## Project 20-1-1-2187

# CNN Acceleration and Simulation on an FPGA

### Chaim Gruda 312562721

### Shay Tsabar 208723627

## Instructor: Yoni Seifert

#### Tel Aviv University

##### Faculty of Engineering

## **Abstract**

In recent years image analyzing and recognition has become a vital feature in ever more applications, ranging from autonomous vehicles, security, healthcare and more. Convolution neural networks (CNN) algorithms are widely used in many such applications since its high accuracy for image recognition. However, with the high accuracy come very high computational complexity. The need to include such image recognition capabilities in systems with tight real-time and power constraints, lead the design for hardware accelerators for CNNs. Such accelerators may be implemented on an FPGA, GPU, or ASIC.

In this project we explored the potential of FPGA-based CNN acceleration, by implementing and simulating the CNN operation, and comparing it to software-based implementation of the same CNN.

The CNN accelerator was synthesized using High Level Synthesis (HLS), and the FPGA platform used is the Zynq-7000 based ZedBoard™.

## **Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 4 |
|  | 1.1 | Artificial Neural Networks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 4 |
|  | 1.2 | Convolutional Neural Networks . . . . . . . . . . . . . . . . . . . . . . . . . . | 5 |
|  | 1.3 | CNN Acceleration . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
|  | 1.4 | Field-Programmable Gate Arrays . . . . . . . . . . . . . . . . . . . . . . . . . . | 6 |
|  |  |  |  |
| 2 | Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | | 7 |
|  | 2.1 | High Level Synthesis |  |
|  | 2.2 | Sliding Window |  |
|  | 2.3 | Processing Element |  |
|  |  |  |  |
| 3 | Implementation | | 11 |
|  | 3.1 | FIFO |  |
|  | 3.2 | AXI interface |  |
|  | 3.3 | ZedBoard |  |
|  |  |  |  |
| 4 | Results | | 16 |
|  | 4.1 | CNN Simulation |  |
|  | 4.2 |  |  |
|  |  |  |  |
| 5 | Discussion | | 18 |
|  | 5.1 | Conclusion |  |
|  | 5.2 | Future Work |  |

## **1 Background**

**1.1 Neural Networks**

Neural networks are a family of computation architectures inspired by the biological nervous system. That system consists a large amount[[1]](#footnote-1) of neurons, which are electrically excitable cells, that are connected to each other in junctions named synapses[[2]](#footnote-2), forming a massive network. Each neuron receives input signals and produces output signals which branch out and connect to the other neurons. The synapses connecting the neurons influence the transfer of information from one neuron to the other by amplifying, attenuating, or inhibiting the signals transfer. Together, the billions of simple neurons form an incredibly complex interacting network which enables all the brain activity.

The basic building block of an Artificial Neural Network (ANN) is the artificial neuron. The artificial neuron receives several input signals from other neurons. These input signals are multiplied with weights to simulate the synaptic interaction. The weighed input signals are summed, biased, and fed into a non-linear activation function, which produces the neuron’s output signal. A neural network is formed by interconnecting many artificial neurons. Usually, the neurons are grouped into layers, and connections are only allowed between neurons of adjacent layers. Such layered connections are called Fully Connected layers.

When neural networks are employed for image-related tasks, their input usually consists of pixel data. Even for images with modest resolutions, the input consists of large amounts of elements (especially when dealing with multiple channel images, such as RGB). Large input data results in a need with amounts of neurons and connections that are a challenge for computing systems, and place a burden on the efficiency, both in time and in power, of analyzing images effectively in such neural networks.

**1.2 Convolutional Neural Networks**

Convolutional Neural Networks (CNNs) are a special class of neural networks, suited for operation on 2D input data such as images. The idea behind the CNN is the locality of information in images, which means that important information in images  
can be captured from local neighborhood relations. To decide whether there is a car in the center of an image, one does not need to consider the top-right corner pixel, and the bottom-right pixels usually do not influence the class assigned to the top-left pixels. Strong contrasts indicate edges, aligned edges result in lines, combined lines can result in circles and contours, circles can outline a wheel and multiple nearby wheels can point to the presence of a car. This locality of information in images is exploited in convolutional neural networks by adding new layers before the fully connected layers, namely convolutional layers, and pooling layers.

The purpose of convolutional layers (CONV) and pooling layers (POOL) is to reduce the spatial data of 2D images into smaller data sets, that contain the most distilled data from within the image. After that process is done, the refined data can be passed through fully connected layers (FC), which will now be much smaller and tailored for the purified information. A CNN may have multiple CONV and POOL layers.

* **Convolutional layer** – This layer preforms the mathematical 2D convolution. This operation is described by Eq. (1):

|  |  |  |
| --- | --- | --- |
| (1) |  |  |

The operation basically slides filter of size , where each pixel in the filter has its own weight , over the 2D image, and computing the sum of the product of all overlapping pixels. The filter is also known as a kernel, and the convolution output is called a feature map. The operation is described in **Figure 1**.

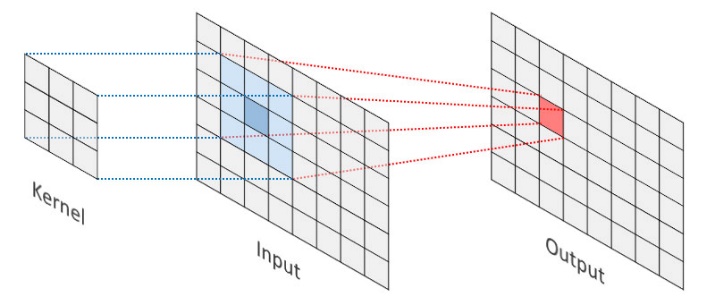


Figure 1: 2D convolution illustration

* **Activation Layer –** This layer applies a non-linear activation over the CONV layer, introducing non-linearity into the CNN. The non-linearity increases the power of deepening the network (moreover, if the CNN would be linear then a set of N layers could be represented in a single layer). One of the most common non-linear activation functions used in CNNs is the Rectified Linear Unit (ReLU) function expressed in Eq. (2) and seen in **Figure 2**.

|  |  |  |
| --- | --- | --- |
| (2) |  |  |

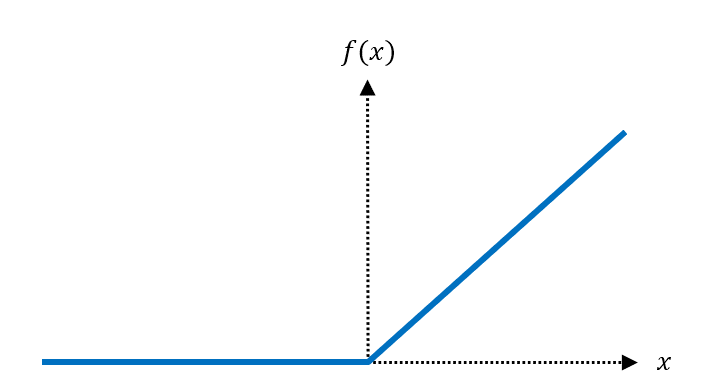


Figure 2: ReLU function

* **Pooling layer** – This layer reduces the spatial area of the given input, usually being the output of a convolutional layer. This is done by selecting the max value or calculating the average value from a 2D section. These are called Max-Pool and Avg-Pool, respectively. Max polling can be visualized in **Figure 2**.

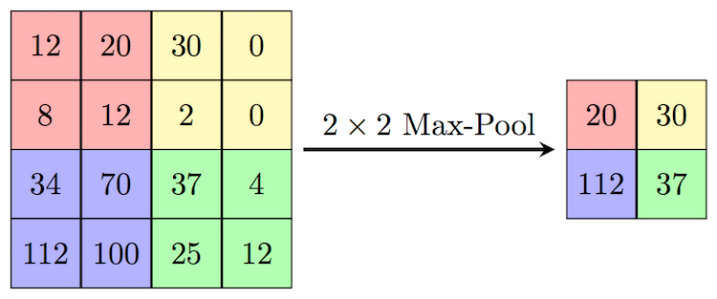


Figure 3: Max pooling illustration

* **Fully Connected layer** – The operation of FC layer can be expressed by Eq. (3). Input features are fully connected to an output feature vector by weights (). The and are the lengths of the input and output feature vector and the number of weights is . is the corresponding bias term of the th output feature vector. FC layers usually turn the input feature maps into a feature vector.

|  |  |  |
| --- | --- | --- |
| (3) |  |  |

**1.3 CNN Acceleration**

As CNNs achieve very high accuracy in image understanding and classification, they get embedded in more applications, some of which require tight real-time constraints. This leads to the need to accelerate the CNN calculations.

Most of the computations done in the CNN is in the convolutional layers, around 85-90%[[3]](#footnote-3). Since all layers: CONV, POOL and FC, basically deal with matrices, the software implementation of those layers consists of nested loops. This means all computations are done serially, and thus the time the network operates on a given output may be very long.

To reduce the computation time, and thus to accelerate the CNN performance, the different layers can be implemented in hardware. Since hardware can preform computations in parallel, this basically allows to expand the software loops so that a CONV single operation can be done in a single clock cycle.

Potential hardware platforms that can be used for accelerating the CNN are DSPs, GPUs, ASICs, and FPGAs. The advantages for using an FPGA are discussed in the following section.

**1.4 Field-Programmable Arrays**

Field-Programmable Gate Arrays (FPGAs) are semiconductor devices consisting of a 2D array of configurable logic blocks which are connected via programmable interconnects. The interconnect can be thought of as a network of wire bundles running vertically and horizontally between the logic slices, with switchboxes at each intersection. Modern high-end FPGA generations feature hundreds of thousands of configurable logic blocks, and additionally include an abundance of hardened functional units which enable fast and efficient implementations of common functions. The logic blocks, the fixed-function units as well as the interconnect are programmed electronically by writing a configuration bitstream into the device. The configuration is typically held in SRAM memory, and the FPGAs can be reprogrammed many times

The advantage of FPGA-based systems over traditional processor-based systems including GPUs, is the availability of freely programmable general-purpose logic blocks. These can be arranged into specialized accelerators for very specific tasks, resulting in improved processing speed, higher throughput, and power savings. This advantage comes at the price of reduced agility and increased complexity during the development, where the designer needs to carefully consider the available hardware resources and the efficient mapping of his algorithm onto the FPGA architecture. Further, some algorithmic problems do not map well onto the rigid block structures found on FPGAs. Compared to ASIC (Application-Specific Integrated Circuits) implementations, the FPGA advantages are re-programmability, that allows fast prototyping and short development cycles

## **2 Design**

**2.1 Vivado HLS**

High level synthesis (HLS) is a design process in which a high-level description of a design is compiled into RTL implementation that meets specified constraints. The design is ‘high level’ compared to RTL in two aspects:

1. **Design abstraction:** HLS input is an untimed dataflow and computation specification, rather then RTL cycle by cycle specification.
2. **specification language:** HLS input is specified in languages like C, C++, System C, or MATLAB, and allows use of some more advanced language features like loops, arrays and more.

The output of HLS include:

1. **RTL implementation**: This includes the RTL netlist that contain the datapath, control logic, interfaces to I/O, host, and memories; as well as scripts, libraries, and synthesis timing constraints required to synthesize the RTL netlist using conventional [logic synthesis](https://www.sciencedirect.com/topics/engineering/logic-synthesis) flows.
2. [Analysis feedback](https://www.sciencedirect.com/topics/engineering/feedback-analysis): This includes GUI and reports on performance bottlenecks, mapping of high level source code to RTL, hardware costs, etc, to help user understand and improve the micro architectures.

III. **Verification Artifacts**: This includes simulation test bench, linting checks, scripts, and library for code coverage, etc., to help user develop and debug high level language test suite and reuse these tests for RTL verification.

User specified constraints help HLS construct the desired micro architecture. These constraints include:

1. Target hardware: This includes the platform, technology library, clock frequency, etc, that the design is intended for. HLS uses this information to estimate sub-cycle timing and cost of the datapath.
2. Performance constraint: This may be expressed in the form of input sampling rate, output production rate, input-to-output latency, loop initiation intervals, loop latency, etc. These constraints impose cycle level timing constraints on the micro architecture.
3. Memory architecture: This specifies how multi-dimensional arrays are mapped to memories and [memory interfaces](https://www.sciencedirect.com/topics/engineering/memory-interface), allowing HLS to construct micro architectures containing mult-port, multi-bank, arbitrated, external and internal memories.
4. Interface constraint: This includes the protocol, ports, and handshake/arbitration logic to create at each input, output, host, and external memory interface. HLS generates these interface ports and logic in the RTL netlist so it can be easily integrated with other hardware blocks.
5. Design hierarchy: This partitions a design using hierarchy in the high level input description, allowing HLS to manage design complexity by divide-and-conquer.

HLS is not a substitute for a good RTL designer. For example, if the micro architecture is given, designing in RTL is easier and sufficient. HLS is designed for exploring different algorithms and architectures to find the best micro architecture under a variety of constraints. The primary benefits of HLS derive from its support for high level of abstraction and high level specification language:

Benefits of designing at a high level of abstraction:

* Allows focus on designing core functionality, not implementation details. Easily explore different architectures.
* Easily evaluate algorithmic changes.
* Easily generate memory, IO, and host interfaces, as well as pipeline, stall, handshake and arbitration logic, etc.
* Easily retarget for different hardware constraints or performance from the same input description.

Benefits of verifying at a high level of abstraction:

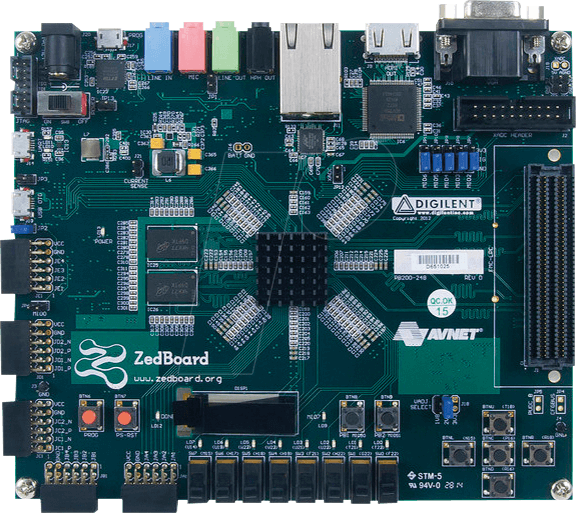
* Easily debug and test functionality of input descriptions.
* Fast and free simulation.
* Test suite is reusable for RTL verification.
* Code coverage and functional coverage is more meaningful and easier to achieve.

Benefits of high level specification language:

* Legacy code can be reused for design and for verification.
* Software development tools (e.g., Visual Studio) can be used.
* Support advanced language features like polymorphism and templatized classes for concise, reusable input descriptions.

**2.2 ZedBoard FPGA**

The FPGA platform used for this project is the Xilinx ZedBoard Zynq-7000 based FPGA, seen in **Figure X**.



**Zynq-7000**

**DDR3**

The Zynq-7000 contains two sub-systems:

1. Processing System (PS) – 2 ARM Cortex A9 CPUs
2. Programmable Logic (PL) – an FPGA

The two subsystems are utilized and work together in the CNN implementation. The PL containing the CNN acceleration cores, and the PS running the cores drivers – initializing and configuring the CNN, feeding it data, and receiving the results. The communication between the systems is by AXI interface. Both systems also have access to the DDR3 memory using an AXI-DMA core controlled by the PS.

**2.3 Fixed Point Calculations**

To simplify HW implementation, data in HW is represented in fixed point format. Out of 32 bits of word length, 10 bits are used to represent the fraction. This gives a resolution of . The downside of using fixed point is the loss of accuracy in calculations, which leads to less accuracy of the accelerated network. The accuracy loss can be seen in the simulation results.

The reason for using 10 bits only for the fraction, is since when multiplying 2 fixed point numbers, the resulting word will have double the bits representing the fraction. Thus, using more bits for the fraction would lead to overflows in the integer values of the represented numbers.

**2.4 Sliding Window**

Since the input data is held in PS memory as an ordered array, when preforming CONV or POOL operations on the data, out of order pixels need to be extracted and operated on. Preforming the extraction in software is quite simple since the array is stored in RAM. In HW this is more complex, and thus there are 2 possible solutions. The simple one is to utilize the SW to extract the needed data per operation and feed it to the HW to calculate. However, this means data pixels are passed from SW to HW multiple times, causing a bottle neck that may reduce the acceleration affect.

The more advanced way to preform the operation, is to have a sliding window mechanism implemented in HW. That way the ordered array can be passed in an ordered stream to HW, and HW will know what pixels to operate on at any given cycle. The sliding window concept is show in **Figure X**.

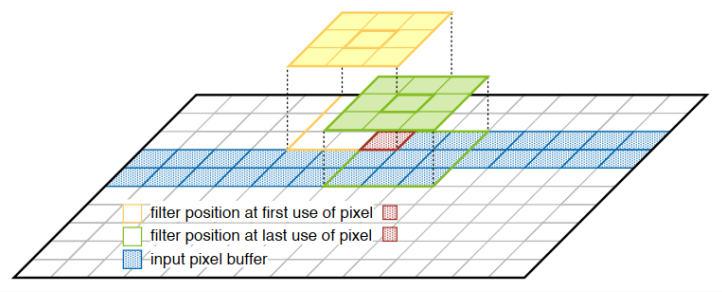


Figure 4:

The main advantage of the sliding window mechanism is that each pixel is passed to HW only one singe time. Another advantage is that the CONV and POOL layers can start preforming the operation on the full image, even as soon as the first needed data pixels arrive and pass their results to the next layers, which in their turn can start calculating even while prior layers are still calculating. This increases the acceleration affect and allows high data throughput in the CNN.

**2.4 System Flexibility**

To increase the systems flexibility, we have considered implementing HW blocks that can handle dynamic sizes of data input and operation (CONV/POOL) dimensions. This proved to complicate the system beyond the capabilities of the Vivado HLS and the Zedboard capabilities. moreover, one may argue that the FPGA would be better utilized if the HW blocks were exactly matched to the CNN data and operation dimensions. Allowing the resources to utilized to the maximum per application.

Instead of that, the solution we decided on was to completely automate the creation of the CNN, both HW and SW. the user need to only configure a settings file, setting the desired network layers data dimensions, and run a script. The script generates all the code, build the block design in Vivado and exports the bitstream to SDK with the needed drivers.

## **3 Implementation**

**3.1 Hardware Processing Element**

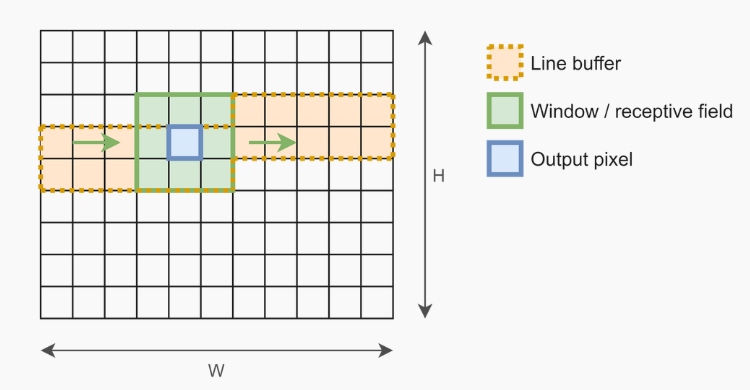
Each processing element

Three layer types were implemented

Conv Elements

Also include activation

Pool elements (support max and avg)



**3.2 Software Drivers**

## **4 Results**

Simulation explanation – timing , accuracy, on arm, HW, and PC

Simulation results

## **5 Discussion**

**5.1 Conclusion**

The

**5.2 Future work**

Fit for an actual CNN

Dynamic sizes

## **References**

1. For example: a mouse brain has approx. 70 million neurons, human brain approx. 100 billion neurons [↑](#footnote-ref-1)
2. Mouse: approx. 1 trillion synapses, human: approx. 125 trillion synapses [↑](#footnote-ref-2)
3. link [↑](#footnote-ref-3)