \in (1, \ldots, q)$
  Initialize $v \leftarrow (0, 0, \ldots, 0, 1) \in \mathbb{R}^q$
  for $t = q, \ldots, p+1$ **do**
    for $j \in previous(t)$ **do**
      $v[j] \coloneqq v[j] + v[t] d_{tj}$
    end for
  end for
  return $(v[1], v[2], \ldots, v[p])$
end procedure

Figure 9: Proposed implementation of automatic differential processing

3.1 Assistive intelligence

The precipitated rise of deep learning in recent years has shifted the norms by which data is processed, modeled, and interpreted. In the vastly complex of decision making and business intelligence such as trade, finance, government, etc. deep learning is used to recognize patterns in spoken or written language, objects, people, buildings, geo-spatial scans, and many more. Deep learning is now actively contributing real change in control systems with such undertakings as autonomous vehicles, unmanned drones and ships, intelligent farming and more. There are so many aspects of living and doing business that could be presented, or even core systems issues such as privacy, encryption, or data compression, energy/resource planning and utilization, but that would fail to objectively discuss what is scoped under this thesis.

Applications such as those highlighted above are very close to their intended users. The growth accumulated in the number of use-cases has followed closely the continued climb in mobile technology or mobile computing to the point where analysts firmly predict deep learning will redefine smart devices of the future. It would be a mistake to ignore the invasive growth of deep learning without noticing its insatiable need for more power and speed, two core requirements that defy the constrained nature of mobile computing.

HPC has skipped over the fence predicted by Moore's law by promoting parallel computing using accelerators or completely rethinking the hardware components that best match the needs of deep learning researchers. Domain-specific hardware such as Google's TPUs have contributed to the ongoing perception that deep learning is an expensive undertaking requiring stellar investments that common mobile computing users can only consume, but not contribute to.

With the possibility offered by these expensive accelerators, new horizons of possibilities have come to light where models and data growth has grown in terms of storage capacity in ways that essentially leave a common individual unable to

build or apply deep learning to their own situation, whatever that may be. But since we're talking about building systems that must be consumed, in turns out that the consumers have strongly chosen to consume on smaller and smaller devices. This therefore has made it hard to ignore intelligence on small form factors. Across multiple domains in hardware, software and learning techniques there are proposals that benefit resource constrained computations.

This thesis is among the efforts promoting assistive intelligence away from high-performance computing and close to end-users to establish a platform for training and inference. The term "assistive" is chosen to signify the augmenting capability that AI offers to its non-artificial counterpart. Under the human in the loop (HITL) concept this term further consolidates the dependence on a human's constant oversight and presence at all steps of training and inference for deep learning to truly be consumed, as opposed to the large industry investments that relegate deep learning to those who have enough capital.

The role of training is to generate parameters based on available, a process currently assumed to happen outside of a final destination, notably referred to as inference device. What is also assumed is that all the data needed, called ground-truth must be available before the sequential activities generally understood by "training a network" are engaged. Any new data made available after training has been engaged is put on hold to serve in inferring or training anew. A convolution is a process formulation that essentially extracts representations from input data and increase the complexity of the representation further down the layers of the learning network until input data is abstracted enough to accurately estimate or recognize signals or patterns.

There are multiple ways to observe intelligent abilities, whether shown by a human or not. The imitation game, for instance, used language to design observable tasks that, once executed, could lead to a measure of intelligence by using a human as benchmark. Understandably, using natural language ability alone was a limiting proposal.

Over the years more tests have been designed and implemented by including other abilities, such as speech understanding, speech, object recognition and more. Subject-specific benchmarks were also devised to monitor advances in technology from multiple fronts.

Embedded systems offer the least requirements to host assistive intelligence, but heavy reliance on floating point precision makes it difficult to counter the energy requirements. Human-computer interaction scenarios should flow quickly and seamlessly. Therefore, embedded systems should be able to support the requirements of assistive intelligence.

The experiments we conducted provide results towards the realization of assistive intelligence as a means of augmenting or assisting natural intelligence with artificial intelligence.

3.2 Accelerated computing in embedded systems

Using a kernel library such as OpenCL means critical computation that has been marked for acceleration is compiled to a lower machine instruction code. Memory management may decide which resources to use, either the ones closest to the processor unit or even larger ones availed by the platform itself, such as RAM or hard disks. Luckily this complex undertaking is easy to recognize as a pattern of control and data flow. Memory management can be done manually by the programmer or automated by a caching mechanism that prioritizes reusability and handles overflows. It becomes a matter of balancing, even when units perform in a heterogeneous way. Kernel libraries are responsible for tiling or splitting tensors in smaller tensors and then passing them to different caching orders at the same time while auto evaluating the tiling process itself.

To make optimal use of caching, it is better to concentrate the bulk of operations on one or a few tiles to avoid bottlenecks on the I/O routes. Execution may include repetitions such as loops that cascade into other loops. In this case

multiple inner-loop executions for multiple tiles can be serialized to their corresponding outer loops in a process called fusion. Generally, caching is implemented by a technique referred to as SIMD, or single instruction multiple data, and is part of the instruction set architecture (ISA) of a computing device.

Parallel computing addresses some processing bottlenecks by spreading the instructions over multiple simultaneous processors, or accelerators, to both speed up computation and freeing the core processor, or CPU, to focus on orchestration.

Neural networks rely on parallel computing [30], constituting a major setback for embedded systems whose resources and energy consumption discourage parallelism. However, as we will demonstrate in this report, there are domain-specific accelerators that are tuned for convolution operations and consume little energy.

There is a strong correlation between progress registered in machine learning, deep learning, and progress in parallel computation [31]. Deep learning applications, in particular tasks like computer vision, machine translation, speech recognition, etc. have shown that more computation leads to more energy consumption. Operating costs have forced deployments to a client-server approach because low-powered devices cannot bear the energy consumption demands of deep learning applications. There are increasingly more energy-limiting options because industry trends lean towards privacy-preserving options.

3.3 Representing and manipulating numbers

High-performance computing (HPC) remains the de facto environment for many deep learning projects. Ongoing efforts are required to contextualize machine learning to edge devices. In the early days of modern computing, the energy requirements far exceeded what is experienced today. Part of the efforts

is therefore to continue achieving high results with minimal consumption. This trend is gaining more traction in machine learning and deep learning.

Let's say there is an unsigned fixed-point integer with a scaling of 0.01. Based on this assumption, the calculation 0.01 + 0.01 will naturally result in exact 0.02. Assigning the same calculation using floating-point, 0.01 + 0.01 will result in 0.019999999552965164184570312 5. It's close to 0.02 but not the exact value. If we have a look at the value 0.01, we can see that it is neither represented in an exact way. This becomes clear if we look at how the value 0.01 is really represented in floating-point. A floating-point value is made up of 3 elements: 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa totaling 32 bits. To represent a certain number, the mantissa represents a certain value with the exponent.

Computer scientists are expected to understand the binary representation of numbers, even if most are not comfortable with its use daily. Binary integers are easy enough: the digits, going from right to left, represent 20, 21, 22, 23 and so on. So, the number 10011 is 1 + 2 + 16 = 19.

It is less common to see floating point numbers represented in binary, but just as useful to understand how they work. The left of the binary point, or the whole part of the number, is represented like a binary integer. That is obvious. The more confusing part is to the right of the binary point, or the fractional part of the number. Here the digits represent 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4} (1 /2, 1 /4, 1 /8, 1 /16) and so on. So, the number 0.11011 is 0.5 + 0.25 + 0.0625 + 0.03125 = 0.84375

The IEEE-754 along with its 2008 revision are engineering standards that specify how digital floating numbers arithmetic is implemented on software and hardware. Every iteration of the standard is meant to keep up with the current state-of-the-art in hardware design while pursuing efficiency and stability. These objectives have led researchers to investigate a finer control of exponent and mantissa bits to strike the right balance of accuracy and energy consumed. For example, numerical computations often iterate over multiple solvers where the

residual error is a function of the input itself. In such cases the only way forward is to manually commit conversions and multi-versioning, which is a threat to code objectivity. Programs could potentially get off track with the intent of the software they are working on if they manually handle too many of these residual errors. Modern compilers provide many safeguards that can help mitigate residual errors to stay within acceptable ranges of accuracy and power consumption. Compilers play a crucial role that leverages efficient use of floating-point numbers by targeting representations supported by the host system.

To maintain good results when manipulating digital numbers, programmers must design the basic arithmetic operations such as addition, subtraction, and multiplication, to handle the values defined in the notation on the host system. This requires considerations of the host architecture with details such as parallelism, large multipliers, leading zeros, counters, etc. Although they continue to be challenging engineering problems, little consideration was given to the importance of energy consumption and latency, especially for embedded systems.

With the high availability of data and the rise of deep learning [32], power and speed are exacerbated with every new development in the field. Figure 9 shows a proposed method to convert a floating-point number to Q-format notation.

Algorithm 2 Q-format Conversion

```
l ←Bit scheme (short, single, or double)
e ←Lower bit scheme
p ←Higher bit scheme
m ←Mantissa bit size
x ←Exponent bit size
a ←Exponent value
b ←Mantissa value
s ←Sign value
o ←Output
procedure (ALTER)
    while l < p do
        l = l << 2
        m++
    end while
    while l > e do
        l = l >> 2
        x++
    end while
    a = l + (1 << m)
    b = l − (1 << x)
    for i = 1, . . . , l do
        if b < 0 then
            a = a + (b << i)
            b = b + (a << i)
        else b > 0
            a = a − b(b << i)
            b = b − a(a << i)
        end if
    end for
    o = b × s
    o << (x − m − 1)
    return o
end procedure
```

Figure 10: Proposed conversion from IEEE754-2008 to Q-32 notation

Tensor representations on computer systems have evolved with the remarkable achievements in computer vision tasks [31]. Tensors are represented as large multi-dimensional arrays of floating numbers. Although the manipulations do not go beyond mere summations and multiplications, the tensor itself as a data structure takes millions of steps to be fully traversed, often using multiple levels of control loops, and other repetitive instructions. Floating

numbers are suited for this type of computation at the expenses high energy and latency costs, which embedded systems cannot overcome easily.

As shown in Algorithm 2, we propose a 32-bit width with user defined rounding mechanisms. The algorithm shows how a single precision floating number can be converted to the proposed Q-format notation in Section 4.

Efficiency in deep learning, especially for embedded systems, is measured according to:

° The energy consumption of the architecture for any evaluated algorithm, model, or computation [33]. Here we detail the specifications of the process that we subject to computational analysis, although we return to the convolution since it is the most used. The number of trips from processor to memory should also be accounted for.

° The latency corresponds mostly to memory bandwidth [34]. This metric should be found in the actual run time for various executions by comparing timestamps at different stages of execution.

° The area cost of the physical processor or chip. It is the core execution area spent by the instructions.

3.4 Hardware equipment used in experiments

We compare our configuration with different other modes available, including accelerators like the Intel Movidius2 Neural Stick. Popular frameworks like PyTorch and TensorFlow already provide most of the proposed features like automatic differentiation.

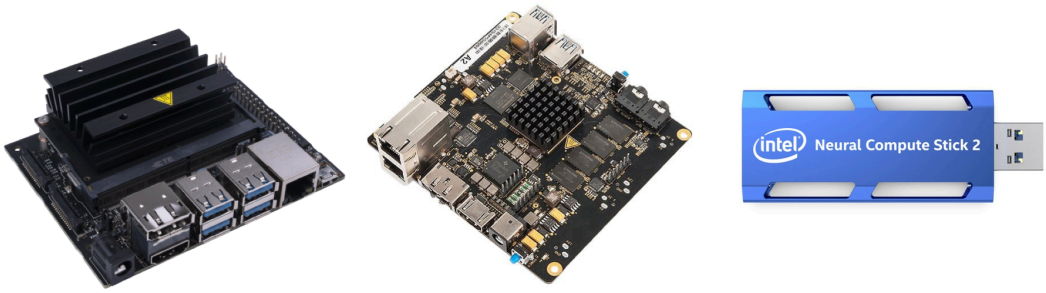
Our implementations focus on embedded systems (see Fig.3) by applying automatic differentiation to a proprietary number format that is more adapted to low-powered, low-latency environments.

Convolution has a periodic characteristic to that makes it particularly suitable for digital signal processing. Convolution operations take a lot of time

and therefore parallelism in hardware accelerators should be harnessed to shorten the time. Considering a typical convolutional neural network, say with 15 layers consisting of 9 convolutions and 6 pooling layers; It would take as less as half the time for the Winograd algorithm to process all the layers through a digital signal processor with parallelism as it would if only targeting a CPU with sequential computing. DSP can only support a limited range of MACs [35] but we feature this type of accelerator in our experiments because it requires little power. Floating point format is supported in most DSP accelerators at the cost of increased power consumption. Integers are known to consume less energy compared to floating numbers.

Consumer devices have features that address some of the challenges raised in this report, and therefore provide a benchmark on which to assess our implementations. Expressions are evaluated on these pre-packaged devices using Numpy and Python with hardware acceleration abstracted by the open-source OpenCL kernel library that is supported in all 3 devices shown in Figure 9. Some devices allow for limited configurability beyond a certain point for reasons that fall outside the scope of this work. We take advantage of the differences in devices to show how each supports our proposed methods.

We compared automatic differentiation for first-order derivatives on our implementations against open-source alternatives mentioned in Section 2.5. We also focused on scalar-valued functions that allowed the use of fixed-precision format.



NVIDIA Jetson Nano B01 TI AM572x C66x Intel NCS2

Figure 11: Three embedded systems used for our experiments. More and more vendors continue to show interest when it comes to deep learning on small form factors

Thanks to programmable hardware technology such as FPGA it is possible to envision and implement a DSP accelerator, adapt, and apply Winograd and FFT algorithms to it then analyze how values are manipulated by the custom system. Such flexible and speedy experimentations allow for better benchmarks to be adopted. For example, a convolutional neural network such as YOLOv2 can be improved, first using a power efficient value notation such as Q-format, next by targeting a low-powered accelerator like the DSP, then by using the Winograd algorithm to reduce by half the number of multiplication operations, resulting in reduced processing times. A leading and often expensive accelerator can process a large image in 16 milli-seconds, but this low-powered version can set a new benchmark, lowering the time it takes to process the same image to 10 milli-seconds.

CHAPTER 4

EXPERIMENTS AND RESULTS

For our experiments we designed a 32-bit implementation of the Q-format notation using a modified open-source code base, and we also used the OpenBLAS scientific calculations library as a benchmark for latency measurements.

The structure of a neural network in deep learning is comprised of layers in a graph-like execution process. Each layer has nodes where certain mathematical operations take place, the result of which are passed on to the nodes in the following layer. A loss function is used to supervise this entire mechanism by applying differentiation to generate parameters, or weights, that best estimate a solution for the problem being solved.

Layers, nodes, and parameters are essentially matrices, and the operations taking place are summations and multiplications. There are so many matrices multiply operations that it serves to measure the complexity of a neural network by simply counting the number of matrix multiplications. Technically the result of the multiplication is added to an accumulator, therefore a MAC, or multiply-accumulate, is a unit of operation. MAC is used to measure the efficiency, expressed in pico-Joules per MAC (pJ/MAC), and throughput, expressed in Giga-MAC per second (GMAC/sec), of our modified algorithms. We also measure the same for IEEE 754-2008.

In our experiments we design tensors and implement the tensor manipulation algorithms discussed in Section 2.5. These algorithms are used to write functions common in deep learning. Since many libraries have different implementations of the same functions, we benchmarked our results against the open-source OpenBLAS commonly associated with tensor manipulations. The functions are: (i) 2D Convolution, (ii) Element-wise addition, (iii) absolute, and

(iv) product, (v) Softmax, (vi) Batch normalization, (vii) ReLU and (viii) the fast-Fourier transform (FFT).

We are interested to find out how efficient and performant our algorithms are when subjected to hardware acceleration on 3 different architecture types. DSP acceleration requires the least energy at the expense of limited applications.

4.1 CUDA acceleration

Writing kernels or assembly instructions is a recommended method for targeting hardware acceleration on NVIDIA graphic accelerators. Using the proprietary CUDA language with its custom compiler, we applied the Q-32 notation on several algorithms.

The results shown in Figures 10 and 11 illustrate how the same algorithms perform using conventional 32-bit floating numbers versus Q-format notation. Auto-differentiation has no increased effects as expected.

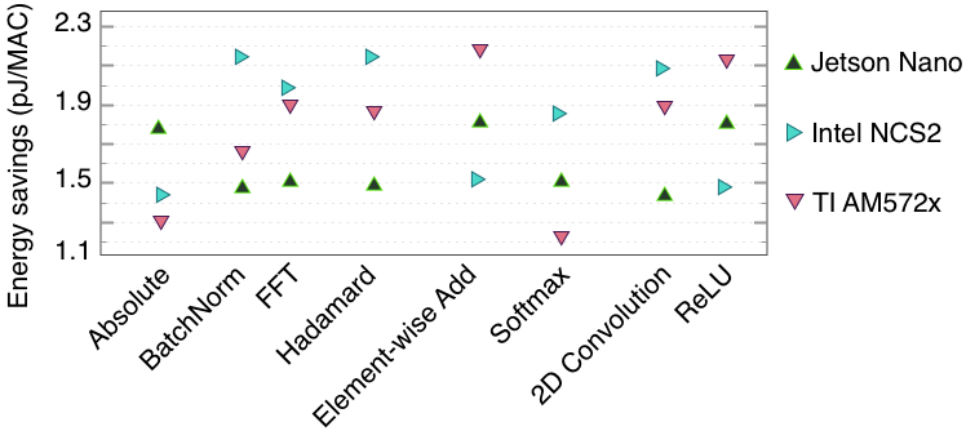


Figure 12: Energy gains (picojoules per MAC operation) in comparison to IEEE-754 floating point for similar tasks on 3 different architectures

4.2 MKL acceleration

Intel hardware acceleration takes place in the central processing unit, giving it a notable difference over the other options we observe in our

experiments. Given this particularity, we used proprietary libraries to better take advantage of this architecture. Figure 12 shows more details.

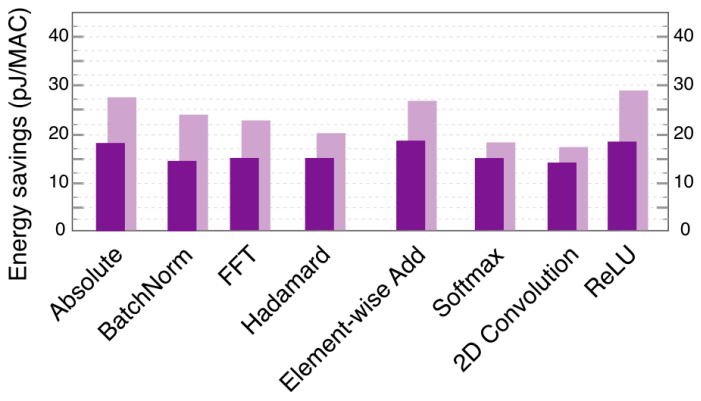


Figure 13: Q-32 notation (dark purple) shows performance improvements over its IEEE754-2008 alternative (light purple) on an NVIDIA Jetson Nano

4.3 OpenCL acceleration

DSP acceleration is limited to signal processing and other related functions. Incidentally, convolutions are particularly suited for this type of acceleration. Using the open-source OpenCL kernel library we can target computations that best utilize this architecture. Of all the devices used for experimentation, DSP is the least power-consuming platform and therefore presents an obvious advantage.

Our proposed method could only be fully implemented with the open-source OpenCL library. Although supported on all 3 devices presented, only 1 features a DSP accelerator that best improve the symbiotic performance between hardware and software. Refer to Figure 12 and more details in Table 3. The percentage gains compared our method to their floating-point alternative.

Table 3: FPBench aggregate precision error comparative results on AM572x with C66x DSP (with arbitrary rounding modes)

Algorithm	Energy savings (pJ/MAC)	Latency savings (GMAC/sec)	Energy Gains (%)	Latency gains (%)
Absolute	1.33	3	11	14
FFT	1.9	1.8	24	28
BatchNorm	1.67	2	27	25
2D Conv	1.9	2	21	17
Softmax	1.2	2	16	16

Abbreviations: ABS (Absolute), FFT (fast Fourier transform), BatchNorm (batch normalization), 2D Conv (2-dimensional convolution, Softmax (Softmax)

4.4 Advantages of using DSP hardware accelerator

As explained in section 3 OpenBLAS is an open-source library for scientific computations and it contains several algorithms for linear algebra like matrix multiplications. The right-most column in table 2 shows how algorithms adapted to our proposed method compare to their OpenBLAS counterpart in terms of energy gains. This library makes extensive use of multiply-accumulate operations, giving us an ideal environment for evaluation.

Figure 14: CPU acceleration for the same Q-32 notation (dark purple) still shows significant gains over IEEE754-2008 (light purple) on Intel NCS2.

We apply our evaluations on 3 devices with the use of Q-format notation for auto-differentiation. For similar tasks, we notice that domain-specific tasks that target DSP acceleration have a clear opportunity to spend less resources without sacrificing performance. Q-format introduces the missing feature for low-powered intelligent assistants unless IEEE 754-2008 addresses the design gaps highlighted in previous sections. But as Table 3 shows, Q-format aggregate precision error (or adjustment error) is slightly reduced from IEEE754-2008 regardless of host architecture. Rounding-up or truncating is a common operation in digital arithmetic. FPBench is a benchmark used to evaluate precision errors in floating-point digital arithmetic.

We compare the algorithms on throughput and latency. We also show the quality gains on replacing floating numbers with Q-format notations. To check

for errors, we manually re-write some arbitrary algorithms to handle our proposed notation, which means we define the operands logic for add and multiply.

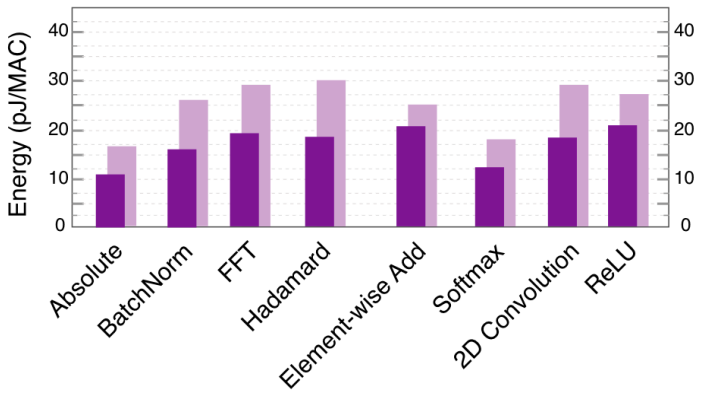


Figure 15: DSP accelerated algorithms show best overall results because of domain-specific environment for convolutions. In dark purple: Q-format with auto-differentiation. In light purple: the IEEE 754-2008 results under similar conditions.

We divided our experiments into multiple phases for as many devices as we have. Each phase measures energy and speed, along with precision error for several algorithms and the energy gains compared to floating numbers.

Our sample implementation uses a 32-bit Q-format equivalent to IEEE 754-2008 single floating number. Other bit schemes can be used. Conventionally switching from a single floating number to a double floating number is used to further increase the precision of operations at the expense of computing resources. When representing images as 2D matrix with Q-format notation, convolution operations benefit from the reduced errors caused by rounding and a few other known limitations in the IEEE 754-2008 standard.

Table 4: FPBench aggregate precision error comparative results on AM572x with C66x DSP (with arbitrary rounding modes)

Operation	Q-32 roundup (*e-23*)	Q-32 truncate (*e-23*)	Float-32 round-to-nearest (e-23)
Sum	6.9	7.1	7.2
Logexp	6	6.3	6.2
matrixDet1	11.7	13.7	13.9
matrixDet2	11.8	14.1	14.4
Sqrt_add	9.9	10.3	11

Abbreviations: Logexp (exponential logarithm), matrixDet1, 2 (determinant matrix 1, 2), Sqrt_add (addition of square roots)

The proposed method to improve the convolution can be interpreted under the lens of the systolic array concept. Matrix multiplication in Figure 14 and its execution in Figure 15 present similarities, at least visually, between a 2D matrix layout and an array of MAC units. Matrix multiplication can be considered a systolic structure because one matrix is fed one row at a time while the other is fed a column at a time. Values are passed into MAC units until each unit has seen a whole row or column. At this point the result can be passed back to memory and the structure can proceed with its flow down and across the input arrays.

By comparison, a CPU differs from a systolic design in its per-watt performance capacity, which is a "by design" constraint since CPUs are general-purpose processors. Any processor featuring a systolic design is not general-purpose, but its limitation is also its raison d'être. To implement its generality, CPUs store values in registers, and a controller tells the ALU (arithmetic logic unit) which register to load, the operation to perform (addition, multiplication, etc.) and the register into which to store the result. A program is made of a sequence of such instructions to read, operate, then write but each step has a cost in resources such as time, energy, and storage.

Systolic arrays typically reduce the number of reads by reusing one register's value multiple times before completing a read-operate-write cycle. As discussed in 2.5 the Winograd algorithm has limits with regards to the input size. Our implementation is able to chop an input matrix into equal size sub-matrices and distribute them among all available registers to limit the off-chip shared memory access.

For as long as the critical path of the proposed convolution algorithm is taking place there is reduced trips outside of the acceleration chip, and the resulting effect is reflected in the results (Figures 11, 12, 13, and Table 4).

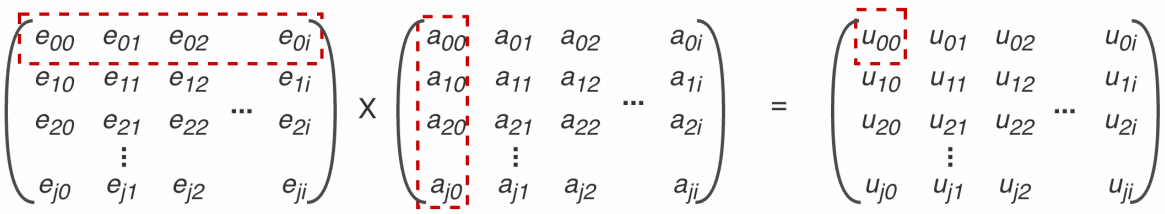


Figure 16: Matrix multiplication rule

The other reason is largely attributed to the way our algorithm leverages chip architecture to optimize how input is fed to the accelerator. There are 2 ways input is fed: one is sequentially which may present bottlenecks and leave certain portion of Pes waiting until they are activated depending on the size of the input; the other essentially fills all the input registers of the MAC units in a systolic manner, which means that the critical path of the execution is not blocking or forcing processing to wait for activation.

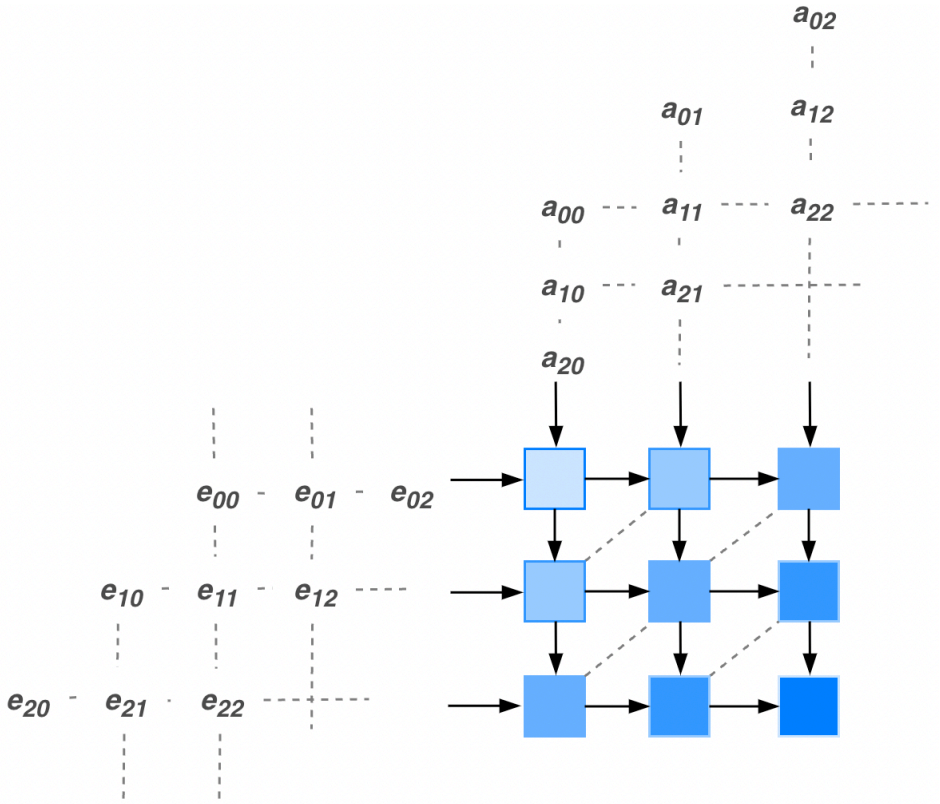


Figure 17: MAC units are activated and passing input downstream in a manner which fills the blue array diagonally. From the first input (top-left square) down to the last unit (bottom-right square)

Q-format notation has the same energy requirements as integers which are less then floating-point requirements by design. By proposing this notation, we

provide a possibility to lower the energy requirements without affecting the accuracy of models built because of this thesis proposals.

Systolic arrays ensure the entirety of the input is kept on the processing unit's internal registers and buffers, and that only the result is put back into shared memory or persisted to disk. Traditionally only chunks of the input would be fed to the processor, one at a time, resulting in unnecessary back-and-forth exchanges that cause a heavy toll on the energy consumption.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

In this work we propose an alternative notation that mimics integer format. We are interested in analyzing the gains this replacement has on embedded devices in terms of power consumption and latency.

Our evaluation shows gains in throughput and latency on a few common algorithms used in deep learning. We show up to 21% gain in energy savings and up to 22% reduced latency while maintaining optimal output error (2.1% average relative error). Our measurements are based on multiply-accumulate (MAC) with throughput, or performance expressed in the number of MACs per second, and energy in pico-Joules per MAC. Processor manufacturers publish theoretical numbers for speed, usually expressed in FLOPS (floating operations per second) which we based our estimated latency reduction on.

Assistive intelligence is a growing field provides interfaces where continuous input is expected throughout execution. The foundations of computing architecture can be revisited without stalling the progress achieved. Hybrid hardware-software tinkering is open to improvement and better numerical approximations. Support systems and feedback loops, especially in human-in-the-loop systems, benefit from the mobility and low-power requirements of small-factor computing architectures, and it will only serve to achieve more research in areas such as healthcare, self-driving cars, or robotics. In specific applications like diagnosing cancer from MRI scan data can only assist, not replace, the experience of a professional. A handy assistant is one that is always within reach.

To lower the cost of implementation, DSP processors are designed for use with only integers. Q-format is a notation that allows to mimic floating numbers by using integer types. By manipulating the binary notation of the fractional and exponent parts of a decimal number, we can spoof the processor into supporting

digital floating number arithmetic. Depending on where the binary point is placed, a given number can be interpreted as several different values. For simplicity, we generally rely on the programming language being used to maintain a fixed binary point. In the case of Q-format notation that means explicitly stating how many bits hold the exponent, and how many hold the mantissa. For example, to specify that we are using 3 bits for the exponent part and 4 bits for the mantissa, we define a Q-format notation as Q3.4. Although this notation is only defined once and used for all values in the expected operations, the choice of bits is a strategic one to avoid any possible overflows or under-utilization that may cost in terms of resources down the execution pipeline. Adding two N-bits Q-format numbers may result in a N+!-bits value. Any such possibilities must be supported and safeguarded when designing the notation. Once set, a Q-format must be maintained.

Q-format notation demands a manual overhaul in the number notation design, which require advanced skills in several fields in computer science and artificial intelligence. Researchers are encouraged to spend more of their skills in lowering the barrier of entry for all people to join this field. That is a much better endeavor to making deep learning an affordable problem-solving toolkit.

A major consideration in future works is to fully implement a task such as object classification or pose estimation using the proposed methods and comparing whether the accuracy of the models generated maintain state-of-the-art results with minimal consumption.

Proven mathematical operations like the Hadamard product have helped improve convolutional neural networks in the context of digital arithmetic. Others have proposed to replace multiplications with additions to reduce the load on the processing unit or holding critical operations on the processing unit until a result is obtained to reduce the travel between memory and processor. In point 2.1 of the literature review we discuss how matrix to matrix multiplication has always been a difficult operation on computers. We also show that much has been done

to make it easier, highlighting key benchmarks and achievements that are continuously contributed by mathematicians and computer scientists. What we don't mention is that most of these achievements consider matrix that are too large for common computers to handle. The algorithms designed to break the records we highlight are called galactic algorithms because they are too large to handle on current computers. The Winograd algorithm however is not in that category, which means that its benefits are directly accessible to not only high-performance computing, but to embedded systems as well.

Optimizing algorithms is never a completed task since the host systems they rely on evolve, and therefore opportunities for improvement arise. In some cases, FFT-based convolutions can be faster than Winograd minimal filtering, especially since the latter is only limited to small kernels (3x3 sizes). We use both methods to make room for as many use cases as possible for the next phases of our experiments, when we put into consideration the accuracy of the results achieved using our proposed method. We can do so by completing a model architecture that takes in values in Q-format notation, manipulates them and outputs a score or some type of metric that we can compare with existing floating-point based models, such as the open-source collection of models compiled by the TensorFlow Lite community.

BIBLIOGRAPHY

1. Johnson, J., “Rethinking floating point for deep learning,” arXivpreprint arXiv:1811.01721, 2018.

2. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H.,and Kalenichenko, D., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” 2017.

3. Lin, D., Talathi, S., and Annapureddy, S., “Fixed point quantization of deep convolutional networks,” International conference on machine learning, pp.2849–2858, PMLR, 2016.

4. Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A.K., Constable, W., Elibol, O., Gray, S., Hall, S., Hornof, L. and Khosrowshahi, A., “Flexpoint: An adaptive numerical format for efficient training of deep neural networks.”, Advances in neural information processing systems 30, arXiv:1711.02213, 2017.

5. Roel, S. D., Gilbert, T., Kohli, N., “A Broader View on Bias in Automated Decision-Making: Reflecting on Epistemology and Dynamics”, ICML, 2018.

6. Gustafson, J.L. and Yonemoto, I.T., “Beating floating point at its own game: Posit arithmetic”, Supercomputing frontiers and innovations, 4(2), pp.71-86, 2017.

7. “Precision & performance: Floating point and ieee 754 compliance for nvidia graphical processing units (GPUs)” https://developer.nvidia.com/sites/ default/files/akamai/cuda/files/NVIDIA-CUDA-Floating-Point.pdf, 2020. Retrieved: 2020-11-08.

8. Manolopoulos, K., Reisis, D., “An efficient dual-mode floating-point multiply-add fused unit”, 2010 17th IEEE International Solid-State Circuits Conference (ISSCC), pp. 82-83, 2016.

9. Babu, N. S., Himaja, V. and Srinu, B., “High performance advanced signed multiplier for DSP applications”, Proceedings of the Indian J.Sci.Res, volume 17-12, pp. 329-331, 2018.

10. Sze, V., and Chen, Y.H., Yang, T. J., and Emer, J. S., “Efficient processing of deep neural networks: A tutorial and survey”, Proceedings of the IEEE, volume 105-12, pp. 2295-2329, 2017.

11. Hongxiang, F., Cheng, L., Chenglong, Z., Martin, F., Zhiqiang, Q., Shuanglong, L., Xinyu, N. and Wayne, L., "F-E3D: FPGA-based acceleration of an efficient 3D convolutional neural network for human action recognition", 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), volume 2160, pp. 1-8, 2019.

12. Yang, C., and Kurth, T., and Williams, S., "Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system", Concurrency and Computation: Practice and Experience, volume 32-20, pp. 5547, 2020.

13. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., "Mastering the game of Go with deep neural networks and tree search.", nature, 529(7587), pp.484-489, 2016.

14. Wijeratne, S., and Jayaweera, S., and Dananjaya, M., and Pasqual, A., "Reconfigurable co-processor architecture with limited numerical precision to accelerate deep convolutional neural networks", 2018 IEEE 29th international conference on application-specific systems, architectures, and processors (ASAP), pp. 1-7, 2018.

15. Rastegari, M., Ordonez, V., Redmond, J., Farhadi, A., "XNOR-Net: ImageNet classification using binary convolutional neural networks", arXiv 1603.05279, 2016.

16. Lin, D., Talathi, S., and Annapureddy, S., "Fixed point quantization of deep convolutional networks," International conference on machine learning, pp.2849–2858, PMLR, 2016.

17. "Overcoming challenges in fixed point training of deep convolutional networks", ICML Workshop, 2016.

18. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H.,and Kalenichenko, D., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

19. Ruizhe, Z. and Wayne, L., "Optimising Convolutional Neural Networks for Reconfigurable Acceleration", doi 10.13140/RG.2.2.29524.30083, 2017.

20. Mathieu, M., Henaff, M., and LeCun, Y.. "Fast training of convolutional networks through ffts." arXiv preprint arXiv:1312.5851 (2013).

21. Vasilache, N., Johnson, J., Mathieu, M., Chintala, S., Piantino, S. and LeCun, Y., “Fast Convolutional Nets With fbfft: A GPU Performance Evaluation”, arXiv:1412.7580, 2015.

22. Lavin, A., and Gray, S.. "Fast algorithms for convolutional neural networks." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4013-4021. 2016.

23. Kochura, Y., Gordienko, Y., Taran, V., Gordienko, N., Rokovyi, A., Alienin, O., and Stirenko, S., “Batch size influence on performance of graphic and tensor processing units during training and inference phases”, International Conference on Computer Science, Engineering and Education Applications, pp. 658-668, 2019.

24. Seongsik, P., Seijoon, K., Seil, L., Ho, B., Sungroh, Y., “Quantized memory-augmented neural networks”, Proceedings of the AAAI Conference on Artificial Intelligence, volume 32-1, 2018.

25. Baydin, A. G., Pearlmutter, B. A., Radul, A. A. and Siskind, J. M., “Automatic differentiation in machine learning: A survey”, Journal of Machine Learning Research, Vol.18, No.153, pp.1-43, 2018.

26. Deng, L., and Hinton, G. E., “New types of deep neural network learning for speech recognition and related applications: an overview”, IEEE International Conference on Acoustic, Speech and Signal Processing, pp. 8599-8603, 2013.

27. Martín, A., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M. et al. “Tensorflow, A system for large-scale machine learning”, 12th USENIX symposium on operating systems design and implementation (OSDI 16), pp. 265-283. 2016.

28. Paszke, A., Gross, A., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T. et al. “Pytorch, An imperative style, high-performance deep learning library”, Advances in neural information processing systems 32 (2019): 8026-8037.

29. Van Merriënboer, B., Breuleux, O., Bergeron, A., Lamblin, P., “Automatic differentiation in ML: Where we are and where we should be going”, arXiv:1810.11530, 2018.

30. Alman, J., Williams, V.V., “A Refined Laser Method and Faster Matrix Multiplication”, 32nd Annual ACM-SIAM Symposium on Discrete Algorithms, 2020.

31. LeCun, Y., Bengio, Y., and Hinton, G. E., “Deep Learning”, Nature 7553:436, 2015.

32. Johnson, J., “Rethinking floating point for deep learning”, arXiv preprint arXiv:1811.01721, 2018.

33. Tagliavini, G., Mach, S., “A transprecision floating-point platform for ultra-low power computing”, Design, Automation & Test in Europe Conference & Exhibition IEEE, pp. 1051-1056, 2018.

34. Mingzhen, L., Yi, L., Xiaoyan, L., Qingxiao, S., Xin, Y., Hailong, Y., Zhongzhi, L., Lin, G., Guangwen, Y., Depei, Q., “The deep learning compiler: A comprehensive survey”, IEEE Transactions on Parallel and Distributed Systems, volume 32-3, pp. 708-727, 2020.

35. Gautschi, M., Schiavone, P. D., “Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems", volume 25-10, pp. 2700-2713, 2017.