Hardware accelerator for the CNN algorithm used with analog input signals

Moro Peréz, G.

Award date:
2016

Hardware accelerator for the CNN algorithm used with analog input signals

Gonzalo Moro Peréz
Dept. of Electrical Engineering, Electronic Systems Group,
Eindhoven University of Technology, The Netherlands
E-mail: g.moro.perez@student.tue.nl, javialler1@hotmail.com

***Abstract*– This paper proposes a hardware accelerator for the Convolutional Neural Network (CNN) algorithm for analog sensor data and a design flow to obtain the pareto CNN structures for a certain problem. CNNs are commonly used in different image recognition tasks like face detection or object classification. Nevertheless, in this paper, the algorithm is used to process analog sensor data in Human Machine Interface (HMI) applications such as gesture recognition, proximity sensing, etc. The algorithm is very computational intense and general purpose processors are not able to fulfill the requirements of the different applications in which it can be used. Besides, energy consumption will be too high for wearable devices. A hardware accelerator is designed and tested with which a reduction in execution time of 25 times is achieved with respect to an ARM Cortex-M4, thus meeting the QoS requirements of many applications and which furthermore yields in a reduction of a factor of 20 in energy consumption.**

***Index Terms* – Convolutional Neural Network, Human Machine Interfaces, Hardware Accelerator**

I Introduction

The evolution of HMI (Human Machine Interfaces) to more natural user interfaces requires the use of complex algorithms which must achieve high accuracy rates. Besides, in many cases, they are run in wearable devices in which energy is also a major concern. These algorithms are used to perform tasks such as voice recognition, gesture recognition, eye tracking or proximity touch and are being required by more and more applications. Nevertheless, current proposals do not really match the performance requirements and therefore, new ideas should be explored.

Most of these systems use data from MEM (Micro-Electro-Mechanical) sensors, cameras or microphones as input data and they can be divided in two parts: the feature extractor and the classifier, as shown in figure 1. Examples for feature extractors are edge detection, corner detection or thresholding while examples for classifiers are decision tree, Naive Bayes or SVM. It is in general a very hard task to decide upon the best feature extractors to obtain the relevant information from the input data. In this context, machine learning algorithms are a very attractive solution since the extraction of relevant information and the classification strategy are totally learned by the algorithm during the training phase.

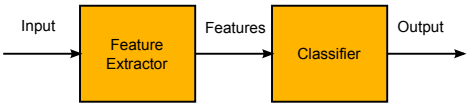

Figure 1: Block diagram of a classical recognition system

Focusing on the MEM sensor-based gesture recognition problem, traditional systems which use handcrafted algorithms do not match the performance requirements. It is a very challenging problem due to three factors [1]: Dynamic variations (gestures performed at different speeds), semantic variations (users may perform the same gestures in a different way) and volumetric variations (users can be right or left handed, the gestures can be performed differently in different contexts, etc.); and also the offset of the gravity in the acceleration measurements [2].

In a very recent work, Duffner *et al.* [1] proposed to use a machine learning algorithm, the Convolutional Neural Network (CNN), and their results outperformed all other previous approaches. The CNN algorithm had been used for long time in the field of image recognition and it currently achieves the best results in all image recognition benchmarks [3, 4]. Nevertheless, it had never been used to process MEM-sensor data. They achieved this by means of sensor fusion which is based on encoding the data from the MEM-sensors (in this case accelerometer and gyroscope) as a 2D matrix where each row is the data from each sensor and the columns represent the time domain, as shown in figure 2.

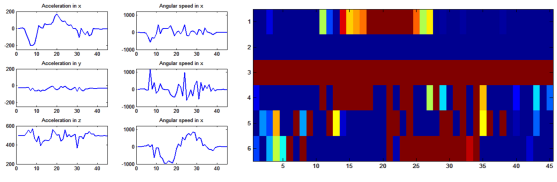

Figure 2: Sensor fusion. On the left side the raw data from accelerometer and gyroscope. On the right side, the same data encoded as a matrix

This opens possibilities, not only for gesture recognition, but also for other applications in which the CNN algorithm may outperform the methods which have been used until now. The contribution of this work is twofold:

1. A design flow to obtain the pareto CNN structures for a given problem;

2. Development and implementation of a flexible, low-power IP capable of running the CNN algorithm used to process analog-sensor signals. The IP has an optimum dataflow in terms of memory accesses via a register array to profit from data reuse in the convolution operations. Besides, whenever it is possible, the IP can completely or partially offload the host-processor to reduce power consumption.

The content of this paper is as follows. Section II discusses related work regarding CNN hardware accelerators. Section III presents a brief introduction to the CNN algorithm. Section IV discusses the CNN algorithm when used with analog sensor signals. Section V explains the design flow to obtain the pareto CNN structures for a gesture recognition application. Section VI introduces the architecture proposal and section VII describes the implementation of the CNN IP in a test platform. Section VIII shows the results. Section IX summarizes the conclusion and finally, section X proposes some future work in the scope of this project.

II Related Work

Due to the good results of the algorithm in the image recognition field, CNN hardware accelerators are a hot topic nowadays with several papers being published in these recent years [5, 6, 7, 8, 9, 10]. Their purpose is to speed-up the algorithm in order to use it in real-time applications. They all propose different architectures regarding the level of parallelism which is exploited and the internal dataflow, but they have something in common, they assume that the CNN algorithm will be used for image recognition, which had always been the case until the work of Duffner et. al. [1]. For instance, Peemen *et al.* [5] proposed a memory-centric accelerator in which convolutions are performed in parallel over the same feature map row by different processing elements (PE). Each PE has its own memory bank but it can access any memory bank of any other PE in the accelerator, which will be expensive if the number of PE is high. Besides, there is no data sharing among different PE, therefore PE data reuse is not exploited in this implementation. Later on, Peemen *et al.* [7] proposed a modified version in an SIMD processor in which a register vector is used to provide PE data reuse. Nevertheless, this implementation cannot handle subsampling and this should be done by the main processor becoming the bottleneck. Chen *et al.* [8] proposed an accelerator which parallelizes both the inner computations of a convolution filter and the convolution filters over a feature map. In the same way, no data sharing among different PE exists. Besides, different filter sizes may yield under-utilization of PE resources. Yu-Hsin Chen *et al.* [9] proposed an accelerator which consists of a 2D PE array to parallelize convolutions over the same feature map. Each PE is connected to each of its neighbours so that data reuse can be exploited. Jaehyeong Sim *et al.* [10] proposed an accelerator where the computations of convolution filters over the same as well as over different convolution maps are combined. This is done in order to profit from the data reuse available when using the same input feature map to compute several output feature maps. The proposal in this paper combines ideas from these implementations such as the parallelisation strategy in [5] or the PE data reuse in [7, 9].

III CNN algorithm overview

Convolutional Neural Networks (CNNs) belong to the family of machine learning algorithms. It is very similar to the classical Artificial Neural Network (ANN) but it has been specifically designed based on the assumption that the input is an image (or at least 2D data). Artificial neurons emulate the biological neurons in the human brain. The model of such an artificial neuron is given in figure 3.

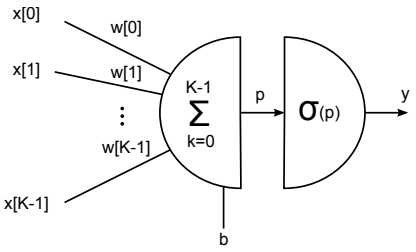

Figure 3: Model of an artificial neuron

It computes the following formula:

$$y = \sigma(b + \sum_{k=0}^{K-1} x[k] \cdot w[k]) \quad (1)$$

where y is the output of the neuron, x is the input, w is the weights, b is the bias and $\sigma()$ is a non-linear function, such as the sigmoid function or the hyperbolic tangent function.

ANNs do not scale very well when the input is an image because all input pixels should be connected to all neurons, meaning that there are a lot of connections. To cope with this problem, CNNs add a feature extraction stage before the ANN. This will filter the input image and subsample it so that the number of connections in the ANN will be considerably reduced. Therefore, two different stages in the CNN algorithm can be distinguished:

- Feature extraction layers - Convolutional layers and subsampling layers.

- Classification layers - ANN layers.

All neurons in the CNN algorithm still follow the scheme introduced in figure 3. Nevertheless, there is a difference between the neurons in the feature extraction layers and the neurons in the classification layers. The first ones connect a subset of the input pixels to create a pixel in the so-called output feature map and they all have the same weights within the same output feature map. The latter ones are fully connected to all input pixels as in the regular ANN algorithm. An example of the structure of a CNN is shown in figure 4.

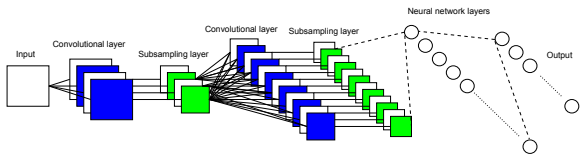

Figure 4: Convolutional neural network

This instance of the CNN algorithm includes two convolutional and two subsampling layers followed by a two layer deep ANN. In the first convolutional layer, 4 convolution filters are run, while in the second convolutional layer, there are 12 convolutional filters.

Convolutional layers run a convolution filter over a given input:

$$O(x,y) = \sum_{k=0}^{K-1} \sum_{l=0}^{L-1} w[k,l] \cdot I[x+k, y+l] + b \qquad (2)$$

where O and I are the output and input feature maps, K and L are the vertical and horizontal dimensions of the convolution filter and $w[k,l]$ and b are the weights and the bias of the filter. This operation gives one very interesting characteristic which is position invariance since the same filter runs over the whole image.

Subsampling layers are used for dimension reduction. There are different ways to carry out the subsampling such as averaging:

$$O(x,y) = \sum_{k=0}^{S-1} \sum_{l=0}^{S-1} I[x \cdot S + k, y \cdot S + l]/(2 \cdot S) \qquad (3)$$

or max-pooling:

$$O(x,y) = max_{\substack{k \in \{0..S-1\} \\ l \in \{0..S-1\}}} I[x \cdot S + k, y \cdot S + l] \qquad (4)$$

where S defines the size of the subsampling. Dimension reduction is interesting in the sense that small patches over a compressed version of the input will cover information of a rather bigger region of the input image. After each convolutional or subsampling layer a non-linear function, $\sigma()$, is added which will squeeze the data into a certain range. Neural network layers are fully connected layers which are added to carry out the final classification. In most cases, the result of the preceding layer is expressed as a one dimensional input. The operation is:

$$n(i) = \sum_{k=0}^{K} w[i,k] \cdot I[k] + b[i] \qquad (5)$$

where $I[k]$ is the input, $n[i]$ is the output, $w[i,k]$ and $b[i]$ are the trainable weights and bias and K is the number of neurons in the preceding layer.

In machine learning algorithms, all the weights are learned during the training phase. Training is the adaptation of weights such that the error between the desired output and the network output is minimized. In the case of the CNN algorithm, the weights which are learned are the coefficients of the convolution kernels, the weights in the ANN and the bias.

The most popular training algorithm is the back-propagation algorithm [11]. The weights are initialized at random and afterwards the back-propagation algorithm is run. A labelled training and a validation dataset are needed for the training process. In each training iteration, a sample from the training dataset will be picked and the machine learning algorithm will be run. Then, the back-propagation algorithm takes over. It calculates the error at the output of the machine learning algorithm and it will update its weights in such a way that the error will be reduced. After a certain number of iterations, called epoch, the performance of the trained algorithm is evaluated on a validation dataset. This process is repeated until some stop criteria is met such as an accuracy threshold for the training or the validation dataset.

The CNN algorithm has been studied for quite some time in the image processing field. Optimizations at the algorithmic level have been proposed, as done by Peemen [12] where he merged the convolution and subsampling layers in order to reduce the number of multiply-accumulate (MACC) operations without any accuracy penalty. The resulting equation for the merged layer is:

$$O(x,y) = \sum_{k=0}^{K+S-2} \sum_{l=0}^{K+S-2} \tilde{w}[k,l] \cdot I[x \cdot S + k, y \cdot S + l] \qquad (6)$$

where $\tilde{w}[k,l]$ is the new convolution kernel. This approach is interesting because then it is not needed to have specialised hardware for the convolutional and subsampling layers, since both operations can be merged into one.

Besides, different techniques have been studied in order to enhance the recognition performance of the classifier. The main problem of such machine learning algorithms is overfitting due to the large number of weights. The consequence is that it will have problems generalizing with real-world data and it will end up with a poor recognition accuracy [11]. Some of these techniques are: L2 regularization, dropout or artificially increasing the dataset.

IV CNN for analog sensor data

There is a lot of potential for the CNN algorithm to be used in applications which are based on analog sensor data. Applications in this domain may have very different requirements, from applications based on slow motion movements, for example to turn a light on or off, to gaming applications in which a fast response is needed (45ms in order not to be perceived by the user [13]).

Even though the input to the algorithm is still a 2D matrix, the information that it carries is different. The vertical dimension of the input matrix, V, will fix the number of sensors that are used while the horizontal dimension, H, will fix the time window. V could vary from a few sensors up to a maximum of 20 where accelerometers, gyroscopes, pressure sensors or microphones could be combined [14]. H together with the sampling frequency f_s will fix the maximum time during which the sensors will be sampled to complete a classification, t, according to the following equation:

$$t \leq \frac{H}{f_s} \tag{7}$$

A higher H will mean that more data needs to be processed; and for a constant f_s, it will mean more time during which data will be sampled for one classification. If f_s is increased for a constant H, the time will be reduced. If it is needed to increase the resolution (the number of samples for a classification) while keeping the time constant, both H and f_s should be increased by the same amount. Taking into account that the sampling rates of MEM-sensors can be in the order of kHz, H could easily increase up to 4000 or more.

Duffner *et. al.* [1] used an input matrix of 6x45 with a sampling frequency of around 10Hz. The reason is that they used a 3d accelerometer and gyroscope to measure slow gestures in a time frame of approximately 4 seconds. Nevertheless, as motivated above, this could be extended to input sizes of 10x100, 10x1000 or even 20x2000. The classification rate is also an important requirement of the application which could vary from 1Hz up to 100Hz. The requirements for the processor will be very different in these cases. Some estimations are shown in table 1 for an ARM Cortex-M4 processor. These are calculated by checking the number of cycles that is required for the feedforward computation of the algorithm.

For the first cases, up to operating frequencies of 240MHz, a normal processor could satisfy the QoS requirements of the application. But for the later cases, an accelerator should be implemented to achieve a considerable speed-up in the algorithm execution to meet the QoS requirements.

Besides, for a small input size, it is possible to think that the whole algorithm can be independently run by the hardware accelerator which would allow to offload the host processor completely, reducing power consumption. Nevertheless, it should be able to compute any input size regardless of the memory requirements. Therefore, three different modes should be supported by the accelerator:

Table 1: Worst-case application problem constraints

Input	Recog. rate [fps]	Required freq. [MHz]
6x45	1	1.6
6x45	10	16
6x45	100	160
20x2000	1	240
20x2000	10	2400
20x2000	100	24000

- Full (mode 0). This mode relates to the use case when the data can entirely be held in the CNN register files. Both feature extraction (FE) and classification (ANN) layers are computed in one run. It is shown in figure 5.
- Full Feature Extraction (mode 1). This mode relates to computing the feature extraction once followed by one or more runs of the classification stage. This offers opportunities to update the weights of the classification stage so that distinct classifications can be done with the same extracted feature set. The motivation for this mode is that the number of weights in the ANN phase is much higher than in the FE phase. It is shown in figure 6.
- Partial (mode 2). This mode is needed when the data does not fit in the register files. In this case, it is necessary to carry out the feature extraction and classification in several stages, and reconstruct the entire process in the host processor. This is the only mode which is present in previous proposals. It is shown in figure 7.

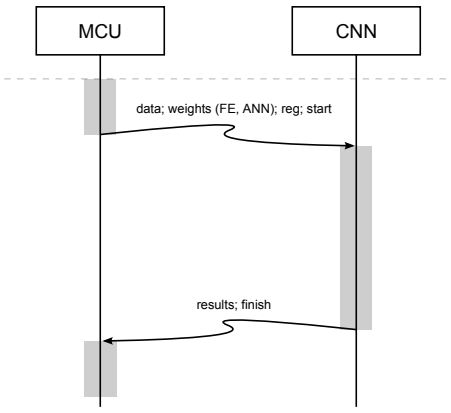

Figure 5: Full

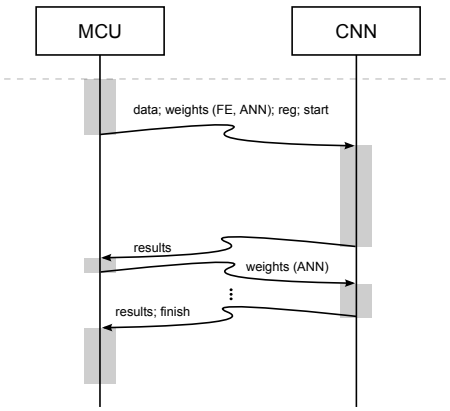

Figure 6: Full feature extraction

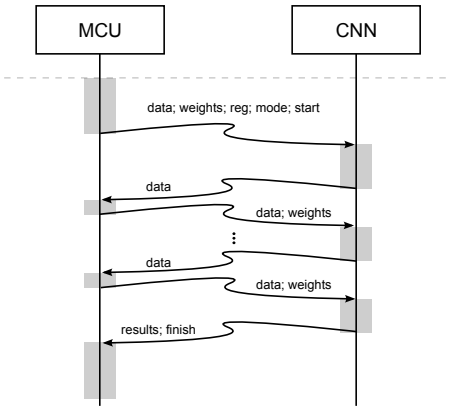

Figure 7: Partial

V Pareto CNN structure

The CNN algorithm has many free parameters which can be modified, such as the number of layers, the number of filters per layer or the sizes of these filters. These parameters will fix the so-called CNN network structure. The structure of the CNN will have a different impact than when used for image recognition. For instance, if the vertical dimension of the convolution filter is greater than one, data from different sensors will be combined, i. e. sensor fusion . Otherwise, the features will be extracted individually from each sensor. Since the structure space is huge, the task of choosing the CNN structure for a given problem is more or less random. The aim of this approach is to conveniently analyze different aspects of the network structure in a given problem so that a more rational choice regarding the structure of the CNN can be made. For this purpose, a parametrized code for the CNN algorithm and the back-propagation algorithm is written in Matlab which allows to conveniently train different CNN structures so that they can be compared in terms of different performance metrics.

A dataset of gestures was created such as done by Duffner et. al. [1]. The dataset contains 9 simple gestures which are shown in figure 8. It consists of 990

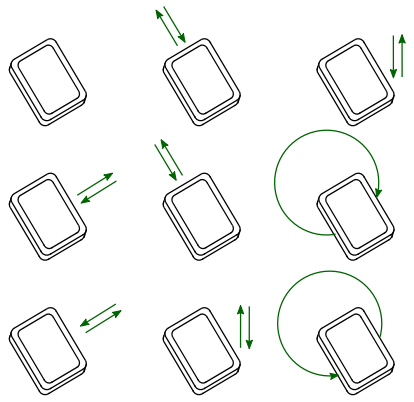

Figure 8: Movements covered in the gesture dataset

samples of these 9 gestures from 22 people, each performing 5 repetitions of each gesture. To make it more realistic and harder for the classifier, no specific instructions were given regarding how each movement should be performed. The NXP test platform *LPCXpresso54102* was used to make the measurements. A 3-d accelerator and gyroscope were used to sample the data at $f_s =$ 11Hz. The vertical dimension, V, of the input matrix is therefore 6 while the horizontal dimension, H, is set to 45 timestamps, as used by Duffner *et. al.* [1], meaning that each gesture has to be performed in approximately 4 seconds.

Different CNN network structures were trained and compared in terms of recognition accuracy and required number of MACC operations for the feed-forward execution. The recognition accuracy will give an insight of the effectiveness of the algorithm in the given problem. It is calculated as the percentage of correct classifications in the validation dataset:

$$\text{Accuracy} = 100 \text{ - } \#\text{ missclassifications} \cdot \frac{100}{\#\text{ samples}} \tag{8}$$

and the required number of MACC operations will give an idea of both computation time and energy consumption. It is calculated with the following formula:

$$\begin{aligned} &\#\text{MACC} = \\ &\sum_{l=0}^{Conv\_lay-1} (((K_l \cdot L_l) \cdot (N_l - L_l) \cdot (M_l - K_l)) \cdot \frac{conn_l}{sub_l}) \\ &+ \sum_{ln=0}^{ANN\_lay-1} (N\_num_{ln} \cdot N\_num_{ln+1}) \end{aligned} \tag{9}$$

where $Conv\_lay$ is the number of convolution layers, $conn_l$ is the number of convolution filters at each stage and ANN_lay is the number of ANN layers.

The pareto graph of the CNN structures is shown in figure 9 where the dark blue points are the pareto points together with the orange point which is the CNN structure used by Duffner et. al. [1]. Table 2 shows the parameters of each pareto CNN network structure.

From these results, it seems that the most attractive

Table 2: Parameters in the design space exploration

Pareto	Accuracy	MACC	K_num$_1$	K$_1$	L$_1$	Sub$_1$	K_num$_2$	K$_2$	L$_2$	Sub$_2$
1	89.583	2233	2	1	2	3	2	1	2	3
2	89.861	6121	4	1	2	3	5	1	4	3
3	90.694	8641	4	1	3	3	5	1	4	3
4	90.832	11817	5	1	3	3	4	1	4	2
5	90.833	13645	5	1	3	2	6	1	4	3

K_num: number of convolution filters, KxL: size of the filter, *sub*: subsampling factor

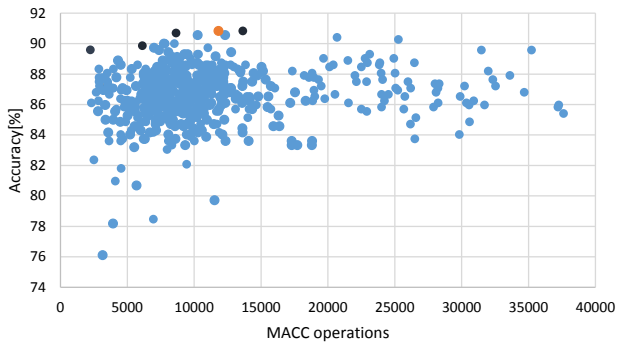

Figure 9: Graph showing the performance of different CNN structures

solution is the first pareto-point since it has a considerably lower number of required MACC operations, which will in the end yield in a reduction of execution time and energy consumption, while the loss in recognition accuracy penalty is less than 1.5%. It should be noticed that the recognition accuracies are overall lower than the ones obtained by Duffner *et. al.* [1]. This could be due to the fact that the dataset is not very rich and also some people did not perform the gestures in a very proper way. Another remark is that all the pareto configurations do not combine data from different sensors in the feature extraction layers ($K_l = 1$). This could be due to the fact that combining data from different sensors in early stages can lead to erroneous outputs. It may be preferable to first obtain simple features from the individual sensors and later on combine all the information in the classification layer.

The idea of this design approach is that for a given problem, the only task which is needed to be carried out is to collect a good and representative dataset. Then, running this code will generate a pareto-graph with the different CNN structures. The engineer should select the one which best fits the needs. In the end, since the accelerator will be flexible, all of the different structures can be mapped to it.

VI Architecture proposal

The block diagram of the architecture is shown in figure 10. It has the following main blocks:

- Control unit. It is a FSM which controls the datapath, i.e. directs the operations in all the modules of the accelerator.
- Configuration registers. These are meant to configure the structure of the algorithm. They include the kernel size, number of layers, number of kernels per layer, connections between feature extraction layers and number of neurons in the classification stage. Besides, it also includes the operation mode, and status and start registers which indicate when the CNN has finished or needs to begin the computation.
- LUT to compute the non-linear function, $\sigma()$. A look-up table is used to implement the non-linear function in each neuron. It is a fast way to realize a complex function in digital logic. The advantage is that computing the function only takes a single memory look-up regardless of the complexity of the function, so it is very fast. The disadvantage is that it increases the memory, above all when high precision is required.
- Processing slices. A processing slice consists of a processing element (PE), an individual register (R) in the register array and a register file (RF). The PE array receives the weights and bias stored in memory as well as the input data through the register array. The register array allows for PE data reuse which is not present in previous implementations [5] [8]. When the PE array finishes the computation with the given data, it writes it back to memory after going through the LUT.

Figure 11.a and 11.b respectively show a PS and a PE in detail. Note that two additional registers are added between PS to handle subsample rates of 2 and 3. The implementation of Peemen *et. al.* [7] is missing this extra registers to handle subsample. The number of registers in the register array will then be:

$$\#R = 3 \cdot \#PS – 2 \tag{10}$$

Figure 12 illustrates the data processing flow of the CNN IP for a 3x3 convolution filter with subsampling 1. The convolution operations for the first four output pixels are shown in blue, green, yellow and red and the top left input pixel of each of them is circled in red. First,

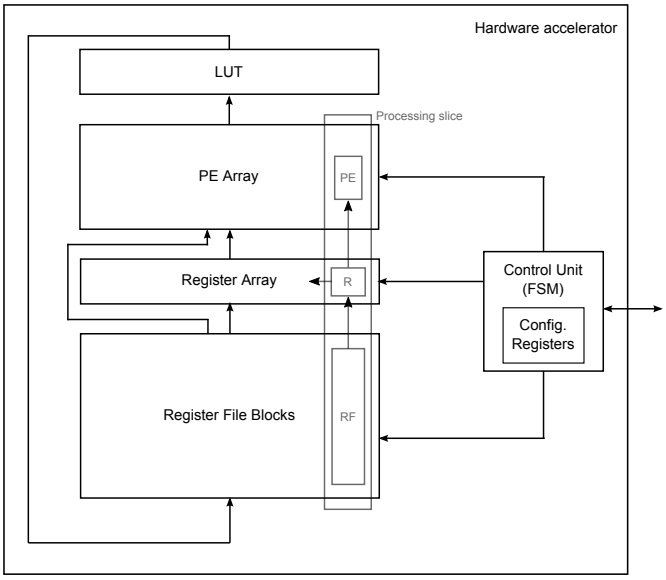

Figure 10: Block diagram of the hardware accelerator

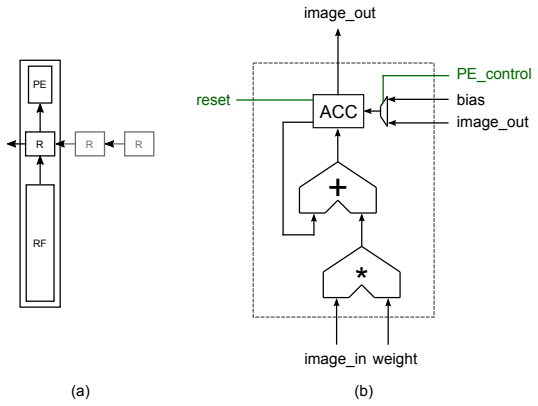

Figure 11: Block diagram of a processing slice (a) and of a processing element (b)

the input data should be distributed over the different register files in such a way that all PEs can access the values that they need at each point in time and there is no contention, meaning that two PE will never need to access a different memory address of the same register file concurrently. This is achieved by applying the following criteria and it can be checked on the right-hand side of figure 12:

$$\text{Register file} = \text{Pixel row}\%\#\text{PS} \tag{11}$$

Below, each PE is shown together with the 9 pixels that it needs at each cycle to compute an output pixel. The blue circles mean that the pixel is loaded from the register files while the green arrows mean that it comes from a neighboring PE, hence there is PE data reuse. It is possible to see how the pattern repeats: for each row of the convolution filter, all PE load an element from their own register file. Afterwards, only the last PE, in this case $PE3$, loads an element from a RF (not necessarily its own) while the others take the element which was previously used by its neighboring PE.

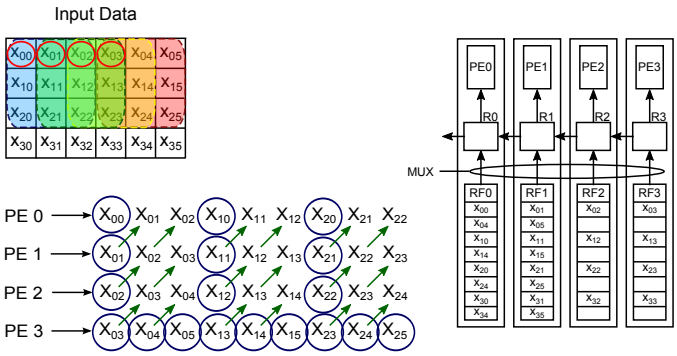

Figure 12: Example of the dataflow

Figure 13 displays the final architecture of the accelerator for 4 PS. This implies that there will be 4 PE, 10 R and 4 RF, plus the memory for the weights and the bias which is independent of the number of processing slices. The multiplexors make sure that the values follow the correct datapath from the RF to the registers in the register array and they are commanded by the control unit. The registers which are active in the register array depend on the subsampling rate of the given layer in the following way: if the subsampling rate is 1, only the registers numbered as 1, 2, 3 and 4 will be used. If the subsampling rate is 2, the registers numbered as 1, 1_2, 2, 2_2, 3, 3_2 and 4 are used. If the subsampling rate is 3, the registers numbered as 1, 1_2, 1_3, 2, 2_2, 2_3, 3, 3_2, 3_3 and 4 are used. The PEs will received the necessary data from the RF (data, weights and bias) to perform the operations and they will write the results back to the RF.

VII System implementation

The hardware accelerator needs to be programmed and receive data from a host processor. Therefore, it is added to the Beetle platform from NXP as shown in figure 14. The technology node is GF40nm HVT and the synthesis conditions are SS, -40C, 0.99V. The instance which is added has PS = 4.

The host processor is an ARM Cortex-M4 and the communication between the host processor and the CNN IP is done by means of an AHB bus. The interface of the accelerator is implemented as required by the AHB protocol and it is shown in figure 15. Apart from the clock signal (*ap_clk*), it has four more input ports to specify the memory address which will be accessed (*input_address*), enable the memory (*input_ce*), enable the write (*input_we*) and receive data (*input_in*). Furthermore, it has one output to send data (*output_out*).

The following shows the Application Programming Interface (API) calls that this accelerator supports:

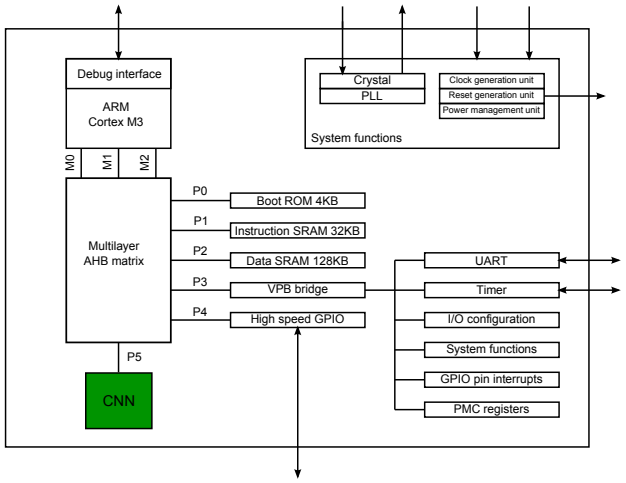

Figure 14: Block diagram of the Beetle platform with the accelerator

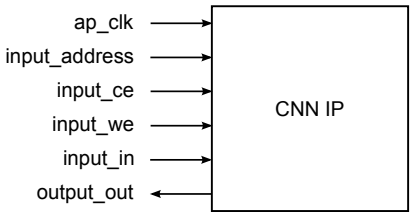

Figure 15: Interface of the CNN IP

- *send_configuration*() – It is used to program the configuration registers which will specify the CNN structure. It is used only in the initialization phase.
- *send_feature_extraction_weights*() – It is used to send the values of the weights in the feature extraction layer. When operating in mode *full* and *full feature extraction*, it is only used in the initialization phase. When operating in mode *partial*, it will also be called during the feed-forward execution.
- *send_classifier_weights*() – It is used to send the values of the weights in the classification layer. When operating in mode *full*, it is only used in the initialization phase. When operating in mode *full feature extraction* and *partial*, it will also be called during the feed-forward execution.
- *send_data*() – It is used to send the input data to process. It is called right before the start command.
- *run*() – It is used to start the computation of the algorithm.
- *read_result*() – It is used to read the results once the feed-forward execution of the algorithm has been completed.

VIII Results

The instance used by Duffner et. al. [1] is run in the Beetle platform with and without the accelerator. The results are shown in the following figure 16.

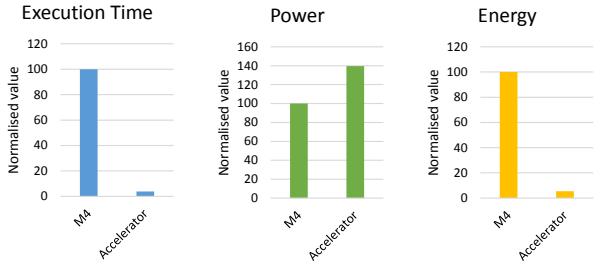

Figure 16: Comparison between processor and accelerator

The figures shows how the optimized dataflow of the accelerator and the parallelization considerably reduce execution time, almost by a factor of 25. Power consumption increases probably due to the higher switching activity. In the end, the energy consumption is reduced by a factor of 20 when running the algorithm in the hardware accelerator.

The trade-offs of the number of PS is also studied. Implementations with different degree of parallelism are tested and the results are shown in figure 17.

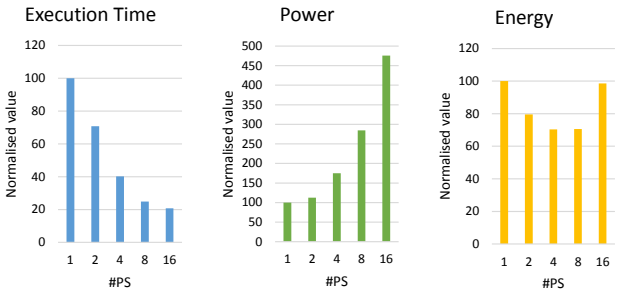

Figure 17: Experiment with different number of PS

Increasing the number of PS decreases execution time and increases power consumption because more operations are done concurrently. It is interesting to see how it influences the energy consumption. Initially, increasing the parallelism decreases the energy consumption but at some point, because there is not enough parallelism to exploit, adding more PS only increases power consumption for no gain in execution time resulting therefore in more energy consumption. This increase in power consumption could be avoided if the PS which are not used were power gated, but this is not being done in the current design.

The impact of the memory size of the accelerator is also studied. An input size of 6x45 as used by Duffner et. al. [1] is used as the input to the accelerator. Results for throughputs of 1, 10, 50 and 100 classifications per second are shown in Figure 18.

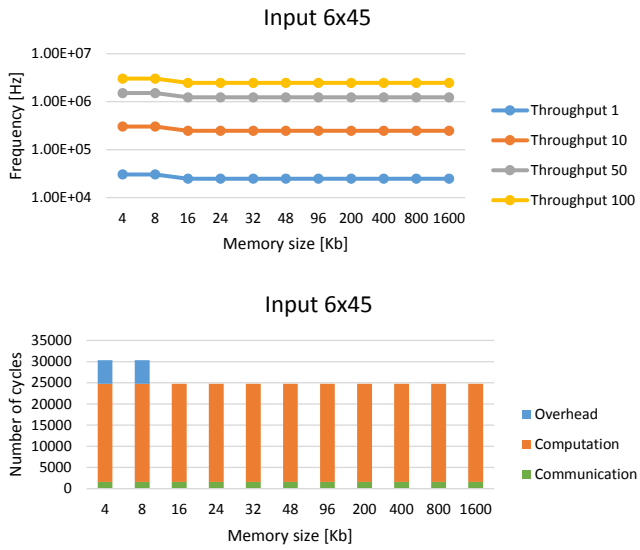

Figure 18: Experiment with different memory size

The memory size determines whether the accelerator can fit all the data required by the CNN for the feed-forward computation or not. In case that it does not completely fit, it will determine the number of extra communications (overhead) between the host processor and the accelerator. In this case, for memory sizes over 16 Kb, the accelerator has enough memory to fit all the data and therefore no communication overhead exists. This means that the total number of cycles to run the feed forward operation will be smaller than when there is overhead and therefore the required running frequency to meet the throughput requirement is lower. On the other hand, when the memory of the accelerator is not enough, there will be an overhead but still the computation of the algorithm will remain to be the bottleneck. Nevertheless, during those cycles, the core will be active so therefore the power consumption will also be higher during that period.

The choice regarding the optimal memory size in terms of throughput, energy and area is tricky. If the input size was fixed, it would be possible to really choose an optimal memory configuration. Nevertheless, this is not the case so the best criteria to choose the memory size is regarding the chip area in order not to make it too big and increase too much the actual price of the chip.

IX Conclusion

In this report, a complete approach to accelerate and decrease the energy consumption of the CNN algorithm is proposed. A new application space for the algorithm is envisioned in which a lot of applications could profit from its use. A design flow is proposed to obtain the pareto CNN structures aiming at achieving the best recognition accuracy. Besides, it will also prevent engineers from using non-optimized structures which will increase the execution time and energy consumption of the algorithm for no accuracy gain.

Furthermore, a hardware accelerator is proposed which could be added to wearable devices. The results show that a reduction in execution time of a factor of 25 is achieved with which the QoS requirements of many applications can be met for a small power penalty resulting in a decrease in energy consumption of a factor of 20. Different trade-offs regarding the architecture have been analyzed such as the degree of parallelism or the memory size which have a big impact on the performance of the accelerator.

From a more generic perspective, one conclusion which can be drawn from this paper is that specialized hardware really gives a considerable performance boost in terms of energy consumption.

X Future work

The work which remains to be done is the FPGA implementation. The accelerator was synthesized and gate-level simulations were successfully run, nevertheless there was not enough time for the real implementation in an FPGA. Besides, further work regarding the optimization of the accelerator IP could be done. During the project, an analysis of the minimum fixed data point representation for no accuracy loss penalty was done, as encouraged by Peeman [12] and B. Moons *et. al.* [15]. Nevertheless, there was no time left for the actual implementation in the test platform. This will considerably reduce power consumption and also accelerate the computations. Furthermore, it would be interesting to analyze the trade-off between memory footprint and accuracy for the LUT in more detail. In this way, both area and energy could be reduced for no noticeable accuracy loss. Also, interpolation could be added to further reduce the memory footprint. Another interesting point could be to study the possibility to use more complex PE like adder trees to reduce energy. Besides, in the experiments section, it was seen how increasing the number of PS may yield unnecessary power consumption by the non-used PS. This could be avoided by power-gating them.

Finally, more applications should be studied in which this algorithm can be used and is more efficient than the current methods which are being used. Regarding the QoS requirements of the applications, it might be necessary to use this hardware accelerator.

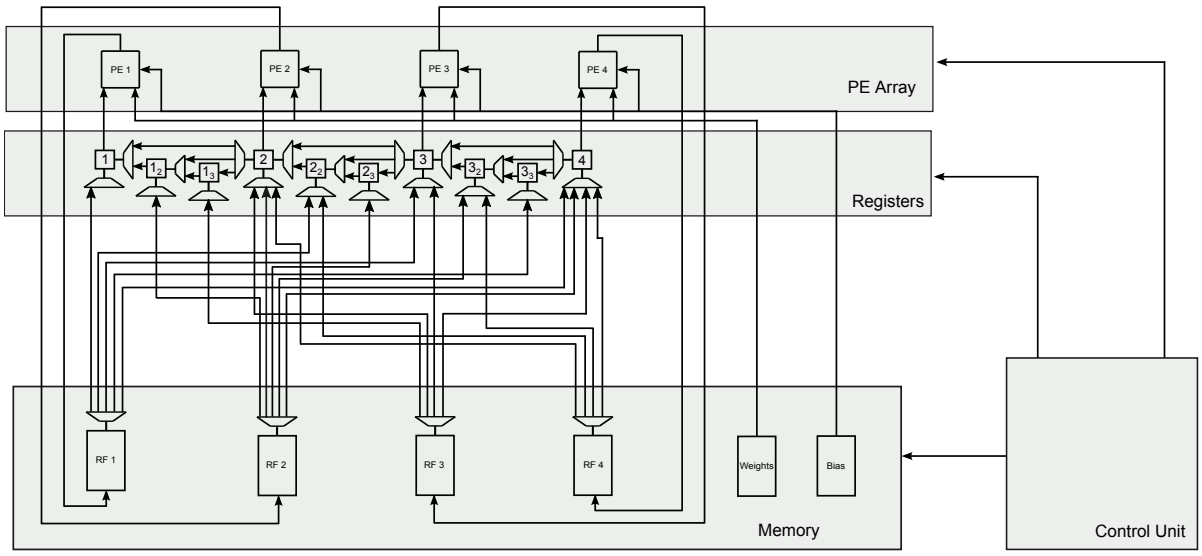

Figure 13: CNN IP for PS = 4

References

[1] Stefan Duffner, Samuel Berlemont, Grégoire Lefebvre and Christophe Garcia, "3d gesture classification with convolutional neural networks," *IEEE*, pp. 5432 – 5436, 2014.

[2] Timo Pylvanainen, "Accelerometer based gesture recognition using continuous hmms," *IbPRIA*.

[3] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[5] M.C.J. Peemen, Arnaud A. A. Setio, Bart Mesman, Henk Corporaal, "Memory-centric accelerator design for convolutional neural networks," *IEEE*, pp. 13 – 19, 2013.

[6] M.C.J. Peemen, Bart Mesman, Henk Corporaal, "A data-reuse aware accelerator for large-scale convolutional networks," *TU Eindhoven*, 2014.

[7] Maurice Peemen, Runbin Shi, Sohan Lal, Ben Juurlink, Bart Mesman, Henk Corporaal, "The neuro vector engine: Flexibility to improve convolutional net efficiency for wearable vision," *IEEE*, pp. 1604 – 1609, 2016.

[8] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, Jason Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, 2015.

[9] Yu-Hsin Chen, Tushar Krishna, Joel Emer, Vivienne Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *ISSCC*, pp. 262 – 263, 2016.

[10] Jaehyeong Sim, Jun-Seok Park, Minhye Kim, Dongmyung Bae, Yeongjae Choi, Lee-Sup Kim, "A 1.42tops/w deep convolutional neural network recognition processor for intelligent ioe systems," *ISSCC*, pp. 264 – 265, 2016.

[11] Michael Nielsen, *Neural Networks and Deep Learning*. 2015. http://neuralnetworksanddeeplearning.com/index.html.

[12] M.C.J. Peemen, "Mapping convolutional neural networks on a reconfigurable fpga platform," *TU Eindhoven*, 2010.

[13] Juan Wachs, Mathias Kölsch, Helman Stern and Yael Edan, "Vision-based hand-gesture applications: Challenges and innovations," *Communications of the ACM*, pp. 60–71, 2011.

[14] ITRS 2.0, "System integration," *International Technology Roadmap for Semiconductors*, 2015.

[15] Bert Moons, Bert De Brabandere, Luc Van Gool, Marian Verhelst, "Energy-efficient convnets through approximate computing," *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016.