CNN Architectures on Heterogeneous Systems

Alexandre Vieira, Aleksandar Ilic, Frederico Pratas, Leonel Sousa

***Abstract*—On-board computer vision algorithms present in modern cars are already capable of identifying multiple objects, and gauge road conditions. However, these are executed in complex computing devices that are not always optimal in terms of compute capability and energy efficiency. To achieve this, an algorithm based in resource constrained convolutional neural networks is proposed, and an hardware mapping methodology is developed to deploy the system in a reconfigurable hardware platform for further acceleration. A thorough study on neural network compression through parameter quantisation and architectural transformations is performed, in order to decrease its resource and execution time cost on a chosen FPGA device, while maintaining high classification accuracy figures. A comprehensive metric is devised to access the performance of a quantised and modified CNN, providing weighted parameters that may favour a desired network property. The preliminary results of the network deployment into a CPU+FPGA platform are presented, which show up to 50% execution time reduction when compared with a larger state-of-the-art implementation.**

Keywords: convolutional neural network, fpga, accelerator, opencl, autonomous car, driver assistance

I. Introduction

Self-driving cars are an emerging topic in several research communities, as it approaches many engineering fields. Cars are equipped with a wide range of sensors, such as cameras, Radio Detection and Ranging (RADAR), Light Detection and Ranging (LIDAR) and Sound Navigation and Ranging (SONAR), which when combined can map the surroundings of the vehicle. All this sensing technology is used to assist the driver in some particular situations. However, fast and correct acting on multiple events is needed, and to achieve real-time and reliable performance on cars, the computational power of the embedded systems used in Advanced Driver Assistance System (ADAS) has been increasing alongside their tasks' complexity. New algorithms for data analysis are always being developed and improved to better manage the vehicles.

Object recognition algorithms exist for many years, and recent developments have moved from more classic clustering algorithms to a more advanced kind of Machine Learning (ML) algorithm called Neural Networks (NNs). The latter are based on self-learning techniques, and thus very flexible types of algorithms that can be designed to perform various tasks. Currently, they are mostly designed to be processed in Graphics Processing Units (GPUs) and/or Central Processing Units (CPUs), but there is an effort to port them to efficient hardware accelerators in Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs).

Big companies are moving towards the usage and also construction of programming languages and development platforms related to ML, more specifically NNs. TensorFlow from Google, has had great developments in recent years, mainly due to the large developer community that frequently updates and improves the platform. It offers modularity and customisation that were key in experimenting with novel compression techniques to be applied on state-of-the-art Convolutional Neural Networks (CNNs), and thus study the impact of said techniques in decreasing computational requirements to hardware deployed networks.

The main goal of this work is to arrange a mapping technique for computationally compressed CNNs into hardware accelerators, to further be used in low-latency ADASs. For that we will need to:

- develop a CNNs algorithm to detect vehicles;
- study operation-level compression techniques for CNNs;
- optimise the CNNs so as to use the least memory and arithmetic operations as possible;
- transpose developed system into FPGAs in Intel's upcoming processor architecture.

To fulfil the objectives proposed, three main topics need to be extensively studied. These will lead to the success of the finalised system.

First, we propose an algorithm re-utilisation technique, reducing the total development time. Subsequently, in order to reduce the computational and storage needs for the proposed algorithm, data quantisation methods will be explored, reducing both the system memory requirements, and enabling the use of less powerful arithmetic units. This leads to a reduction in energy consumption and execution time, allowing the deployment into a reconfigurable hardware platform.

The remaining of this paper is structured as follows: Section II describes background subjects, important concepts for understanding the described work and introduces the state-of-the-art developments in Deep Learning (DL) research; Section III presents a high-level overview on the contributions and details the implementation process of the proposed solution for this work; Section IV explains in detail the performed work in the scope of this paper; Section V states the conclusions drawn from the performed work.

II. Background

Complex ADASs, like *Pre-Crash* and *Lane-Departure Warning*, require demanding computational analyses, including image/video processing. To accomplish this functionality, current research trends rely on ML derived algorithms to implement these functionalities, such as NNs and CNNs.

CNNs are an adaptation of NNs to multi-dimensional inputs, and are divided in several different layers. Because of the inherent image analyses affinity and their weight sharing property, CNNs are the de-facto standard for image classification and object recognition for the past years. There are several state-of-the-art CNN models that were proposed in the literature and/or developed in computer vision competitions,

such as ImageNet Large Scale Visual Recognition Challenge (ILSVRC). These include AlexNet, VGG-16, GoogLeNet-v1, and ResNet-50 CNN architectures, that have taken part on ILSVRC in the years of 2012, 2014 and 2015.

From the previously mentioned models, AlexNet [1] network was the first CNN to significantly outperform the common image classification methods. Motivated by this achievements, several different CNNs have been developed, with an evident trend of increasing the number of layers, filters and their sizes. Nonetheless, AlexNet still remains a standard to which many CNN studies compare their performance to.

Efforts to resist increased computational, memory and energy usage costs on CNN inference have been reported in recent bibliography. These methods target to build an equally accurate network architecture, while requiring either less memory, fewer (or less expensive) operations, or both. A widely reported cost reduction method is parameter quantisation, which consists of using lower precision data types for storing each individual parameter. A lower number of bits means a smaller memory footprint, more efficient arithmetic units, lower latency, higher memory bandwidth, but also that the number of different values that can be represented is lower, potentially affecting the CNN's accuracy.

Common bit-widths for quantised values are 12, 10, 8 and 5 bits, however, recent investigation also leans towards weight binarisation, *i.e.*, weights with a 1 bit value. This drastically decreases forward-propagation times, as the hardware accelerator (usually FPGAs) can work at a higher clock speed, since they are not limited by the typically lower performance of other Arithmetic Logic Unit (ALU) operations. A fully binarised network is called Binarised Neural Network (BNN).

Improvements on BNNs can be achieved if the network is retrained with the quantised weights, sometimes re-achieving the original performance. Results have shown up to 98% accuracy on a LeNet-5 BNNs, meaning that BNNs have the potential to redefine the NN paradigm [2].

In a BNN, the network architecture is slightly different from a common CNN. In Figure 1 an example architecture is presented side-by-side with a common CNN. Its differences go as follows:

- Convolution - convolutional filters' weights are quantised to 1 bit, storing either -1 or +1;
- Intermediate values - feature maps resulting from convolutions are stored in 12 bit integer values during the following operations;
- Biases - the addition of biases is omitted because;
- Normalisation - a new method, Batch Normalisation (BatchNorm), is introduced and consists of linearly rescaling feature map values, such that the output distribution has zero mean and unit variance.

A. Software Tools for Machine Learning

With the increasing advances in micro-processor architectures and manufacturing processes, General Purpose Computer (GPC) platforms have become more affordable and powerful, while power-efficiency increases. Today GPUs are a popular

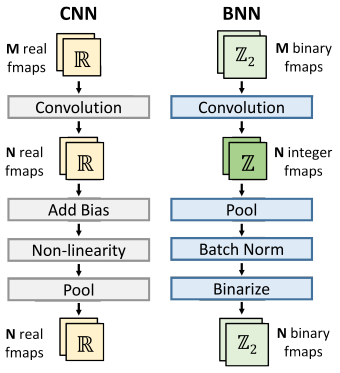


Fig. 1: BNN general architecture (adapted [3]).

choice for CNN training as they are equipped with fast arithmetic units that efficiently perform matrix operations using floating point numbers, and can operate on multiple processing threads concurrently.

To take advantage of the powerful hardware on GPUs, specific software tools have to be used, so that the correct instructions are given to the device and thus efficiently use the dedicated computing units within the GPU's silicon. These software are called *DL frameworks*, and the work herein described will cover:

- CUDA-convnet [4], developed using a Compute Unified Device Architecture (CUDA) backend, widely used in benchmark tests, as it offers great performance;
- TensorFlow [5], as it exposes complex tensor operations to the user, enabling the development of a completely customisable model.

B. Hardware Implementation (CPU+FPGA)

DL algorithms can also be accelerated by pairing an hardware accelerator with a CPU, offloading the management and control tasks to the GPC CPU, while complex mathematical data operations are running on the accelerator. This accelerator can be custom designed to meet the task's requirements, improving its compute capabilities and/or energy efficiency. Development boards populated with an FPGA+CPU System on Chip (SoC) exist in many varieties, and typically consist of an ARM processor in conjunction with a top-of-the-line FPGA chip to allow for Hardware-Software co-processing. These designs are promising since each component executes tasks that are best fitted for its architecture.

To deploy accelerators in FPGAs, Hardware Description Language (HDL) paradigm is used, with the goal of designing a physical hardware device. HDLs' compilation results into a set of instructions that describe the connections between integrated support blocks, so that the described hardware can be deployed onto, for example, an FPGA. Big companies in the reprogrammable hardware market have launched their own approaches to the High Level Synthesis (HLS) paradigm, with solutions from Maxeler that uses Java, Xilinx who works with C/C++ constructs, and Altera that has adopted Open Computing Language (OpenCL) [6].

OpenCL is designed to provide a unified framework for heterogeneous computing platforms, and enhances the code

portability and programmability of different GPC devices, at the expense of optimisation granularity. It has become a popular alternative tool for heterogeneous hardware programming because it is open source and royalty-free, as opposed to CUDA.

Currently there exist no abundant solutions to easily implement CNN models in FPGA devices using OpenCL. At the timing of writing, PipeCNN Wang et al. [7] is the only open source OpenCL FPGA-CNN inference accelerator, and consists of system composed of different OpenCL kernels, each performing one of the layer components in generic CNN architectures. There are 5 main blocks which represent the OpenCL kernels: 1) Memory Read, 2) Convolution, 3) Local Response Normalisation (LRN), 4) Max Pooling, and 5) Memory Write. To perform the inference process, the accelerator needs multiple streams of data, which are loaded by the Memory Read kernel, that outputs it into the inter-kernel memory channels. The Convolution kernel is engaged, and its results are streamed into either the memory channels that connect to the LRN kernel, Max Pooling kernel, or directly to the Memory Write kernel. Once in the Memory Write kernel, data is prepared for the next accelerator iteration and this process is repeated as many times as the number of layers in the model.

Moreover, Intel's upcoming platforms will consist of a CPU accompanied by an FPGA in the same package [8] that supports OpenCL programming, which will enable a deep integration of reconfigurable hardware in a GPC platform.

III. Solution Design - Hardware Accelerated CNNs

This work focuses on the analyses and implementation of a CNN algorithm, which is envisioned as part of a larger platform system, with the goal to detect, differentiate and possibly identify individual vehicles. In particular, the envisioned system to be implemented and used in the vehicle receives as input the images of the vehicle's surroundings, and from those images detect whether there are any other vehicles in the proximity.

Before the deployment on the final hardware platform, the chosen CNN architecture will undergo some optimisation transformations, namely: 1) classification layer adaptation, 2) architecture resource optimisation, and 3) migration to hardware accelerator.

A. Network Domain Tuning

To decrease network training times, using pre-trained CNNs for further specialisation is becoming a common practise. This is feasible since the pre-trained CNN, is already capable of identifying image features that, when grouped, can classify a particular object. Moreover, by using a different classifier layer and retraining the network, the training process is much faster, since the high-level features are already learned. Recent bibliography names these networks "multi-domain networks".

The work herein developed uses network domain tuning optimisation on a CNN algorithm to further improve its identification performance on the particular task of identifying vehicles. By selecting a collection of target objects for the identification algorithm, the initial network model can be adapted to identify particular vehicle types in order to inform the main ADAS solution and act accordingly. With the smaller number of end results, and because only the last fully-connected layer is changed to have less outputs, the network performance should increase in terms of accuracy.

For the specialisation of the AlexNet CNN architecture, a reduction in the number of output categories needs to be performed. As such, the vehicle types that the network shall be trained to detect were chosen to be "cars", "motorcycles" and "trucks", as they represent the main vehicle types found in roads. A correspondence is made to convert the original dataset labels into the new smaller set of categories.

B. Network Cost Reduction

Because CNNs involve highly parallelisable computations, they are a fit to current fine grain programmable logic devices. However, because Digital Signal Processor (DSP) units remain a limited resource in FPGAs, and data routing paths are constrained, using floating point data-types to represent network parameters is very costly and requires many logic elements. FPGAs provide orders of magnitude in efficiency improvement over software, while enabling network reconfiguration and without having to lock into a fixed hardware design, as happens in ASIC solutions. That said, network quantisation methods need to be studied, in particularly the emerging BNNs paradigm, as it offers the most benefits in reducing hardware resources for a network model of the same size.

The standard AlexNet CNN architecture was developed to be used with floating point variables, in both training and inference, and thus quantisation transformations must be applied. However, since network accuracy figures are expected to be very disparate between both the original network architecture and the altered version, a pseudo-metric for evaluating their comparative performance is introduced. The performance penalisation P_i is obtained by dividing the accuracy acc_i of each i quantised network, by the accuracy acc_N of the original network. The memory footprint improvement factor M_i is calculated by the ratio of the used memory between the original network's size in bits n_N by the altered network configuration n_i. Note that the performance difference is penalised quadratically according to

$$\mathrm{performance} = \left(\frac{\mathrm{acc}_i}{\mathrm{acc}_N}\right)^2 \times \frac{n_N}{n_i} \tag{1}$$

because the goal is to find the network that occupies less memory while maintaining high accuracy figures.

To improve the modelling studies, several quantisation steps need to be tested to assess if full parameter binarisation will allow for good performance. Also, and to minimise the impact of the loss of precision by the quantisation, several architectural modifications are proposed and tested, targeting a balance between accuracy and performance cost.

The aforementioned architectural changes were envisioned to mimic similar bandwidth requirements, such that a network

with a smaller quantisation step may have the same bandwidth as the immediately larger quantisation step. These changes act on the following parameters of the network:

- filter/neuron count - with an increasing number of filters and/or neurons, the network gains the ability to learn more feature types;
- filter size - larger filters enable the network to learn and distinguish larger features in the input image;
- convolutional layer addition - improves the network ability to learn higher level features.

Combinations of the above changes were applied to CNN models, resulting in the proposed 8 different types of new network architectures:

- Half - the network is synthesised with half the number of output maps in each convolution layer and half the number of neurons in each hidden fully-connected layer;
- Double - the network is synthesised with double the number of output maps in each convolution layer and double the number of neurons in each hidden fully-connected layer;
- Size - the input image size is doubled with a bilinear extrapolation algorithm (when needed), the size of the first convolutional filters is doubled, and the pooling window of the last pooling layer also doubles;
- Extra - an extra convolutional layer is added with the same number of channels and filter sizes as the last convolutional layer already in the model;
- Half/Size - combination of Half and Size, *i.e.*, all layers have half the number of output channels (either maps or neurons, depending on the layer type) and the input image size is increased with the necessary adaptations to the first and last convolutional layers;
- Half/Extra - combination of Half and Extra, *i.e.*, all layers have half the number of output channels and there is an extra convolutional layer with the same configurations as the previous one;
- Double/Size - combination of Double and Size, *i.e.*, all layers have double the number of output channels and the input image size is increased with the necessary adaptations to the network;
- Double/Extra - combination of Double and Extra, *i.e.*, all layers have double the number of output channels and there is an extra convolutional layer;

With all the incremental transformations aforementioned, there are increasing computational requirements, larger memory size, and more data transfers. In GPC platforms, however, it is difficult to predict exactly how either Floating-Point Operation (FLOP) and memory access counts will be affected, as this is extremely dependent on the framework implementation and its optimisations, and also on the hardware properties. With that said, in CNN inference hardware implementations, the impact of these transformations in the needed resources is also implementation dependent, with a different impact (if any) on each particular implementation paradigm.

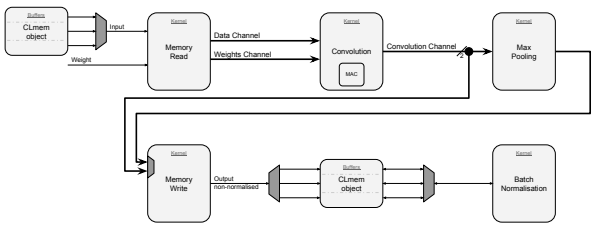


Fig. 2: OpenCL BNN accelerator architecture.

C. BNNs in Dedicated Hardware

The use of OpenCL programming language enables high level abstraction, thus focus may be directed to the design and construction of an accelerator architecture capable of the necessary reconfigurability to test the previous hypothesis referenced in Section III-B. The PipeCNN project is not locked into any specific library, thus becomes very adaptable and can be improved/changed to meet the requirements of this work, *i.e.*, the accelerator can be adapted to execute quantised BNNs, thus achieving a deployment platform for the proposed work.

In FPGAs the available hardware is limited, thus a smart resource management needs to be engineered to fit the necessary structures into the device to perform the desired task. For this reason, resource reuse through cycling computational structures will be used, as well as data vectorisation to take advantage of the parallelisation capabilities of FPGAs. The PipeCNN accelerator already uses these techniques, and a diagram with the new architecture can be seen in Figure 2.

The datapath within the accelerator is changed from floating point to Arbitrary Precision Integer (AP-Int), so as to guarantee that integer operations are performed. The AP-Int library constructs arithmetic units, logic and data lanes with the precision specified by the user. With the use of these integer values, the arithmetic units within the FPGA chip will be synthesised faster integrated integer multipliers/adders, or hardware Look-Up Tables (LUTs).

Because this platform is intended for general BNN use and to be network architecture independent, both the architecture parameters and the data bit widths were made user configurable. This way, the accelerator fully supports all aforementioned architectural transformations with no need for re-synthesis. This means that there is no impact on hardware resources, and consequently the synthesised FPGA size. However, execution times and memory size/transfers will vary with each applied transformation.

IV. Experimental Evaluation

To evaluate and develop all the different propositions in this work, several tests were performed with different software tools for each purpose. During the different development phases, multiple combinations of software and hardware platforms were chosen as the most fitting for the required task. For the architectural tests of the network, mainly GPUs and NN frameworks were used, while for the final hardware architecture development, an OpenCL Software Development Kit (SDK) and an FPGA device were the main tools.

TABLE I: LeNet-5 error-rate for different architectures.

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	0.59	0.63	0.57	**0.48**	0.64	0.52	0.61	0.69	88.65
Qts. 5 bits	0.87	0.83	0.82	0.83	**0.71**	**0.71**	0.82	0.78	0.79
Qts. 3 bits	0.94	0.88	0.83	0.78	**0.71**	0.80	0.86	0.78	0.82
Qts. 2 bits	1.08	1.27	0.88	1.04	1.19	0.91	0.95	**0.82**	0.97
Qts. 1 bit	2.05	3.92	1.26	**1.08**	2.01	1.67	5.82	1.13	1.24

TABLE II: CIFAR-10 error-rate for different architectures.

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	15.29	18.30	**14.17**	18.53	15.85	21.65	18.75	16.74	15.07
Qts. 5 bits	20.09	21.76	20.20	**18.19**	19.98	20.09	21.21	18.42	19.87
Qts. 3 bits	29.80	31.14	33.15	**25.00**	28.35	25.67	29.58	25.33	27.90
Qts. 2 bits	38.50	39.40	47.43	**30.92**	38.50	32.25	39.06	31.03	39.73
Qts. 1 bit	55.69	61.27	49.44	38.95	47.54	48.21	55.25	**34.82**	43.42

A. GPU Results

The first step in the development of the network architecture is to test the vanilla version of the AlexNet model. These tests were performed using the CUDA-convnet tool and the standard dataset for the ILSVRC 2012 competition. Results of the network training revealed that it is possible to achieve a highly accurate object classification. The final classification error is lower than 30% on the training images, and 42% on the test images.

Results for the reduced number of dataset categories were also obtained with the CUDA-convnet tool. Here, the classification error significantly decreases in both training and testing sets, being lower than 1% and 2%, respectively. However, due to the test-set not being composed of a large enough sample size for the 3 desired categories, the network overfits to the training dataset, performing worse on the validation inputs with the increasing training steps.

B. Quantisation Cross-Comparison

When faced with the challenge to perform tests with BNNs, CUDA-convnet revealed to be of closed architectural construction, making TensorFlow a much friendlier platform to work on in this respect, because of its open architecture description, allowing the developer to fully describe the operations and data-path with discrete functions for each layer type. This flexibility enables custom data operations in-between the data layers, as opposed to other development platforms, and thus an algorithm to reduce the precision of the data can be introduced to emulate the quantisation of the values.

To test the quantisation effect on CNNs with TensorFlow, DoReFa's code [9] for weight quantisation was used, as it provided a TensorFlow compatible structure to apply flexible quantisation functions given a chosen bit width. Experiments were performed initially with LeNet-5, a small CNN model trained for the MNIST database. Results show that with our quantisation functions it is possible to replicate previous works [10], proving that a BNN can perform well on image classification tasks.

Table I summarises the experimental accuracy results of all network transformations (lower is better). The best performing transformation in each bit-width is highlighted in bold for convenience. It is obvious that there is no "absolute winner" in terms of accuracy, and that results are inconclusive as to what transformation has better impact in improving the accuracy of a quantised Lenet-5 network. There is, however, a slight bias towards the "Half/Size" transformation, as its accuracy figures are more consistent than the remaining ones.

Results show that all 3 modification types and their combinations do affect the network performance, though their influence varies with the quantisation applied, *i.e.*, while one type of alteration improves performance for 1 bit quantisation, it worsens on 2 and 3 bit. For example, the "Extra" transformation is expected to improve accuracy by adding another layer to store and identify higher-level image features. However, while it has a positive impact on 3 bit quantisation, it has close to no effect on both 2 bit and 1 bit.

An homologous comparison was made to evaluate the performance of networks with similar memory requirements. Overall, results show that single transformations do improve a specific bit width result, but fail to match and/or outperform the immediately higher tested bit width. When results for 1 bit transformations are compared with their homologous higher bit-width quantisations, the latter overcome any of the 1 bit transformations. The same kind of observation can be made for 2 bit transformations, as the higher bit width quantisations always match or surpass the equivalent lower bit width ones in terms of accuracy.

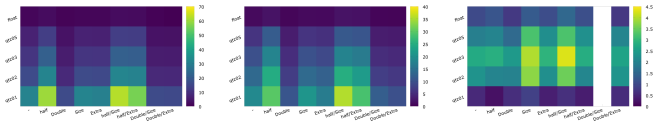
To validate the previous observations, the network model for the CIFAR-10 dataset [11] was tested. Table II summarises the obtained results, and unlike the results for LeNet-5, here it is possible to observe that the "Size" transformation has the majority of best results. This is likely to be related to the nature of the CNN's task, as for LeNet-5 the input images are monochromatic and have simple structures, while the CIFAR-10 dataset is composed of object images with complex and colourful patterns. Hence larger CNN filters tend to boost its accuracy. Nonetheless, the incoherence observed in the Lenet-5 results also applies to CIFAR-10's, as the impact of each transformation is not equal throughout the quantisation levels. For example, "Double/Extra" improves results in all quantisation levels except 2 bits, while "Double/Size" improves results on all but the original floating point implementation.

After testing the network modifications and benchmarking the performance of smaller networks (LeNet-5 and CIFAR-10), these tests were repeated on the AlexNet CNN, and the validation results can be observed in Table III. Once again the accuracy results do not match any of the previously tested networks. This time, the best performing transformation is "Double/Extra" by being the best result in 4 out of 5 parcels, and "Double" taking the 5th. Unlike what was seen in CIFAR-10, this time the preferable model parameter increase is the number of output maps, which indicates that the vanilla AlexNet network model is lacking filter space to learn all the necessary input features. This is inline with what was discussed in Section II, as all newer network models feature a larger number of filters per layer.

Note that these results show accuracy levels in baselines much lower than those of LeNet-5 and CIFAR-10, as result

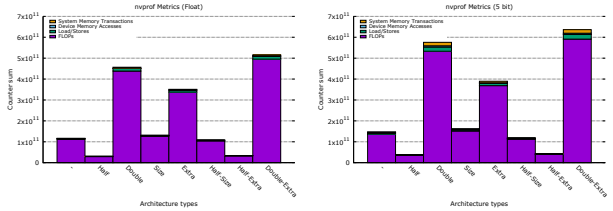
TABLE III: AlexNet error-rate for different architectures.

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	59.23	68.83	50.49	53.77	58.17	64.65	69.84	-	**50.36**
Qts. 5 bits	77.62	83.81	71.78	71.06	75.85	79.50	82.82	-	**67.27**
Qts. 3 bits	79.32	85.15	72.57	75.10	78.53	81.75	85.12	-	**71.27**
Qts. 2 bits	84.54	89.73	79.73	80.31	84.69	86.49	89.59	-	**77.86**
Qts. 1 bit	94.82	97.75	**92.22**	93.30	95.56	96.03	96.96	-	92.37

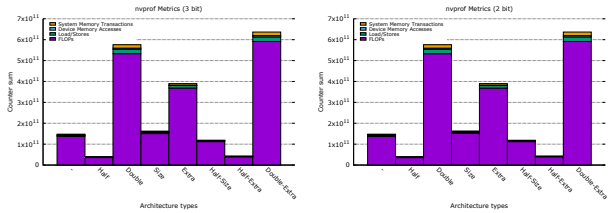


(a) LeNet-5 (b) CIFAR-10 (c) AlexNet

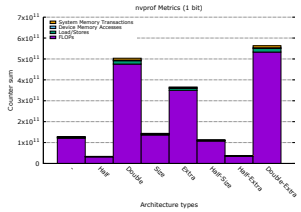
Fig. 3: Graphical representation of PSM.



(a) Float transformations (b) 5 bits transformations



(c) 3 bits transformations (d) 2 bits transformations



(e) 1 bit transformations

Fig. 4: AlexNet profiling figures for different architectures.

of the short training time and the great difference between the complexity of those networks. Due to limitations in GPU memory size, it was not possible to acquire results for the "Double/Size" architecture with TensorFlow, thus those results in Table III are marked as "—".

Since the objective is to minimise the memory footprint of the model to be deployed, the memory requirements for each network and their variants were calculated. Based on each of the networks' configurations, the size in bits of the network's parameters was measured, and using Equation (1) the performance metric was calculated for all 3 networks. A graphical representation can be seen in Figure 3 (lighter is better).

It is noticeable that for the 2 smaller networks the performance pattern is very similar for the used transformations. This shows that despite the best transformation in each network not being the same, and accuracy-wise the results not being consistent, when observed from an overall performance perspective, the results do match. In both cases, the transformation with the higher performance metric is the 1 bit quantisation with "Half/Size" transformation.

However, the larger AlexNet network's accuracy is highly affected with quantisation, and therefore the performance results are worse on the smaller precision variants. This is due to the quadratic accuracy penalisation, which impedes transformations such as "Double", which produces the higher accuracy values across all transformation variants, from overcoming the memory footprint increase.

In order to acquire the total number of operations needed on GPU platforms in order to infer a single input image, a full validation cycle was performed under *nvprof* execution, for every test mentioned above. To evaluate the performance of the network, several metrics must be analysed, that reflect the physical requirements for a particular device to run the inference engine. Figure 4 shows the histogram of the number of floating point operations, device cache and memory accesses, as well as system memory accesses for each run, all normalised to a single input image. These results were obtained on a host machine equipped with a single NVIDIA GTX 980.

The histogram trends presented in Figure 4 are explainable by analysing the quantisation code used in these studies (derived from the work of Zhou et al. [9], as mentioned in the previous sections). The plots show a similar pattern for all the parameter bit widths, meaning that the coded architectural changes equally affect the network models in terms of computational requirements, which is to expect. Moreover, for all the analysed metrics, the "Half" transformation is the less computationally expensive across all quantisation bit widths. This is expected as it is effectively the network transformation with the least number of parameters, and thus less needed data operations. However, when analysing the metric count differences between different bit widths, it is noticeable that 5, 3 and 2 have very similar magnitudes. This is due the fact that all these require the same number of operations to be quantised in the quantisation functions, whereas 1 bit uses a different (and less expensive) quantisation method, and thus appears lower on the plots. Moreover, the fact that the floating-point implementations have lower counter values than the quantised ones is also expected, as the TensorFlow model does not apply any quantisation functions to its execution.

Furthermore, the "Half-Extra" transformation has an unexpected behaviour. Results show that is has a lower impact than "Half-Size", contradicting the rationale of their single transformations, *i.e.*, if "Size" is lower than "Extra" when used alone, then "Half-Size" should be lower than "Half-Extra". The initial speculation presented in Section III stated that the application of multiple transformations should be cumulative, thus applying "Half" should equally affect both. This difference can only be explained by aggressive optimisation-dependant execution changes by the TensorFlow and/or CUDA software,

as the models are correctly implemented and the accuracy results are coherent. For these reasons, the operation count values herein presented are not usable for final conclusions on hardware computational requirements, as they were obtained on GPC platforms that vary their execution order and resource utilisation based on many different factors.

C. Hardware Accelerator Performance

With the system modelling finished and a test architecture chosen, the hardware tests were prepared. For that, a set of TensorFlow network trainings was performed with modifications to the network model in order to enable the gathering of all necessary parameters and the intermediate values of the network, for further comparison against the outputs from the OpenCL accelerator. The layer weights are obtained directly via the save points of the TensorFlow software, however, the BatchNorm auxiliary values are not simple to be extracted from the model. Because the BatchNorm layer is part of the TensorFlow internal functions, it is designed to store those values in private variables that are also retrieved upon session restore, but are never user accessible. As such, the network model was changed to use a custom wrapper that we developed, which manages the normalisation variables and allows easy data retrieval without compromising functionality.

For the tested bitwidths, the accelerator's outputs and intermediate results exactly match the outputs gathered from the test TensorFlow script, *i.e.*, the raw outputs from the last layer are an exact match to those of TensorFlow, which shows that the accelerator is mathematically correct and does indeed emulate a convolution and dot product engine.

As the accelerator works on quantised data, naturally there is a direct impact on the memory space needed to store the network parameters, as discussed before. However, care should be taken on some key details regarding the FPGA implementation. Though there is an OpenCL library available that uses shorter bit width data paths and arithmetic units for the AP-Int variables, this precision reduction is not transported to device's memory buffers. The reason is that memory structures available to the FPGA are still byte addressable, and as such the variables are stored in byte multiples. Moreover, though the network weights are integers, the BatchNorm's parameters are still float values, so those sizes must be taken into account.

For the particular case of our accelerator, which was envisioned for bitwidths lower than 8, the memory savings are shown in Table IV. For both networks, the input and weight numbers are exactly the same, since the input images are the same size and have the same number of colour channels, and the network filters are of the same size and depth (if no transformations are applied), so the only difference comes with precision in these cases. However, because we are using different CNN paradigms, in the first floating point CNN there exists the addition of a bias at the end of each convolution/dot-product, and no need for normalisation parameters (LRN only relies on the output feature maps' values to normalise them). On the other hand, the quantised BNN has no bias addition and needs floating point values for the normalisation kernel. In the end, and using single byte blocks to store each value, there is a space saving of $1 - 59.79/238.54 = 75\%$ by using BNNs, which is very significant. This difference will be even more pronounced for networks with a greater number of layers and also with larger layers, but not for architectures with larger layer depths, as the BNN normalisation parameters only grow with the number of output channels of the layers, and not with their size.

TABLE IV: Memory footprint comparison.

	Inputs + Weights	Biases	Normalisation	Total (MiB)
Float CNN	62 522 363	10 568	-	238.54
≤8 bit BNN	62 522 363	-	42 272	59.79

TABLE V: Synthesised accelerator resource utilisation (%).

	BNN [8/3]	BNN [8/6]	BNN [16/3]	PipeCNN
Logic Utilisation	**74**	82	80	78
ALUTs	**32**	35	34	33
Registers	**43**	48	47	46
Memory Blocks	**44**	51	50	60
DSP Blocks	50	50	50	**34**

Though memory savings are not as pronounced as desired when it comes to memory buffers, the synthesised hardware uses lesser data lines to represent the AP-Int data types, thus reducing the logic utilisation on the device in the routing and integer arithmetic units. However, because the CPU+FPGA Board Support Package (BSP) is only compatible with version 16 of the SDK, this library is not available, leading to hardware utilisation reports higher than the minimum achievable.

To facilitate device's buffer management, all buffers are created with the same size, such that every buffer has the necessary space to accommodate the largest memory string. As such, in the particular case of the tested implementation, 44 buffers are created in the device, for a total of $90\,725\,408\,\mathrm{B} \approx 86.52\,\mathrm{MiB}$

D. Deployment Results

The last set of experiments performed as part of this work consist in synthesising the accelerator designed and evaluated in the previous section, for deployment in a reconfigurable hardware platform. The hardware was synthesised with and without the *–profiler* SDK flag, which enables the generation of auxiliary profiling hardware. During the synthesis process, the SDK outputs the internal resource utilisation estimation, which can be seen in Table V. Several hardware configurations are shown, for the different hardware designs that were both generated and benchmarked. From left to right, the first design corresponds to the first successful configuration to work both on simulation and on the FPGA. The next two columns refer to devices synthesised later to gather more architecture performance data points, the second being 8 bits with a size-generic pooling kernel (as opposed to PipeCNN's fixed size), and the third using 16 bits variables with the original pooling kernel, because using 16 bits alongside the size-generic kernel resulted in a design that doesn't fit the available FPGA resources. The last column refers to the reference PipeCNN design.

The FPGA binaries were transferred to the off-site Intel servers, and the system was able to successfully run the

developed accelerator. The complete runtime of the host code (including the accelerator-only time frame) was obtained by using the *time* command on the Linux Operating System (OS), and is equal to 7.81 seconds, while the accelerator occupies only 0.84 seconds (average), according to the OpenCL event profiling functions. This represents an improvement when compared to the 8.09 that the TensorFlow Python script takes to evaluate the same amount of images, with 1.68 seconds being dedicated to the actual graph execution. This represents a $1 - (0.84/1.68) = 50\%$ reduction in execution time for multiple calls to the evaluation portion of the accelerator, as would happen in the real system for continuous image analysis.

In comparison with the original PipeCNN code, its kernel execution time is slightly shorter, finishing 0.66 seconds after launch. This proves that there is indeed a performance hit by having to wait for the BatchNorm kernel to finish, because execution times of the PipeCNN's normalisation kernel are 5 to 10 times shorter than those of our implementation of BatchNorm. However, when analysing the complete runtime of the application, PipeCNN takes longer than our implementation by occupying 8.06 of processing time, which is on a par with the TensorFlow application, and thus slightly higher than our hardware implementation.

In order not to have a single analysis data point of comparison between the original and quantised accelerators, a test battery was run on multiple configurations of both host and device parameters, *i.e.*, in the original accelerator multiple CNN architectures that corresponded to the network transformations mentioned in Section IV-B were tested, while on the new accelerator also the different data bitwidths were tested. However, it was not possible to complete these tests, because of some limitations in both designs the PipeCNN design and the extra cost due to SDK's generated profiling hardware.

All execution times can be consulted in Table VI, and they confirm the claims made in Section III on how the network architecture modifications should affect the network performance. Because the particular accelerator used in the study, as mentioned before, was compiled without the AP-Int library, the generated hardware for each reported bit width is similar in everything but the hardware integer limiters for the conversion between the intermediate 12 bit registers and the output results. This leads to similar execution times for every bitwidth within each transformation. When we analyse the execution time for bit widths smaller or equal than 8 bits, the column-wise standard deviation is no larger than 0.22 seconds, accounting for 1.4% of the largest execution time, contrasting with the inter-transformation (row-wise) standard deviation of over 4.5 seconds (30%).

For the 16 bits tests, the accelerator had to be compiled without the pooling kernel alteration, because the SDK was not able to fit the larger hardware in the FPGA due to the larger data-path. As such, the results for transformations that include "Size" are absent. The same applies for the PipeCNN tests, as the original design did not contemplate pooling layers with windows larger than 3 pixels wide. It is possible to see, however, that the difference between the average execution time for lower than or equal to 8 bits is similar to that of 16 bits, which means that the SDK generates similar hardware for both designs.

TABLE VI: AlexNet execution times on FPGA (second).

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	0.93	**0.28**	3.79	-	0.94	-	0.29	-	3.85
Qts. 16 bits	0.88	**0.27**	3.33	-	0.89	-	0.28	-	3.39
Qts. 8 bits	0.84	**0.26**	3.05	5.26	0.86	2.03	0.27	15.47	3.10
Qts. 5 bits	0.83	**0.26**	3.01	5.19	0.84	2.00	0.27	15.25	3.06
Qts. 3 bits	0.86	**0.27**	3.11	5.37	0.87	2.07	**0.27**	15.78	3.17
Qts. 2 bits	0.86	**0.27**	3.11	5.37	0.87	2.07	**0.27**	15.77	3.17
Qts. 1 bit	0.83	**0.26**	3.02	5.21	0.85	2.01	**0.26**	15.31	3.08

TABLE VII: AlexNet allocated memory on FPGA (MiB).

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
PipeCNN	755	**580**	1437	-	757	-	580	-	1446
BNN [16]	173	**85**	514	-	174	-	86	-	519
BNN [8]	87	**43**	257	167	87	123	43	338	260

Memory allocation was calculated on the host application, and results can be seen in Table VII. Following the previous rationale, implementations for lower than or equal to 8 bits are similar in hardware, and because host and device memory are byte addressed, resource occupation is the same for all those implementations.

Memory bandwidth figures were extracted from the profiling Graphical User Interface (GUI), and results are shown in Table VIII. Column wise it is possible to see that, within each transformation, the bandwidth is not consistent, and when looking at average numbers there exists a large difference between transformations including "Half", which has a higher average figure. As this transformation requires much less memory to be used during operation, there exists the possibility that the device's cache structures are much more effective at managing memory usage in these cases, and as such improving bandwidth values. This should be the main factor contributing to this increase, as the accelerator hardware remains the same across tests and the same happens to its clock frequency.

For 16 bit implementations, bandwidth is higher than 8 bit and below. This is mainly due to a larger data-path bus, thus increasing the amount of data transferred per memory operation. This doesn't necessarily mean that the accelerator works faster, as seen by the very similar timing results.

With these values collected, there is the need to compare the different implementations to evaluate which one is better, minding the benefits and drawbacks that each imposes on the system. For this, a metric was devised that takes into account all relevant aspects of code execution on reconfigurable hardware. This time, instead of cross-comparing implementations, the metric should take into account only the overhaul performance of the network inference in real-world applications, *i.e.*, how well would the accelerator perform if used in a real system. For that, the metric

$$\mathrm{CNN}_{\mathrm{pm}} = \frac{\text{batch size} \times \text{accuracy}}{\text{time}} \times \frac{\text{bandwidth}}{\text{memory footprint}} \quad (2)$$

is presented, considering the main aspects previously explained. Its result is presented in $\text{correct images}/\text{second}^2$.

TABLE VIII: FPGA memory bandwidth ($\mathrm{MiB/s}$).

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	751	415	**936**	-	750	-	415	-	935
Qts. 16 bits	746	823	781	-	774	-	**944**	-	784
Qts. 8 bits	438	685	408	-	438	-	**792**	-	427
Qts. 5 bits	487	**800**	455	-	468	-	701	-	452
Qts. 3 bits	481	645	429	-	473	-	**725**	-	452
Qts. 2 bits	476	**675**	453	-	495	-	648	-	425
Qts. 1 bit	522	720	466	-	477	-	**799**	-	459

TABLE IX: AlexNet performance on FPGA ($\mathrm{CNN_{wpm}}$).

	-/-	H/-	D/-	-/S	-/E	H/S	H/E	D/S	D/E
Float	0.493	0.394	0.172	-	**0.534**	-	0.339	-	0.169
Qts. 16 bits	0.205	0.405	0.048	-	0.281	-	**0.576**	-	0.085
Qts. 8 bits	0.251	0.698	0.055	-	0.332	-	**1.000**	-	0.101
Qts. 5 bits	0.283	0.822	0.062	-	0.359	-	**0.891**	-	0.108
Qts. 3 bits	0.197	0.456	0.050	-	0.219	-	**0.506**	-	0.062
Qts. 2 bits	0.061	**0.109**	0.016	-	0.059	-	0.108	-	0.021
Qts. 1 bit	**0.001**	0.000	0.000	-	0.000	-	**0.001**	-	0.000

This new metric calculates the number of accurately classified frames per second by taking into account the batch size, network accuracy and execution time, and scales it by multiplying a measure of memory stress given by the memory occupancy and data bandwidth, thus giving a perspective of the implementation performance on both identification accuracy and memory utilisation. These values are then normalised to the maximum of the batch for easier interpretation.

Intermediate results were calculated using the formula's output using the execution data points collected on the FPGA and the accuracy values previously obtained using TensorFlow. It is possible to remark that the best result, according to this metric, is the Quantised 8 bits with "Half/Extra" transformations, closely followed by Quantised 5 bits with "Half". These represent a mid-term between the loss in accuracy due to the quantisation of the parameters, and the gains in speed and memory footprint. Also, according to this metric, Float implementations present very poor results due to their large memory requirements.

This formula, however, is not suitable for performance evaluation on every application. For example, many applications may prefer (or even require) network accuracy over system requirements. To accomplish this, weight factors can be added to the formula so that one or more metrics have a larger impact in the formula's output, to fit the model to the specific need. The weighted performance metric

$$\mathrm{CNN_{wpm}} = \frac{\text{batch size} \times \text{accuracy}^{x_1}}{\text{time}^{x_2}} \times \frac{\text{bandwidth}^{x_3}}{\text{memory footprint}^{x_4}} \tag{3}$$

uses exponential factors to increase (higher x_i) or decrease (lower x_i) the selected metric's impact on $\mathrm{CNN_{wpm}}$.

As many applications need strong inference accuracy values to correctly implement their functionality, new values were calculated using $x_1 = 4, x_2 = 1, x_3 = 1, x_4 = 1$ to give higher relevance to the accuracy metric. Results are shown in Table IX. Once again, the 2 top results remain the same configurations, however, an higher classification is given to the Float configurations as they are the holders of the higher accuracy values.

V. Conclusions

Our work shows different methods to manipulate CNN architectures in order to vary their performance both in terms of accuracy and resource footprint.

With these tests it was possible to conclude that the impact of a particular network characteristic is highly dependent on the purpose and dimension of the original network. Moreover, when quantisation is applied to the network parameters, the impact of the referenced changes also varies, as the network looses precision in the weight parameters, and to counterbalance this disadvantage, the model either benefits from more filters or larger ones, depending on the network size, as shown by our results.

An hardware accelerator for quantised CNNs was developed using the OpenCL language. The accelerator delivers the same accuracy results when compared to a TensorFlow software implementation of the same network model, which means that the hardware architecture is a faithful representation of the developed software data flow. Also, by using a specialised hardware accelerator, execution time savings are achieved in the orders of 50%, which is very significant in a development platform.

References

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

[2] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[3] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating binarized convolutional neural networks with software-programmable fpgas. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 15–24. ACM, 2017.

[4] Alex Krizhevsky. cuda-convnet - fast c++/cuda implementation of nns. https://github.com/akrizhevsky/cuda-convnet2. Accessed: 2017-01-07.

[5] Google Brain Team. Tensorflow's website. https://www.tensorflow.org. Accessed: 2017-10-07.

[6] J Andrade, N George, K Karras, D Novo, F Pratas, L Sousa, P Ienne, G Falcao, and V Silva. Design space exploration of ldpc decoders using high-level synthesis. *IEEE Access*, 2017.

[7] Dong Wang, Jianjing An, and Ke Xu. Pipecnn: An opencl-based fpga accelerator for large-scale convolution neuron networks. *arXiv preprint arXiv:1611.02450*, 2016.

[8] Nicole Hemsoth. Intel marrying fpga, beefy broadwell for open compute future. https://www.nextplatform.com/2016/03/14/intel-marrying-fpga-beefy-broadwell-open-compute-future/. Accessed: 2016-12-26.

[9] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[10] Daniel Keysers. Comparison and combination of state-of-the-art techniques for handwritten character recognition: topping the mnist benchmark. *arXiv preprint arXiv:0710.2231*, 2007.

[11] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html. Accessed: 2017-10-13.