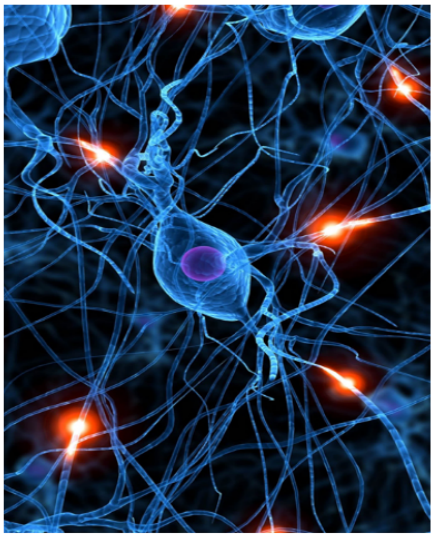
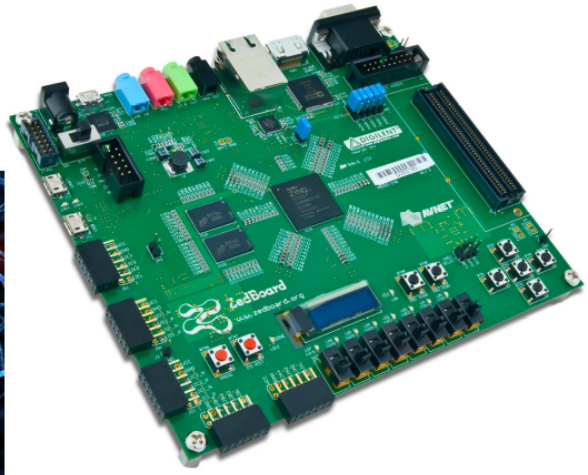
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2017





Project Title:	**Space Brain - Investigating the Suitability of an FPGA-based Convolutional Neural Network for Space Applications**
Student:	**Jacobus (Jukka) Johannes Hertzog**
CID:	**00828711**
Course:	**EE4T**
Project Supervisor:	**Dr. Felix Winterstein**
Second Marker:	**Dr. David Thomas**

1 Introduction

Inspired by the structure of the optical nervous system in animals, a neural network is made up of layers of artificial neurons that recognise certain features from the input data. The idea of an artificial neural network was first introduced in 1980 [1], and driven by increases in computing power and a growth in machine learning applications, neural networks have become very powerful tools. In the last decade, the development of Convolutional Neural Networks (CNNs) has lead to great advancements in image classification accuracy.

CNNs are a powerful Deep Learning tool that can be used to solve extremely complex computational problems. In particular, they have gained popularity in image classification applications [2]. Other popular applications within machine vision include video classification, face detection and gesture recognition. They are also being used in a wide range of other fields including speech recognition, natural language processing and text classification.

However, whilst achieving exceptional performance, CNNs are extremely computation and resource heavy. Typically, they will be implemented on large servers or on GPU based systems to accommodate the need for computational power. As a result, implementing them on embedded systems, which typically have very limited resources, presents many challenges. A promising solution to this problem is the use of an FPGA, which provide a very high computational efficiency with low power usage, in addition to other benefits.

Now suppose that the FPGA based CNN will be implemented on a satellite. In orbit, the satellite will be exposed to intense radiation, and as a result, errors will be produced on the FPGA. Before such a satellite could be designed and launched, it is important to know how these errors would affect the performance of the CNN, and if the CNN could be designed in such a way that could mitigate the impact of these errors. Therefore, the aim of this project is to implement a CNN on an FPGA-based system, and then to investigate the suitability of this system for space applications.

1.1 Report Structure

The rest of the report is organised as follows. Section 2 will describe the key concepts underlying the project. Section 3 will lay out the project specification, and the motivation for the project. Section 4 will go into detail about how the project will be implemented, including a rough time line. Finally, section 5 will describe the metrics through which the project will be evaluated.

2 Background

This section will provide some details and information that will provide context to the project, and will be useful for understanding key concepts in the rest of the report. The High Level Synthesis tools, CNN structures, FPGA implementations of CNNs, and the issues surrounding the use of FPGAs in space will all be discussed.

2.1 Target Platform

The hardware used in this project is a product from Xilinx called a Zedboard, a development board for the Zynq-7000 System On a Chip (SoC). This board was chosen because of its useful hardware features and its integrated support for useful Xilinx proprietary tools. These tools include High Level Synthesis

systems, SoC design tools and the Soft Error Mitigation IP core. The Zynq-7000 consists of a Xilinx FPGA as its Programmable Logic, and a dual-core Cortex-A9 ARM processor as its Processing System integrated together. The ARM processor facilitates easy control of the system, whilst the FPGA will be able to efficiently perform heavy computations.

2.2 High Level Synthesis

Typically, digital circuits on reconfigurable hardware architectures, such as FPGAs, are designed using a Hardware Description Language (HDL) such as Verilog or VHDL. While these methods allow the designer to have a great level of control over the system, and can produce extremely efficient designs, the designer is forced to specify functionality at a very low level of abstraction, specifying cycle by cycle behaviour. Use of HDL tools requires hardware expertise and, when designing a complex system, makes for a long and cumbersome development process.

High Level Synthesis (HLS) tools are a solution to this problem. An HLS tool is a piece of software which can interpret the desired behaviour of code written in a programming language like C or C++, and generate an HDL implementation of that behaviour, allowing the designer to work at a higher level of abstraction. This means that a software engineer can create FPGA designs without having to build up hardware expertise, allowing them to utilise the speed and efficiency of hardware designs. It also benefits hardware engineers, allowing them to design the system more quickly and reliably, facilitating the development of more complex systems [3]. Popular HLS tools include academic, open-source software like LegUp, and commercial tools like Catapult-C and Xilinx's Vivado. In this project, the embedded system will be designed using SDSoC from Xilinx, which utilises Vivado to produce HDL for the Programmable Logic in the Zynq SoC.

2.3 Convolutional Neural Networks

CNNs can be used to classify images in a forward inference process. But before using the CNN for any task, one should first train the CNN on a set of training data. The training of a CNN is usually implemented on large servers with a huge computational capacity, with an enormous set of training data. For the embedded FPGA platform, this project will only focus on implementing and utilising the inference process of a CNN, and not on the training process.

A typical CNN consists of a number of layers that run in sequence. Convolution (CONV), activation function, pooling, and Fully Connected (FC) layers make up a typical CNN model, with CONV and FC being the most important. CONV and FC layers have parameters of called weights, which are set by the training process. These weights will determine which patterns each layer of the CNN will be activated by, and ultimately, what the CNN is able to recognise and classify.

The first layer of a CNN reads an input image and outputs a series of feature maps. The input image will be three dimensional - the height and width of the image make up two dimensions, and colours make up the third layer. For an RGB image, the input will have a depth of 3. Then there will be CONV layers interspersed by activation function and pooling layers, which will make up most of the CNN. These layers will decompose the image into feature maps, varying from low-level features such as edges, lines, curves, etc., in the initial layers to high-level features in the deeper layers [4]. Each subsequent layer reads the feature maps generated by preceding layers and generates new feature maps at its output. Finally a classifier, consisting of at least one FC layer, reads the final feature maps and determines the probability of the input imaging belonging to each category of the training data. A example of a CNN model is shown in Figure 1.

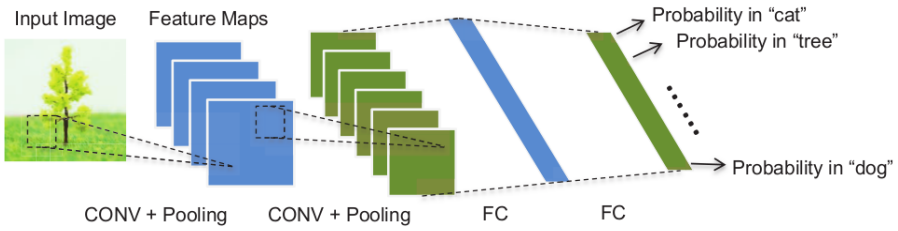


Figure 1: A typical CNN structure from the feature map perspective [5]

2.3.1 Convolution Layer

The CONV layer takes a series of feature maps as input and convolves with convolutional kernels to obtain the output feature maps. It involves 3-dimensional multiply and accumulate operation of N_{if} input features with $K \times K$ convolution kernels to get an output feature neuron value as shown in Equation 1

$$out(f_o, x, y) = \sum_{f_i=0}^{N_{if}} \sum_{k_x=0}^{K} \sum_{k_y=0}^{K} wt(f_o, f_i, k_x, k_y) \times in(f_i, x + k_x, y + k_y) \tag{1}$$

where $out(f_o, x, y)$ and $in(f_i, x, y)$ represent the neurons at position (x, y) in the feature maps f_o and f_i, respectively and $wt(f_o, f_i, k_x, k_y)$ is the weights at position (k_x, k_y) that gets convolved with input feature map f_i to get the output feature map f_o [4].

2.3.2 Activation Functions

CONV layers are followed by an activation function layer. This layer can be thought of as a decision, based on the output of the CONV layer, on to what extent each neuron in the next layer has been activated. The CONV layer output is a linear combination of the inputs and the weights at a position in the network, and role of the activation function is to produce a non-linear decision boundary. The commonly used activation functions in traditional neural networks are non-linear functions such as tanh and sigmoid, which require a longer training time in CNNs [6]. Hence, Rectified Linear Unit (ReLU), defined as $y = max(x, 0)$, has become the popular activation function among CNN models as it converges faster in training, and has less computational complexity than exponent functions in tanh and sigmoid, also aiding hardware design.

2.3.3 Pooling Layer

Spatial pooling or sub-sampling is utilized to reduce the feature dimensions as we traverse deeper into the network. As shown in Equation 2, pooling computes the maximum or average of neighbouring $K \times K$ neurons in the same feature map, which also provides a form of translational invariance [7]. The pooling layer helps abstract higher-level features without redundancy, as well as reducing the dimensionality of lower-level features without losing important information [4].

$$out(f_o, x, y) = \underset{0 \geqslant (k_x, k_y) < K}{max/average}(in(f_o, x + k_x, y + k_y)) \tag{2}$$

2.3.4 Fully Connected Layer

An FC layer is a classification layer where all the input features (N_{if}) are connected to all of the output features (N_{of}) through synaptic weights (wt). These are at the end of the CNN and perform the final classification, based on the features that have been recognised by the rest of the network. Each output neuron is the weighted summation of all the input neurons as shown in Equation 3 [4].

$$out(f_o) = \sum_{f_i=0}^{N_{if}} wt(f_o, f_i) \times in(f_i) \tag{3}$$

The outputs of the inner-product layer traverse through ReLU based activation function to the next inner-product layer or directly to a Softmax function that converts them to probability in the range (0, 1). The final accuracy layer compares the labels of the top probabilities from softmax layer with the actual label and gives the accuracy of the CNN model [4].

2.4 FPGAs and CNN Implementations

CNNs are computationally demanding and consume a lot of resources, and thus are difficult to implement on an embedded systems. FPGAs are the most promising platform for the implementation of an embedded CNN, due to their high computational efficiency and low power consumption. FPGAs are also more attractive than other digital hardware platforms, such as ASICS, because of their reconfigurability and fast development time, aided by HLS tools.

2.4.1 Implementation Challenges

One of the main challenges faced by FPGA implementations is memory bandwidth. A typical CNN model may have more than 60 million weights, which requires over 200MB of memory when represented with a 32 bit number. A standard commercial FPGA does not have enough on-chip memory to store this volume of data, so external memory must be utilised. Having to transfer this information to the FPGA can introduce a computational bottleneck. However, these weights are used in CONV layers multiple times. Thus, the system can be optimised by reducing frequent memory access. Chen et. al achieves this with a tiling method and dedicated buffers for data re-use. Another approach, is data quantization, which is discussed below. By quantizing the 32-bit floating-point weights into 16-bit fixed-point values, the pressure on the memory system can be alleviated. Suda et al. performed a study on the optimal precisions to balance accuracy and efficiency. It was found that the best choices were 8-bit precision for CONV weights, 10-bit precision for FC weights, with less than 1% compared accuracy loss compared to full precision weights [4].

2.4.2 FPGA CNN Accelerators

An FPGA implementation of a CNN, intended to improve the speed and efficiency of the CNN computation, is referred to as an FPGA CNN accelerator. Most research into FPGA CNN accelerators has focussed on using hardware to accelerating the computation of the CONV layers. These layers typically make up around 90% of the CNN's computational workload. The multiply and accumulate nature of the CONV layer's operation makes them ideal candidates for hardware acceleration, as FPGAs are able to perform these operations with great efficiency.

FPGAs also provide flexibility to implement the CNNs with limited data precision [4]. Many CNN accelerators use 16-bit fixed-point numbers instead of floating-point numbers to represent weights and data [8][9][10], especially because FPGAs. Using a shorter, fixed-point data representation can make significant reductions to the memory footprint and the use of computational resources. The CNN's data must be quantised in order for the fixed-point implementation to work, which will introduce a quantization error into the result. However, Chen et al. showed that using a 16-bit fixed-point implementation rather than a 32-bit fixed-point implementation only added an extra 0.26% to the error rate when using the MNIST dataset.

2.5 FPGAs and Space Applications

2.5.1 Single Event Upsets

Electronic circuits are sensitive to high-energy ionizing radiation, which can induce errors called Single Event Upsets (SEUs) [11]. In space this is a particular problem, as the radiation shielding provided by the Earth's magnetosphere, and enjoyed by systems within the Earth's atmosphere, is no longer present. A voltage pulse from an SEU can cause errors including altering digital values. SEU induced voltage pulses are problem for all circuits in space, but the nature of reconfigurable circuits means that they are particularly vulnerable. These SEUs are typically manifested as a bit flip in a memory cell or a transient logic pulse in combinational logic. Habinc et al. describes three main categories of SEU [12].

The first category is the configuration upset. FPGAs are configured by loading the configuration bitstream into an internal configuration memory. This memory controls the configurable logic elements and how the connect together. A configuration upset occurs when an SEU alters a value in the configuration memory, and thus changes the function of the configuration.

The second category is the functional upsets in user logic are SEUs within the FPGA's logic. The FPGA's user logic cannot be tested through a readback of the configuration memory, because their contents will change as part of normal logic operation. These errors can cause transient glitches in combinatorial logic, and can upset sequential logic where the state of the system is vital to its function [13].

The final category is the architectural upset. This is when an SEU occurs in the control elements of the FPGA, e.g. the configuration circuits or reset control [12]. This can have an enormous range of consequences. For example, an SEU in the reset control could cause the FPGA to be unintentionally reset and all state information would be lost as the FPGA is reconfigured. In this project, the other two forms of SEUs will be the main focus of the investigation.

2.5.2 SEU Mitigation Techniques

As a result of the problems that can be caused by SEUs, many techniques have been developed to mitigate the impact of SEUs in space. A few popular design techniques for overcoming SEU errors are described below. Although this is outside the scope of the project, some of these techniques could potentially be used to increase the SEU resilience of the FPGA CNN implementation.

A very effective way of detecting configuration upsets is to readback and verify the configuration memory of the FPGA. As long as the external memory holding the bitstream is uncompromised, any discrepancies caused by SEUs can be detected and corrected [12]. The disadvantage of this method is that any sequential logic within the FPGA will lose all state information.

Functional Triple Modular Redundancy is another popular method of mitigating SEU errors [13]. This involves making three copies of a given design within the FPGA fabric, and having them 'vote' on the result through a combinatorial circuit. This is very effecting at reducing the error rate from combinatorial and user upsets, as two of the three circuits need to be compromised in the same way in order for a false result to be given as the output of the system.

Radiation hardening is the act of physically making a chip less susceptible to radiation, typically by encasing it in a substance that will shield the internal electronics [14]. Many FPGA vendors sell radiation hardened versions of their products, and it should greatly reduce the impact of SEUs, but it is often a very expensive solution due to the cost of the 'rad hard' hardware. In this project, we would like to use standard commercial hardware, so this will not be used.

3 Requirements

The first deliverable is the implementation of a CNN on an embedded platform. The processor, with it's easily developed code, will control the system, while the FPGA, being able to efficiently perform heavy computations, will be used to accelerate functions. The end result will be a fully functional CNN that runs on the Zedboard and is able to process and classify images with a reasonable execution time and accuracy, compared to implementations from literature, as discussed in section 5.1.

The second deliverable of this project is the investigation of the system's fault tolerance against SEUs. As discussed in section 2.5, SEUs from are one of the main challenges to be overcome when using SRAM based FPGAs for applications in space. The system's resistance to SEUs should indicate whether or not the system might be suitable as an on board processor for a satellite in orbit.

4 Implementation

This section will provide a breakdown of what work will be done in the project, and will provide an estimate of when each milestone will be completed. A Gantt chart is provided in Figure 2

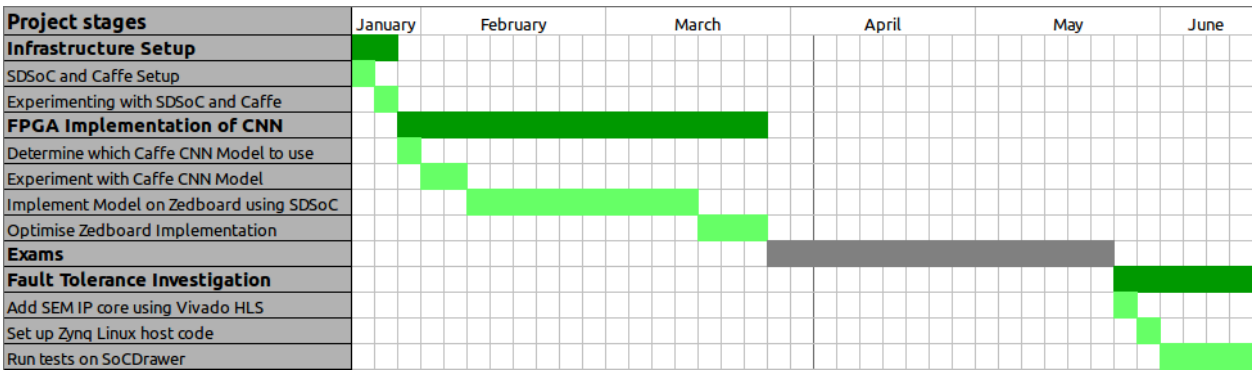


Figure 2: A Gantt chart outlining an estimated timeline for the implementation plan

4.1 Infrastructure Setup

The work will be performed on a Ubuntu computer, and the first part of the project is setting up the necessary tools on the computer. The first tool to setup up is the Xilinx SDSoC development environment. SDSoC will be used to design the system. SDSoC is able to optimise and compile C, C++ or OpenCL source code for a Zynq system. The compiler generates software for the ARM core and, using an HLS tool, a bitstream for the FPGA. This allows the user to design the entire system, easily accelerating functions with the FPGA. SDSoc is a very powerful tool, facilitating the rapid development and design of firmware.

The other important tool to set up is called Caffe (or Convolutional Architecture for Fast Feature Embedding). This is a deep learning framework that can be used to design, train, optimise and test CNNs, with a focus on computer vision [15]. Caffe also includes some examples of pre-trained CNN models that are ready for use. Using Caffe, it will be possible to generate C++ code for a CNN, which can then be implemented on the Zedboard system using SDSoc. This should be completed by the 31st of January, and some experiments will be performed using Caffe and SDSoC until the 3rd of February to build familiarity with the software.

4.2 FPGA Implementation of a CNN

The next part of the project is implementing a CNN on the Zedboard platform. Using SDSoC, control software will be designed to run on the ARM processor, and the layers of the CNN will be implemented in HLS to run on the Programmable Logic. A pre-trained model of a CNN will be downloaded using Caffe and tested on a computer, and then it will be implemented using SDSoC and tested on the Zedboard. Once this is complete, optimisation will be performed wherever possible, the literature concerning FPGA based CNN accelerators will be re-reviewed to check for ways to improve performance. It is expected that this should be complete by the 24th of March.

4.3 Fault Tolerance Investigation

The final stage of the project is to perform experiments, in order to evaluate the system's suitability for space applications. Rather than creating SEUs with actual radiation, they can be simulated through fault injection [16]. An important tool in this process will be the Xilinx Soft Error Mitigation (SEM) Controller. The SEM IP core is able to perform SEU detection, correction, and classification, and can also simulate SEUs with error injection [17]. By adding the SEM core to the system using Xilinx's Vivado HLS tool, the system's resistance to SEUs can be tested and measured.

The tests will be run using Zynq Linux host code. Xilinx provides a Linux image which can run on Zynq systems, including a kernel and a root file system. Using this, high quality results can be printed by the system as the tests are run. Once the setup is complete, the testing will begin. The FPGA has over 28K logic cells, so testing a reasonable number of SEU situations would take a very long time. In order to expedite the testing process, the experiments will be performed using an array of Zedboards, known the SoCDrawer. By running the tests in parallel, results can be gained much more quickly. This should be completed by the 14th of June.

5 Evaluation

Each of the two main deliverables of the project will be evaluated. This section will give details on how each deliverable will be tested, and how the success of each deliverable will be measured.

5.1 FPGA Implementation of a CNN

The testing and critical assessment embedded CNN implementation is crucial to the success of the project. If this cannot be completed, or is completed incorrectly, the rest of the project is jeopardised. If the CNN does not perform sufficiently well, the usefulness of the project's results is diminished. Therefore, the FPGA CNN implementation will be thoroughly tested and examined.

There are two main criteria that the success of this deliverable will be evaluated on. These are the performance, measured in Giga Operations per second (GOp/s), and the resource efficiency, measured in Giga Operations per second per slice (GOp/s/Slice). Slices are a measure of FPGA resources, and are implemented differently on different FPGA families, but on a Zynq 7000, a slice is defined as four Look Up Tables (LUTs), and their associated flip-flops, multiplexers and carry logic [18].

These criteria will be evaluated in comparison to results from literature. For example, Qui et al. implemented an FPGA accelerator with a overall performance of 136.97 GOP/s and a 2.61×10^{-3} GOP/s/Slice. This is a very good result and achieving these values would be ideal. Another example, using a Zynq 7000 system and a CNN model called fpgaConvNet, is that described by Venieris et al., which achieved 12.73 GOp/s and4.5×10^{-4} GOp/s/Slice[19]. Given that this is on the same platform that I will use, this is a more realistic result to aim for.

To confirm that the CNN has been successfully implemented, the classification accuracy will be tested. The accuracy of a CNN is typically measured as top-1 accuracy and top-5 accuracy, which give the proportion of the testing data for which the correct class was assigned the highest probability and was in the highest 5 assigned probabilities respectively. This accuracy will depend on which CNN model is implemented, but as an example, the full precision Caffe AlexNet model achieves a top-1 accuracy of 56.82% and a top-5 accuracy of 79.95% [4]. The accuracies achieved by the FPGA implementation will depend greatly on which model we choose to implement, but if we chose AlexNet for example, we would expect to achieve accuracies close to these results.

Other criteria that may be evaluated are the total execution time, the total resource efficiency, the total power consumption and the power efficiency (GOp/W). There are articles which provide these values for their implementations[8], against which I would compare my implementation's results.

5.2 Fault Tolerance Investigation

The other deliverable to be assessed is the SEU tolerance. A good measure for this is Failure In Time (FIT), which is the number of failures that can be expected in 10^9 hours of operation. This will be tested with the Xilinx SEM IP core and the SoCDrawer, as described in Section 4.3. This result will be analysed and used to evaluate the suitability of the system for space applications.

6 Conclusion

In this report, the outline of this project was presented. In Section 2, the target platform was defined, and important concepts concerning CNN structures, HLS tools, FPGA implementations of CNNs, and FPGA applications in space were briefly described. Section 3 gave the Project Specification, laying out the objectives of the project and what I expect to achieve. Section 4 provided a plan of how each part of the project will be implemented, and what work needs to be done to achieve these objectives. Finally Section 5 set out the metrics that will be used to determine whether each stage of the project has been successfully completed.

References

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybernetics 36*, 1980.

[2] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Computer Vision*, 2015.

[3] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of fpga high-level synthesis tools," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015.

[4] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. sun Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," *FPGA*, 2016.

[5] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang, and H. Yang, "Going deeper with embedded fpga platform for convolutional neural network," *FPGA*, 2016.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *NIPS*, 2012.

[7] Y.-L. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[8] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," *FPGA,*, 2015.

[9] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ASPLOS*, 2014.

[10] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," in *ISCAS*, 2010.

[11] F. Wang and V. D. Agrawal, "Single event upset: An embedded tutorial," *21st International Conference on VLSI Design*, 2008.

[12] S. Habinc, "Suitability of reprogrammable fpgas in space applications," *Gaisler Research*, 2002.

[13] S. Habinc, "Functional triple modular redundancy (ftmr) - vhdl design methodology for redundancy in combinatorial and sequential logic," *Gaisler Research*, 2002.

[14] L. Rockett, D. Patel, S. Danziger, B. Cronquist, and J. Wang, "Radiation hardened fpga technology for space applications," *EEEAC paper 1305*, 2006.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[16] B. Dutton, M. Ali, C. Stroud, and J. Sunwoo, "Embedded processor based fault injection and seu emulation for fpgas," *International Conf. on Embedded Systems and Applications*, 2009.

[17] Xilinx, *LogiCORE IP Soft Error Mitigation Controller v4.0*, 2013.

[18] Xilinx, *Zynq-7000 All Programmable SoC Overview*, 2016.

[19] S. I. Venieris and C.-S. Bouganis, "fpgaconvnet: A framework for mapping convolutional neural networks on fpgas," *IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines*, 2016.

[20] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *ICCD*, 2013.

[21] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPR Workshops*, 2014.

[22] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *ISCA*, 2010.

[23] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar, and H. P. Graf, "A programmable parallel accelerator for learning and classification," in *19th international conference on Parallel architectures and compilation techniques*, 2008.