"Applied Intelligence" manuscript No.
(will be inserted by the editor)

Convolutional Neural Network Acceleration with Hardware/Software Co-Design

Andrew Tzer-Yeu Chen[1] · **Morteza Biglari-Abhari**[1] · **Kevin I-Kai Wang**[1] · **Abdesselam Bouzerdoum**[2,3] · **Fok Hing Chi Tivive**[2]

Received: date / Accepted: date

Abstract Convolutional Neural Networks (CNNs) have a broad range of applications, such as image processing and natural language processing. Inspired by the mammalian visual cortex, CNNs have been shown to achieve impressive results on a number of computer vision challenges, but often with large amounts of processing power and no timing restrictions. This paper presents a design methodology for accelerating CNNs using Hardware/Software Co-design techniques, in order to balance performance and flexibility, particularly for resource-constrained systems. The methodology is applied to a gender recognition case study, using an ARM processor and FPGA fabric to create an embedded system that can process facial images in real-time.

Keywords Computer Vision · Embedded System · Neural Network · Co-design · Hardware Acceleration · FPGA · Real-time · Gender Recognition

1 Introduction

Convolutional Neural Networks (CNNs) are becoming increasingly popular for image processing tasks, combining the use of filters with neural networks in a way that is biologically inspired by the receptive fields of the visual cortex. However, Convolutional Neural Networks, and Deep CNNs in particular, are often very computationally expensive, with many layers of neurons requiring many floating-point multiplications. While this may be manageable in the laboratory, this presents significant barriers for real-world embedded system implementations. With the aim of lowering processing times, researchers first moved from typical CPUs to GPUs to leverage the inherent parallelism of both CNNs and GPUs. More recently, a few companies have developed ASIC-based platforms that place multipliers in a sea of RAM to achieve maximum performance at the cost of design and execution flexibility [1], as well as at great monetary cost.

Optimising CNNs for speed can be focused on the software algorithms or on the hardware execution platform. When trying to identify the most appropriate hardware platform many factors need to be balanced, including the design flexibility, the execution speed of the target platform, availability of fast enough memory, the level of parallelism available in the CNN algorithm itself, the required accuracy, power/energy consumption, and the monetary costs of the system. While taking advantage of parallelism to accelerate execution is generally desirable, it is rare for an algorithm to be fully executed in parallel without some sequential control flow, which affects the achievable speedup.

In general, having a limited range of parallel instructions is better suited for hardware execution, while a broad range of sequential instructions is better suited for software execution. Hardware/Software Co-design (HW/SW Co-Design) offers a balance between speed and flexibility by combining traditional software execution on CPUs or GPUs with more customised hardware accelerators on FPGAs, DSPs, or ASICs, as shown in

✉ Andrew Tzer-Yeu Chen
andrew.chen@auckland.ac.nz

[1] Embedded Systems Research Group, Department of Electrical and Computer Engineering, The University of Auckland, New Zealand

[2] School of Electrical, Computer, and Telecommunications Engineering, University of Wollongong, Wollongong, Australia

[3] College of Science and Engineering, Hamad Bin Khalifa University, Qatar

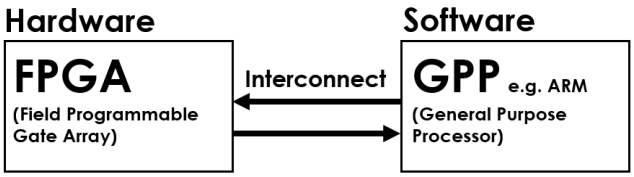


Fig. 1 General Hardware/Software Co-design Architecture

Figure 1. HW/SW Co-design has been applied in system design for embedded electronics in automobiles, avionics, industrial automation, and other areas, yet it is underused in computer vision applications. In this paper, we discuss the use of HW/SW Co-design techniques for accelerating the execution of CNNs in embedded systems. This builds upon our previous publications [2] [3] which describe the algorithmic development and subsequent acceleration of a CNN for binary male/female gender recognition of human faces. In this paper our key outputs are a deeper explanation of the underlying CNN architecture, presenting the acceleration methodology systematically, and using a much larger testing set to demonstrate the accuracy of the system. The main contribution of this paper is an explanation of a methodology for effectively implementing CNNs using Hardware/Software Co-design techniques, allowing real-time execution on embedded system platforms without significant degradation of accuracy.

The remainder of the paper is structured as follows. In Section 2, we present related work while providing some of the background of CNNs and HW/SW Co-design. Section 3 discusses the CNN architecture being studied, and the gender recognition application that we use as a case study. Section 4 describes the use of HW/SW Co-design to accelerate the execution of the CNN post-training, and Section 5 presents the results of our work. Section 6 discusses some ideas for future work and concludes the paper.

2 Literature Review

2.1 Hardware/Software Co-design for CNNs

HW/SW Co-design is a growing field of interest because it offers a path forward for system implementations that have to be both fast and correct while satisfying the design constraints [4][5]. Pure software solutions tend to be more flexible, but the sequential bottlenecks may prevent real-time execution and lead to inefficient energy use. Pure hardware solutions can be very fast, but this comes at the cost of less flexibility, as well as a need to balance the accuracy (especially for floating-point arithmetic) and implementation cost. In addition, it may require significant development costs due to the high cost of hardware fabrication and the shortage of sufficiently skilled hardware engineers. By combining the software and hardware worlds, we can strike a balance between execution speed and execution flexibility in order to achieve real-time performance with sufficient accuracy.

A number of existing papers have revealed how hardware/software co-design can accelerate algorithm execution in an embedded vision context, particularly taking advantage of the parallelism often found in image processing scenarios. For example, a vehicle detection algorithm based on the detection of taillights and finding light pairs was accelerated using an FPGA and PowerPC HW/SW co-design system, achieving 12-20x speedup in comparison to the PowerPC alone [6]. Harris Corner Detection (HCD) and Histogram of Oriented Gradient (HOG) feature calculations were implemented on a Xilinx Zynq system that includes an ARM processor and FPGA fabric, achieving a 15x speedup in comparison to an octo-core i7 CPU [7]. A recent pedestrian detection implementation uses an ARM processor for control and reconfigures an FPGA on the fly, achieving a 120x speedup over a PC-only implementation [8]. While not all HW/SW co-design approaches lead to speedups as impressive as the above cases, the examples serve to show that in the quest for real-time image processing, HW/SW co-design is a valuable part of the developers design options. Not all image processing uses CNNs, and not all CNNs are used for image processing, but the strong relationship between the two suggests that techniques that have been shown to successfully accelerate image processing may also accelerate CNN execution.

The application of HW/SW Co-design to CNN execution has thus far been relatively limited. One class of CNN accelerators exist purely on an FPGA [9] [10] [11] [18] or ASIC [12] [13]. In some cases a soft-core processor is implemented in conjunction with hardware units; in others, a fixed set of hardware units are used, sacrificing most of the flexibility in favour of higher speeds. Li and Areibi [14] present a facial recognition system incorporating a soft-core processor and FPGA fabric for a back-propagating multi-layer perceptron artificial neural network, achieving a 1.69x speedup over a pure software implementation. In [15], a hardware implementation is used to accelerate convolution operations across an image for a face detection algorithm with the help of a Nios embedded soft-core processor for control, enabling real-time processing. However, few papers report using a hard processor to run software code in conjunction with hardware such as a DSP or FPGA fabric in order to accelerate CNN execution, as proposed in our approach. One example is Qiu et al [16], who develop a

deep CNN for the ImageNet image classification competition, achieving a 1.4x speedup with a System-on-Chip with hard CPU and FPGA in comparison to a CPU alone with 14x less power consumption.

One of the continuing challenges in the use of hardware-based approaches for machine learning is the complexity of implementing floating-point arithmetic, and determining the level of precision required without wasting resources [17]. However, recent research [1] [18] has shown that as long as CNNs are trained with floating-point precision, the execution itself can be done using integer arithmetic without significantly harming accuracy. This is supported by the results of our study reported previously in [3] and also here. It should be noted that recently Gupta et al. [19] have also shown that deep CNNs can be trained using only 16-bit fixed-point representations without significantly impairing classification accuracy. In this paper, we focus largely on CNN execution, making the assumption that training can be completed on a powerful PC (i.e. offline) with the algorithm and weights then ported to a hardware implementation. This paper presents a complete look at how CNN architectures are developed, and therefore how they can be accelerated using HW/SW Co-design techniques.

2.2 Gender Recognition Implementations

Ng et al. [20] present a comprehensive survey of vision-based human gender recognition, and show that algorithmically, substantial progress has been made. While humans can achieve roughly 95% accuracy in male/female binary gender recognition, computer vision algorithms have reportedly achieved up to 99.8% accuracy in controlled environments [21] and up to 95% in uncontrolled environments [22]. Ng et al. also describe potential applications, such as biometrics, human-computer interaction, and demographic studies. In particular, they discuss demographic classifiers for customer relationship and marketing systems, which require the ability to process 15-20 images per second. This relatively fast real-time requirement disqualifies many complicated neural network-based algorithms because their processing requirements are too large. This is exacerbated by the reality that for real-world implementation, these algorithms need to run on embedded hardware platforms with constrained resources in comparison to desktop computers.

There have been relatively few hardware implementations of machine learning powered gender recognition algorithms. Perhaps the most significant is Azarmehr et al. [23], which uses a Support Vector Machine (SVM) and Radial Basis Function (RBF) Classifier to process Multi-Scale Local Binary Pattern (MSLBP) features. Their algorithm also detects faces, crops appropriately, and can characterise age. This is implemented on a 1.7GHz quad-core Snapdragon 600 SoC, and the gender recognition portion takes 2.3ms per image with 95% accuracy. The overall algorithm achieves an average performance of 15 to 20 frames per second. Other approaches in the literature include the work by Irick et al., which reports an SVM approach with a pure FPGA implementation [24], achieving lower accuracy (88%) but a much faster throughput at 1,100 frames per second (with a 100MHz clock). Irick et al. have also previously reported an artificial neural network-based system [25] with 83.3% accuracy and only 30 frames per second. Ratnakar and More [26] report an FPGA-based system that achieves 78% accuracy with a propagation delay of 1.9 seconds, which implies a very slow latency but does not necessarily indicate an improvement of the throughput.

In our previous work [3], we describe the use of HW/SW Co-design to speed up an existing shunting inhibitory neural network for a targeted advertising application, achieving just over 20 frames per second. While many gender recognition algorithms use SVMs for higher accuracy and modelling flexibility, neural networks can characterise multiple outputs based on multiple input factors or features, and are more suited for fixed hardware implementations that seek to avoid unused capacity or reliance on dynamic reconfiguration. Neural network approaches also tend to be more easily parallelisable, and can be more easily partitioned between hardware and software for HW/SW Co-design. To our knowledge, our papers are the first reports of implementations of CNN-based gender recognition algorithms in an embedded system using both a hard core processor and FPGA fabric.

3 Convolutional Neural Network Architecture

A Convolutional Neural Network (CNN) bears many similarities to the traditional Artificial Neural Network, with the key difference being that we use filters to operate on windows of an image in a way that emulates the receptive field of biological vision. CNNs scale up for arbitrarily large images much better than multilayer perceptrons, using local connections and shared weights to reduce computational complexity and memory consumption. A strong benefit of this approach is that CNNs are reasonably shift and distortion invariant, helping deal with test images captured in uncontrolled environments. CNNs are built using layers, which are comprised of filters for feature extraction, feature classification, and downsampling (pooling). Recent systems

use deep CNNs, with many, many layers; for example, Redmon et al. [27] uses 26 layers. In this paper, we use a comparatively simple CNN with only three layers to demonstrate the benefits of HW/SW Co-design.

In this paper, we use gender recognition as a case study for an application that can use an embedded system CNN. Gender recognition is an important part of creating computer systems that can develop a better understanding of the humans in their environment, such as for biometric authentication, human-computer interaction, and targeted marketing. Classification of gender is challenging because there can be many variations in physical appearance between genders and within the same gender that can lead to erroneous classification, as well as environmental conditions such as lighting and image/vision degradation that can make accurate characterisation difficult, even for humans. In many of these applications, we need computer systems that can operate in real-time conditions with limited computational resources; for example, an active smart billboard could scan the faces of the people in its audience, and deliver targeted advertisements that best reflect the gender demographics of the people actually viewing the advertisement. Building upon our previous work presented in [2], we use a CNN trained on web images in order to detect male/female binary gender from facial images. The ultimate goal is to accelerate the network so that it can be executed in real-time in an embedded system.

The network developed for gender classification comprises an input layer and three processing layers, including the classification layer, as shown in Figure 2. The input layer is used to receive a 32×32 pixel face image as the input. The first layer (Layer-1) is a convolutional layer comprised of a set of nonlinear filters with fixed kernel coefficients, designed to extract directional contrast information from the input image. The second layer (Layer-2) consists of neurons with adaptive receptive fields which are trained using supervised learning to detect discriminative features for gender classification. Finally, the third layer (Layer-3) consists of two output units to predict the gender of the test face with a certain probability. Note that this architecture is slightly different to the one described in our previous work [3], and presented in much more detail here.

3.1 Layer-1: Directional Filters

The first processing layer focuses on the extraction of contrast-invariant oriented features from the input image at different scales. For this purpose, a set of directional non-linear filters is designed using complex Gabor filters combined with the shunting inhibition mechanism [28]. A Gabor filter is a Gaussian filter modulated by a complex sinusoid, which can be expressed as:

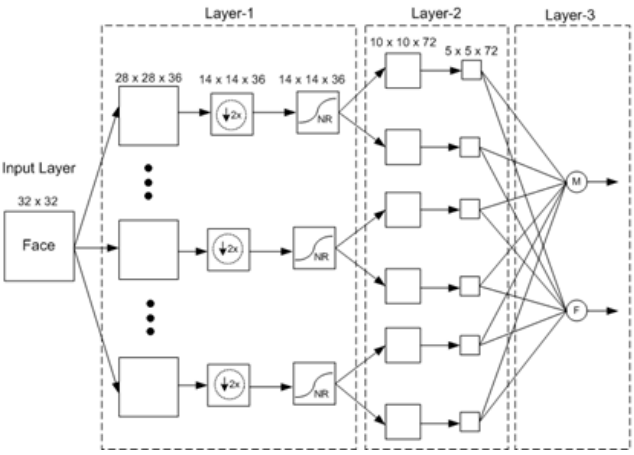


Fig. 2 Gender classification network architecture

$$g(t) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}} e^{i2\pi ft} \tag{1}$$

where σ is the standard deviation of the Gaussian filter and f is the frequency of the complex sinusoid. The bandwidth and the center frequency of a Gabor filter can be controlled by changing, respectively, the parameters σ and f. Gabor filters have been used in numerous image processing applications, including texture classification [29], face recognition [30], and hyperspectral image classification [31] to name a few. Its popularity in computer vision and image processing stems from the similarity with the receptive fields of simple cells in cat striate cortex [32]. Furthermore, Gabor filters have optimal joint resolution in both spatial and frequency domains [33]. The magnitude of the Gabor filter is commonly used as a feature since it is invariant to translation. The filter output response of the j-th orientation and k-th scale is given by:

$$\boldsymbol{X}_{j,k} = \frac{|\boldsymbol{D}_{j,k} * \boldsymbol{I}|}{|\boldsymbol{G}_k * \boldsymbol{I}| + c} \tag{2}$$

where $|\cdot|$ denotes the magnitude operator, $*$ denotes the two-dimensional convolution operator, $\boldsymbol{I}$ is the input image, $\boldsymbol{D}_{j,k}$ is a Gabor filter of the j-th orientation and k-th scale, $\boldsymbol{G}_k$ is a circular Gabor filter of the k-th scale, and c is a positive constant to avoid division by zero. Let f_k denote the k-th scale and θ_j the j-th orientation of the Gabor filter. The kernel of the directional Gabor filter $\boldsymbol{D}_{j,k}$ can be written as:

$$\boldsymbol{D}_{j,k}(x,y) = \frac{f_k^2}{\pi\gamma\eta} \exp(-[\frac{f_k^2}{\gamma^2}\overline{x}^2 + \frac{f_k^2}{\eta^2}\overline{y}^2]) \exp(2i\pi f_k \overline{x}) \tag{3}$$

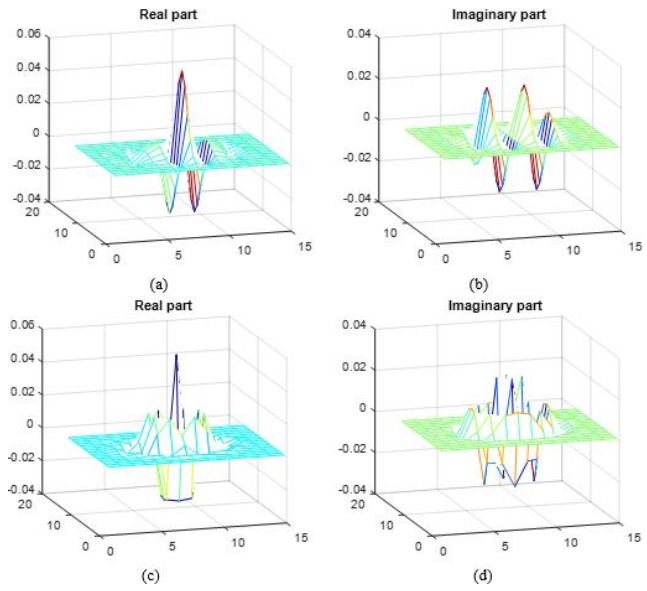


Fig. 3 In the spatial domain, (a) real part and (b) imaginary part of a directional Gabor filter, (c) real part and (d) imaginary part of a circular Gabor filter at $\theta = 0$ and $f = 0.4$

where γ and η are sharpness parameters (here set to 1 ($\gamma = \eta = 1$)), and $(\overline{x}, \overline{y})$ are the rotation of (x, y) through the angle θ_j,

$$\begin{bmatrix} \overline{x} \\ \overline{y} \end{bmatrix} = \begin{bmatrix} \cos(\theta_j) \sin(\theta_j) \\ -\sin(\theta_j) \cos(\theta_j) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{4}$$

The kernel of the circular Gabor filter G_k is given by:

$$\boldsymbol{G}_k(x,y) = \frac{f_k^2}{\pi\gamma\eta} \exp(-[\frac{f_k^2}{\gamma^2}x^2 + \frac{f_k^2}{\eta^2}y^2]) \exp(2i\pi f_k\sqrt{x^2+y^2}) \tag{5}$$

To generate a set of filters with J orientations and M scales, the centre frequency corresponding to the k-th scale is computed as:

$$f_k = \frac{f_M}{(\sqrt{2})^{k-1}}, k = 1, ..., M \tag{6}$$

and the angle θ_j is given by:

$$\theta_j = \frac{(j-1)\pi}{J}, j = 1, ..., J \tag{7}$$

where f_M is the maximum centre frequency. In this paper, a set of 36 filters were designed to detect features at 9 different orientations and 4 scales, i.e. $J = 9$ and $M = 4$, and the maximum centre frequency was set to 0.4 ($f_M = 0.4$). Figure 3 shows the real and imaginary parts of a directional and circular Gabor filter in the spatial domain.

After filtering, pooling and down-sampling operations are applied to each feature map by averaging 2×2 non-overlapping windows. The rationale for this is to reduce the number of pixels in the feature map and introduce some tolerance (invariance) to small translations and geometric distortions. The final operation in Layer-1 is contrast enhancement using the Naka-Rushton equation [34] defined by

$$\boldsymbol{Y}_{j,k} = \frac{(\overline{\boldsymbol{X}}_{j,k})^r}{(\overline{\boldsymbol{X}}_{j,k})^r + \mu^r} \tag{8}$$

where $\overline{\boldsymbol{X}}_{j,k}$ is the input pixel value, μ is the mean value of the down-sampled feature map, and r is a constant that controls the slope of the function (here r=3).

3.2 Layer-2: Trainable Filters

In Layer-2, the filter kernels are trained to detect discriminative features. Each trainable filter is cast as a neuron with a set of free parameters and an activation function. Let $Q_{j,k,l}$ denote the kernel of the jkl-th trainable filter, $b_{j,k,l}$ be a bias term, and $g(\cdot)$ denote the hyperbolic tangent function. The feature map of the j,k,l-th trainable filter is given by:

$$\boldsymbol{Z}_{j,k,l} = g(\boldsymbol{Q}_{j,k,l} * \boldsymbol{Y}_{j,k} + b_{j,k,l}) \tag{9}$$

The same pooling and down-sampling operations as in Layer-1 are applied to each feature map in Layer-2. Then, the down-sampled feature maps are arranged into a column vector, which is used as input to the classifier (Layer-3). Instead of connecting all features maps between Layer-1 and Layer-2, a binary connection scheme is adopted, which connects a feature map in Layer-1 to two feature maps in Layer-2. This connection scheme decreases the number of trainable weights, thereby reducing the complexity of the hardware implementation.

3.3 Layer-3: Classifier

To predict the gender of the input face image, two neurons with the *softmax* activation function are used in the classification layer: one neuron for the male class and the other for the female class. Let $\boldsymbol{x}$ denote the feature vector generated by Layer-2; the response of the i-th neuron with the weight vector $\boldsymbol{w}_i$ and bias d_i is given by:

$$y_i = \boldsymbol{w}_i^T \boldsymbol{x} + d_i \tag{10}$$

Using the softmax activation function on the output responses, we can compute the probability that a given input $\boldsymbol{x}$ belongs to the i-th class as:

$$P(C_i|\boldsymbol{x}) = \frac{\exp(y_i)}{\sum_{j=1}^{2} \exp(y_j)} \tag{11}$$

3.4 Training Methodology

Prior to its implementation using Hardware/Software Co-design, the gender recognition architecture is trained offline using MATLAB. Since the weights in Layer-1 are pre-designed filters, no training is required for Layer-1. The parameters in the second and third layer are learned from a training dataset, using the Levenberg-Marquardt (LM) algorithm. Instead of the sum squared error function, the cross-entropy error function is used as the objective function to be minimized; it is given by:

$$\varepsilon = -\sum_{i=1}^{P}\sum_{j=1}^{2} t_{j,i}\log(y_{j,i}) \tag{12}$$

where $t_{j,i}$ is the desired i-th class label for the j-th training sample and $y_{j,i}$ is the i-th output due to the j-th sample. Let $\boldsymbol{w}$ denote the weight vector comprising all the parameters in Layer-2 and Layer-3. The weight update rule of the LM algorithm is given by:

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \left[J^T J + \mu I\right]^{-1} J\boldsymbol{e} \tag{13}$$

where J denotes the Jacobian matrix, μ is a regularization parameter, and $\boldsymbol{e}$ is the error vector; the Jacobian matrix is computed using the standard backpropagation algorithm [35].

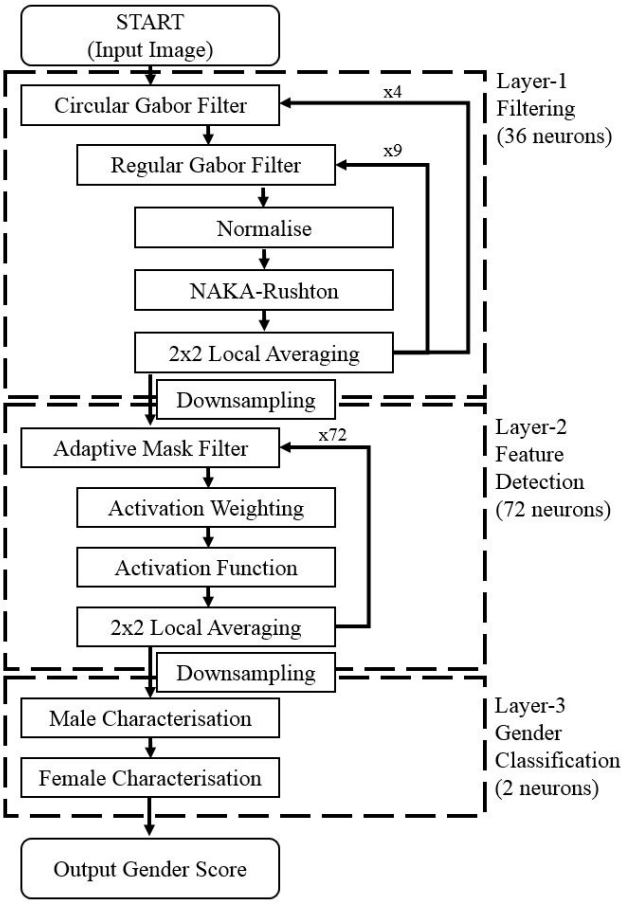


Fig. 4 Flowchart representation of the CNN algorithm

4 Hardware/Software Co-design

In order to execute the CNN in real-time in an embedded context, we can investigate the use of hardware to accelerate execution. One option is to implement the entire network using hardware only, such as on an FPGA or in an ASIC. However, this presents the immediate problem of a high cost barrier, both in terms of development time and hardware cost. We can significantly reduce both of these cost factors by utilising Hardware/Software Co-design principles. In this paper, we will use an FPGA chip with an embedded hard core processor to demonstrate how we can partition the algorithm between hardware and software; these techniques can be applied to other applications and hardware/software combinations to also achieve speed-ups in an embedded context. The FPGA acts as reconfigurable hardware fabric, which can be programmed to emulate digital circuits.

4.1 Algorithm Partitioning

Given that we have both hardware resources in the form of FPGA fabric and software resources in the form of a typical processor, the main question to be addressed is how to partition the overall algorithm between hardware and software. One of the key principles of HW/SW Co-design is leveraging concurrency and/or parallelism wherever possible. In the context of a CNN as shown in Figure 2, we can see that there is inherent parallelism; each neuron in a layer is independent and thus can be processed at the same logical time. These parallelism opportunities are targets for hardware implementation in order to avoid much slower sequential execution on the processor. When the algorithm is presented in a flowchart format such as Figure 4, the division between sequential dependencies and parallelisable loops are easier to see; for example, in Layer-2, the 72 neurons each contain multiple sequential steps, but the overall neurons can be implemented independently in hardware so that they execute at the same time.

However, hardware implementation is both difficult and costly, so the decision is not as easy as pushing all parallelisable execution onto hardware. A management decision has to be made about the amount of time and other resources available to the developer(s). As part of this decision, we should understand where the

Table 1 Execution Profiling (across 62 images)

Function	Time per call (ms)	# of calls	Total Time (s)	Total Time (%)
Gabor Filters	**2.4**	**2728**	**6.56**	**73.87**
Adaptive Mask Filter	0.33	4960	1.66	18.69
2x2 Local Averaging	0.04	7440	0.29	3.27
NAKA-Rushton	0.06	2480	0.14	1.58
Normalisation	0.05	2480	0.12	1.35
Activation Function	0.01	4960	0.06	0.68
Activation Weighting	0.01	4960	0.04	0.45

most efficient performance gains are, i.e. which parts of the algorithm can be implemented in hardware the most easily for the most speed improvement. Execution profiling allows developers to identify bottlenecks in the algorithm. This requires an initial implementation of the algorithm in software; in our case, we ported the algorithm from MATLAB to C (including rewriting built-in MATLAB functions from first principles) and then using *GNU gprof* for profiling. In our case, profiling is done on an ARM Cortex-A9 processor clocked at 400MHz, executed over 62 iterations (i.e. images). These 62 images were randomly selected, with 30 male and 32 female images, and pre-loaded into the memory, purely for the purposes of early-stage execution profiling. Table 1 shows that rather than focusing on the neurons in Layer-2 as suggested earlier, the bottleneck is in fact in the Gabor filters in Layer-1.

There are two main reasons why the Gabor filters are responsible for such a large portion of the execution time. Firstly, as shown in Figure 5, the Gabor filter uses a 5x5 convolution window with real and imaginary components (with floating-point coefficients), leading to a computational complexity of $O(50N)$ (as well as the sum and magnitude operations) for each filter, where N is the number of pixels in the image. This is exacerbated by the fact that the first layer requires 40 passes of the Gabor filter (both the circular Gabor and regular Gabor types) for each input image. Secondly, the Gabor filters are in the first layer of the CNN, before any downsampling has occurred. As the images progress through each layer of the CNN, downsampling compresses the image in order to reduce the amount of information being passed on. It therefore stands to reason that earlier layers are often more computationally expensive than succeeding layers, simply because there is more data to process, and therefore it is more efficient to target the earlier layers of the CNN.

In our application, we parallelise the Gabor filter by describing a kernalised correlation filter in VHDL with 50 parallel combinational multipliers, as well as two summation blocks and an absolute value block. The VHDL code is simulated and tested in Modelsim and then synthesised and implemented on the FPGA fabric.

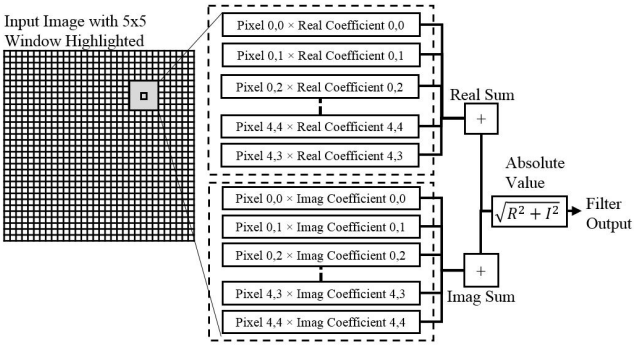


Fig. 5 Diagram of the Gabor filter operation

The hard processor system (HPS) passes a set of real and imaginary coefficients to the filter which are stored in local memory. Successive image windows (5x5 pixels) are then sent from the HPS to the FPGA, with the HPS retrieving the computed value from the FPGA. This design reduces the computational complexity of completing a Gabor filter operation from $O(50N)$ to $O(N)$. Rather than porting the entire algorithm to hardware, we can limit the amount of developer effort by simply targeting the filters in the early layer(s) of the CNN and achieve significant improvements in execution speed, as demonstrated in the results in Section 5. Other literature that covers real-time acceleration of Gabor filters using FPGAs include [36] and [37].

4.2 Data Quantisation

In the C version of the code, the Gabor filters use floating-point coefficients. While it is possible to implement floating-point multiplication in hardware, this can be very resource intensive and expensive. The main solution for addressing this problem is quantisation by rounding or truncating the data and using fixed-point integer arithmetic instead. The main concern with this approach is the inherent loss of precision that could lead to decreased accuracy. However, as discussed earlier in Section 2.2, recent research suggests that CNNs can experience little to no degradation in classification accuracy when network weights are cast to integers. While the specific Gabor filters are selected for their ability to extract certain features, it stands to reason that if integer network weights do not adversely affect the final accuracy, then the impact of quantising the coefficients may not be significant.

In our application, we use a simple Python script to extract the real and imaginary coefficients for each Gabor filter from MATLAB, multiply it by 1000 to move some of the decimal part into the integer part, and then cast the numbers to integers (thus truncating the val-

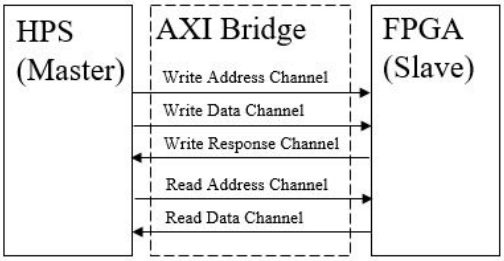


Fig. 6 Channels of the AMBA AXI bridge

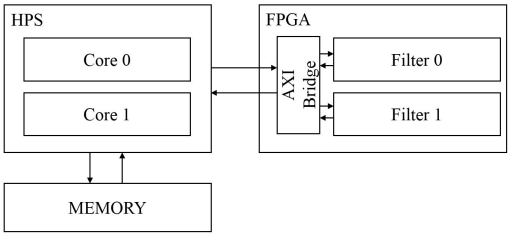


Fig. 7 Computer architecture of the HPS-FPGA system

ues). This allows us to represent each coefficient with 10-bits, reducing the amount of hardware resources required to implement the Gabor filter on the FPGA as well as the amount of memory required. As demonstrated in the results in Section 5, this does not compromise the classification accuracy at all. References [19] and [16] discuss the selection of the number of bits for quantisation in much more detail. It is also possible to learn the number of bits required using offline or online training, but the computation for this outweighs any performance gains [38].

4.3 Communication Constraints

While the FPGA may be able to process data in parallel, it must receive that data and output it afterwards. The communication link(s) between the HPS and FPGA can become a bottleneck when large amounts of data need to be transmitted, as is common in image processing applications [39]. In our application, the HPS-FPGA bridge uses the AMBA AXI bus protocol as shown in Figure 6, which introduces handshaking for passing data in either direction that requires a non-negligible amount of time. For example, a write transaction requires at least six clock cycles; this adds up quickly when we consider that for a 32x32 pixel input image, 25650 transactions are required for each of the 40 filters. We use two main approaches to help alleviate this bottleneck by reducing the amount of communication required between the HPS and FPGA.

Firstly, a shifting (or sliding) window significantly reduces the amount of data that needs to be provided to the FPGA, at the cost of a few hardware resources for control. When windows from an image are passed from the HPS to FPGA, it makes sense to iterate from the top-left corner to the bottom-right corner. As the window moves across a row of pixels, the windows are overlapping; out of 25 pixels, only 5 are actually new. We can exploit this by adding shift registers into the filter on the FPGA so that redundant transactions can be avoided. An additional transaction is required in order to control the FPGA and indicate that a shift should occur, but overall this reduces the number of required memory transactions per image by 77%.

Secondly, in our system the HPS-FPGA bridge has a 32-bit bus. Using the full width to send 10-bit numbers back and forth is therefore wasteful. The coefficient values and pixel values are concatenated together as much as possible to make full use of the 32-bit bus (also known as data packing [10]). We are able to send three coefficients or pixel values in one transaction, reducing the number of required memory transactions per image by 64%. Combining these two methods together achieves an 83% reduction, leading to much faster overall system performance.

Another method for addressing the communication constraint is the use of shared memory between the processor and FPGA, reducing the amount of data that needs to be transmitted through the communication link [7] [40]. For example, the HPS can simply send a memory address, and the FPGA can read from that address. However, there is a careful balance to be struck as accessing external memory can be costly in terms of time for FPGAs, and careful system design is required to avoid memory contention and race conditions. In our case, we did not explore the use of shared memory any further. A further improvement is to detect and remove unnecessary neurons if their operations are ineffectual [41], for example if the pixel value is zero then it may not need to be passed to the FPGA for multiplication. However, this introduces more control costs, and the performance will vary between applications, so the design decision of whether to include these checks should be considered carefully.

4.4 System Parallelism

The last major optimisation is fully leveraging the available computing resources. As previously mentioned in Section 4.1, the neurons within each layer of the CNN do not have dependencies on each other, and therefore can be implemented independently. Our target execution platform is the DE1-SoC development board, which has a Cyclone V 5CSEMA5F31C6 device. This device includes a dual-core ARM Cortex A9 (as hard

processor system or HPS) and FPGA logic cells, DSP blocks, and memory resources. We can quickly reuse the developed hardware filter and instantiate a second instance of the filter, and create two threads in the software since there are two cores, each responsible for processing half of the neurons in each layer. This leads to the system architecture shown in Figure 7. We should not expect the speed-up to be exactly 2x, because the communication bottleneck still exists, and the synchronisation time between the two threads is non-zero. However, this does lead to some performance gain.

5 Results

5.1 CNN Training and Evaluation

The gender recognition network was trained on a set of 100,000 face images (50,000 males and 50,000 females) collected from the web. The face images were collected from the web to represent general cases and avoid bias from an existing dataset with a large sample size, comprising of male and female subjects of different ages and different ethnicities. For training, the male and female face images are labeled as 1 and -1, respectively. Each face image was resized to 32×32 pixels and scaled to the range [0,1]. Two stopping criteria were used to terminate training: the average cross-entropy error falling below 0.07 as calculated using Equation 12, or the number of iterations reaching 100. The trained networks was tested on four image datasets: the BioID face dataset [42], MMI facial expression dataset [43], FERET dataset [44], and FEI Face Dataset [45]. All training and testing were done on a desktop PC in MATLAB.

The theory of the specifc CNN is described in Section 3, but it is relevant to include a summary of the justifications behind those parameters here. Preliminary experiments were conducted to determine the input size and the number of Gabor filters in Layer-1. A set of experiments with different number of scales and orientations of Gabor filter were performed. Increasing the number of filters improves the classification rate, but above 4 scales and 9 orientations, the preliminary experimental results show that there is no significant improvement in the classification rate, so we use 36 filters in order to balance classification rate and network complexity. For the input size, 32×32 pixels was chosen to achieve the state-of-the-art classification rate on the FERET benchmark dataset as reported in [2], with a compact network structure that can be implemented in hardware with real-time processing. The number of trainable filters in Layer-2 is based on the connection strategy used to connect Layer-1 and Layer-2. In this paper, a binary connection scheme was used; therefore, Layer-2 contains twice the number of Gabor filters in Layer-1. The weights of the trainable filters were obtained using the Levenberg-Marquardt supervised training algorithm, as explained in Section 3.4.

Table 2 Classification Rate of on Different Face Datasets

Database	Number of Faces	Classification Rate (CR)
BioID [42]	976 Males, 545 Females	95.4%
MMI [43]	6558 Males, 3728 Females	90.0%
FERET [44]	1718 Males, 1004 Females	93.5%
FEI [45]	200 Males, 200 Females	93.3%
Overall	9452 Males, 5477 Females	91.3%

Table 3 Comparison of Gender Recognition Classifiers on the FERET Face Database [44] (Five-fold Cross Validation)

Classifier	Number of Faces	Classification Rate (CR)
Regression Function + SVM [46]	1158M + 615F	98.8%
Our CNN	1152M + 610F	98.5%
Gabor + Fuzzy SVM [47]	160M + 140F	98.0%
SVM-RBF [48]	1044M + 711F	96.6%
2D-PCA + SVM-RBF [49]	400M + 400F	94.9%
Adaboost [50]	1495M + 914F	94.3%
PCA + SVM-RBF [51]	200M + 200F	92.3%

The gender of each input image is determined by the output with the highest probability. Table 2 presents the classification rate (CR) for each database. On average, the gender network achieves a CR of 91.3% across all four data sets. Among the four datasets, the network achieves the highest accuracy on the BioID dataset with a CR of 95.4%. The MMI dataset contains images with varied facial expressions. Since the gender recognition network was not trained on different facial expressions, its classification accuracy is affected slightly by these varying facial expressions. On the other hand, the FERET and FEI databases contain faces of subjects from a variety of ethnic backgrounds, and some of the subjects are wearing spectacles or have facial hair (beards and moustaches). Based on the cross-database evaluation, the gender network achieves a CR of 93.5% on the FERET database when trained on our web images.

In a separate test, where the network was trained and tested on the FERET images alone using five-fold cross-validation, the CR reaches 98.5%. In Table 3, we compare our CNN against other gender recognition algorithms in terms of classification rate on FERET (although with different subsets of that dataset), showing that our network achieves competitive accuracy. This further indicates that CNNs can achieve sufficiently high accuracy levels for it to be worth the design effort required to further accelerate execution towards real-time processing, if that high accuracy can be retained.

Table 4 Execution Time and Classification Rate (across 14,929 images from four datasets)

Implementation	Total Execution Time (s)	Execution Time per Image (ms)	Classification Rate (%)
Desktop PC (MATLAB)	1326.66	88.86	91.31
Desktop PC (C)	130.10	8.71	91.74
ARM Processor (HPS) Only	1955.52	130.99	91.74
HPS-FPGA Implementation	682.56	45.72	91.68

5.2 HW/SW Co-design Evaluation

We test the speedup achieved by our Hardware/Software Co-design methodology by comparing the performance of the network on the ARM Cortex-A9 processor alone with the HPS-FPGA system. The ARM processor is running at 400MHz, and the FPGA logic is clocked at 100MHz. We use both cores of the ARM and two copies of the filters on the FPGA in the HPS-FPGA system. We have expanded our test set since [3], and now use 14,929 images across four datasets. The images are resized to 32x32 pixels before processing begins, then pre-loaded into memory for the purposes of only testing the algorithm speed. For fair comparison, the software optimisation from the gcc compiler is left at the default -O0. As shown in Table 4, the final implementation achieves a 2.9x speedup in comparison to the ARM processor alone, without any adverse impacts on the classification rate (accuracy). The classification rate reported here is lower than in our previous paper [3] because of the change in training and testing datasets. We also provide the times and classification rate for the Desktop PC (3.60GHz i7 processor with 8GB RAM and SSD) for comparison purposes.

As shown in Table 5, the speed-up is largely attributable to the fact that the Gabor filters are now performed in hardware. Since the filter operates combinationally, results are available one clock cycle after all inputs are provided (i.e. 1/100MHz = 10ns). The filter executes concurrently while the HPS processes instructions that allow it to read from the AXI bridge. The bottleneck is still the HPS-FPGA communication rather than the computation itself, taking on average $\approx 200ns$ for each transaction, dominating the overall execution time. Overall, the HPS-FPGA implementation speeds up the Gabor filter in comparison to the HPS-only implementation by 3.5 times.

The execution time per image for the entire network in the HPS-FPGA implementation is 45.72ms, implying that it is possible to process at least 20 frames per second. This is fast enough for real-time applications, achieved with a fraction of the hardware cost and power consumption. Importantly, this significant speed-up was achieved with relatively low development time and cost when compared to a hypothetical pure hardware implementation of the algorithm.

Table 5 Gabor Filter Timing (across 14,929 images)

Implementation	Filter Time (s)	Time per Filter Operation (ms)	Filter Total Time (%)
Desktop PC (C)	57.85	0.10	70.46
ARM Processor (HPS) Only	1441.72	2.41	73.73
HPS-FPGA Implementation	408.70	0.68	59.31

In Figure 8, we show that while the classification rate is roughly similar between the MATLAB and HPS-FPGA implementations, the loss in precision has introduced some differences in the classification results for each image. The average error of the HPS-FPGA implementation in comparison to the MATLAB implementation is 0.0435 across all samples in terms of the raw score, and the overall classification rate has statistically negligible error. The softmax activation function computes the probability of each class (male or female), which is converted to an overall score between -1 and 1 by subtracting the output of the female neuron from that of the male neuron. Scores larger than 0 are male, and scores less than 0 are female. As the table shows, the difference is very small; the error is generally much smaller when the classification confidence is high in comparison to the borderline cases. The network is capable of working on a variety of image conditions, with different face orientations/poses, as well as artefacts such as glasses and beards. When the image is not cropped properly, contains either part of a single face or part of more than one face, or lighting reflections obscure part of the face, then the ability of the algorithm to make a robust characterisation of gender decreases.

In Table 6 we briefly present a comparison of our results with other gender recognition algorithms that have been implemented on embedded platforms, as presented in the related works. This comparison should

	MATLAB	HPS+FPGA
	0.9998	0.9998
	-0.9940	-0.9929
	0.9982	0.9980
	-0.9838	-0.9839
	-0.9699	-0.9693
	0.9606	0.9561
	-0.8622	-0.8523
	0.6042	0.5736

Fig. 8 A sample of test face images and gender scores

Table 6 Comparison of Embedded Gender Recognition System Implementations

Implementation	Accuracy	Execution Time (ms)
Ours (CNN-based, 400MHz dual-core ARM + FPGA at 100Mhz)	91.68%	45.7
SVM-based, 1.7GHz quad-core SoC [23]	95%	2.3
SVM-based, FPGA at 100MHz [24]	88%	0.9
ANN-based, FPGA [25]	83.3%	33.3
PC + myRio FPGA Kit [26]	78%	1900

be interpreted carefully, as each method uses different datasets for training and testing, with different execution platforms of varying architectures and speeds, with different underlying algorithmic approaches, and different elements of the processing included in the timing analysis. In general, our implementation approach offers the advantage of finding efficient ways of performing HW/SW Co-design, enabling real-time performance on resource-constrained embedded systems without losing significant accuracy and with low design cost.

6 Conclusions and Future Work

In this paper, we present an approach for accelerating Convolutional Neural Networks (CNNs) using a Hardware/Software Co-design approach. Using a gender recognition CNN as a case study, we investigate optimisations in two parts: in the design of the CNN itself, and in the implementation of that CNN. In terms of the CNN design, there is often a trade-off between the number of neurons and the classification rate of the system. In this paper we report that increasing the number of orientations and scales of the Gabor filters in the first layer increases the amount of features extracted from the face image and also increases the classification rate. However, including too many scales and orientations generates redundant features that produce diminishing returns and slows down the training of the neural network. Since the paper is about the development of hardware/software co-design techniques, the most compact network structure with the best classification performance on the FERET database is chosen.

In terms of the implementation of the CNN, we use four key steps that can be generally applied to CNNs. Firstly, we partition the network using execution profiling in order to identify bottlenecks, which is suitable for CNNs since they can generally be easily decomposed into constituent parts. Secondly, we apply data quantisation, which is particularly important in the context of CNNs that often use computationally expensive floating-point arithmetic. Thirdly, we alleviate communication bottlenecks by intelligently organising the data transactions. Lastly, we leverage system parallelism, which in general is only applicable if the application/algorithm being implemented is parallelisable, and is therefore very important for CNNs which are inherently parallelisable since each neuron within a layer can be executed independently. Overall, these approaches allow us to achieve a 2.9x speed up without compromising the accuracy of the network in this particular application.

Further acceleration can be achieved through addressing the communication constraints by investigating shared memory, and moving more components of the CNN from software to hardware (as long as the increased development and hardware cost is acceptable and the performance gain is larger than any overhead losses of transmitting data between the HPS and FPGA). This paper shows how accelerating even a small part of the algorithm can lead to significant speed improvements. This improvement process could also become more automated [52], depending on the library of hardware units available for parameterisation and automatic instantiation on the FPGA. The system is im-

plemented on a SoC containing an ARM processor and FPGA fabric, achieving 20 frames per second, thus satisfying the real-time requirements of the embedded system. This work demonstrates that HW/SW Co-design can be used to bring algorithms, including CNNs, out of compute-heavy research environments and into real-world resource-constrained embedded systems.

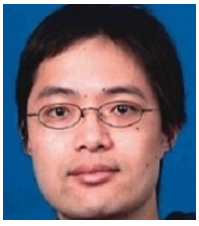
References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M, Kudlur M, Levenberg J, Monga R, Moore S, Murray DG, Steiner B, Tucker P, Vasudevan V, Warden P, Wicke M, Yu Y, Zheng X (2016) TensorFlow: A System for Large-Scale Machine Learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI).
2. Tivive FHC, Bouzerdoum A (2006) A Gender Recognition System using Shunting Inhibitory Convolutional Neural Networks. In: Intl. Joint Conf. on Neural Networks (IJCNN), pp. 5336-5341.
3. Chen ATY, Biglari-Abhari M, Wang KIK, Bouzerdoum A, Tivive FHC (2016) Hardware/Software Co-design for a Gender Recognition Embedded System. In: Trends in Applied Knowledge-Based Systems and Data Science 9799, pp. 541-552.
4. de Michell G, Gupta RK (1997) Hardware/Software Co-design. Proceedings of the IEEE 85(3):349-365.
5. Teich J (2012) Hardware/Software Codesign: The Past, the Present, and Predicting the Future. Proceedings of the IEEE 100:1411-1430.
6. Alt N, Clause C, Stechele W (2008) Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments. In: Design, Automation, and Test in Europe (DATE), pp. 176-181.
7. van der Wal G, Zhang D, Kandaswamy I, Marakowitz J, Kaighn K, Zhang J, Chai S (2015) FPGA acceleration for feature based processing applications. In: Conf. on Computer Vision and Pattern Recogn. (CVPR), pp. 42-47.
8. Tasson D, Montagnini A, Marzotto R, Farenzena M (2015) FPGA-based pedestrian detection under strong distortions. In: Conf. on Computer Vision and Pattern Recogn. (CVPR), pp. 65-70.
9. Farabet C, Poulet C, Han JY, LeCun Y (2009) CNP: An FPGA-based processor for Convolutional Networks. In: Intl. Conf. on Field Programmable Logic (FPL), pp. 3237.
10. Sankaradas M, Jakkula V, Cadambi S, Chakradhar S, Durdanovic I, Cosatto E, Graf HP (2009) A Massively Parallel Coprocessor for Convolutional Neural Networks. In: 20th Intl. Conf. on Application-specific Systems, Architectures, and Processors (ASAP), pp. 53-60.
11. Farabet C, Martini B, Corda B, Akselrod P, Culurciello E, LeCun Y (2011) NeuFlow: A runtime reconfigurable dataflow processor for vision. In: Conf. on Computer Vision and Pattern Recogn. Workshops (CVPR), pp. 109-116.
12. Cavigelli L, Gschwend D, Mayer C, Willi S, Muheim B, Benini L (2015) Origami: A Convolutional Network Accelerator. In: 25th Great Lakes Symposium on VLSI (GLSVLSI), pp. 199-204.
13. Pham PH, Jelaca D, Farabet C, Martini B, LeCun Y, Culurciello E (2012) NeuFlow: Dataflow vision processing system-on-a-chip. In: 55th Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1044-1047.
14. Li X, Areibi S (2004) A hardware/software co-design approach for face recognition. In: 16th Intl. Conf. on Microelectronics (ICM), pp. 55-58.
15. Che M, Chang Y (2010) A Hardware/Software Co-design of a Face Detection Algorithm Based on FPGA. In: Intl. Conf. on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 109-112.
16. Qiu J, Wang J, Yao S, Guo K, Li B, Zhou E, Yu J, Tang T, Xu N, Song S, Wang Y, Yang H (2016) Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. In: Intl. Symposium on Field-Programmable Gate Arrays (FPGA), pp. 26-35.
17. Maclean WJ (2005) An evaluation of the suitability of FPGAs for embedded vision systems. In: Conf. on Computer Vision and Pattern Recogn. Workshops (CVPR), pp. 131-138.
18. Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J (2015) Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In: Intl. Symposium on Field-Programmable Gate Arrays (FPGA), pp. 161-170.
19. Gupta S, Agrawal A, Gopalakrishnan K (2015) Deep Learning with Limited Numerical Precision. In: 32nd Intl. Conf. on Machine Learning (ICML), pp. 1737-1746.
20. Ng CB, Tay YH, Goi BM (2012) Recognizing Human Gender in Computer Vision: A Survey. In: Pacific Rim Intl. Conf. on Artificial Intelligence: Trends in Artificial Intelligence (PRICAI), pp. 335-346.
21. Zheng J, Lu B (2011) A support vector machine classifier with automatic confidence. Neurocomputing 74(11):1926-1935.
22. Shan C (2012) Learning local binary patterns for gender classification on real-world face images. Pattern Recogn. Lett. 4(33):431-437.
23. Azarmehr R, Laganiere R, Lee WS, Xu C, Laroche D (2015) Real-time embedded age and gender classification in unconstrained video. In: Conf. on Computer Vision and Pattern Recogn. Workshops (CVPR), pp. 56-64.
24. Irick KM, DeBole M, Narayanan V, Gayasen A (2008) A hardware efficient support vector machine architecture for FPGA. In: 16th Intl. Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 304-305.
25. Irick K, DeBole M, Narayanan V, Sharma R, Moon H, Mummareddy S (2007) A unified streaming architecture for real time face detection and gender classification. In: Intl. Conf. on Field Programmable Logic and Applications (FPL), pp. 267-272.
26. Ratnakar A, More G (2015) Real time gender recognition on FPGA. Int. J. Sci. Eng. Res. 6(2):19-22.
27. Redmon J, Divvala S, Girshick R, Farhadi A (2016) You Only Look Once: Unified, Real-Time Object Detection. In: Conf. on Computer Vision and Pattern Recogn. (CVPR), pp. 779-788.
28. Tivive FHC, Bouzerdoum A, Phung SL, Iftekharuddin KM (2010) Adaptive Hierarchical Architecture for Visual Recognition. Applied Optics 49(10):B1-B8.
29. Fogel I, Sagi D (1989) Gabor filters as texture discriminator. Biological Cybernetics 61(2):103113.
30. Wu J, An G, Ruan Q (2009) Independent Gabor analysis of discriminant features fusion for face recognition. IEEE Signal Processing Lett. 16(2):97100.
31. Li W, Du Q (2014) Gabor-filtering-based nearest regularized subspace for hyperspectral image classification. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 7(4):10121022.
32. Jones JP, Palmer L (1987) An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex. Journal of Neurophysiology 58(6):1233-1258.

33. Daugman JG (1985) Uncertainty relation for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. Journal of the Optical Society of America A: Optics, Image Science, and Vision 2(7):1160-1169.
34. Naka KI, Rushton WAH (1966) S-potentials from Colour Units in the Retina of Fish (Cyprinidae). The Journal of Physiology 185:536-555.
35. Hagan MT, Menhaj M (1994) Training Feedforward Networks with the Marquardt Algorithm. IEEE Trans. on Neural Networks 5(6):989-993.
36. Cesur E, Yildiz N, Tavsanoglu V (2012) On an Improved FPGA Implementation of CNN-based Gabor-type Filters. IEEE Trans. Circuits and Systems 59(11):815-819.
37. Pauwels K, Tomasi M, Alonso JD, Ros E, van Hulle MM (2012) A Comparison of FPGA and GPU for Real-time Phase-based Optical Flow, Stereo, and Local Image Features. IEEE Trans. on Computers 61(7):999-1012.
38. Han S, Mao H, Dally WJ (2016) Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In: Intl. Conf. on Learning Representations (ICLR).
39. Chen Y, Xu W, Zhao R, Chen X (2014) Design and evaluation of a hardware/software FPGA-based system for fast image processing. Photonic Sensors 4(3):274-280.
40. Gudis E, Lu P, Berends D, Kaighn K, van der Wal G, Buchanan G, Chai S, Piacentino M (2013) An embedded vision services framework for heterogeneous accelerators. In: Conf. on Computer Vision and Pattern Recogn. Workshops (CVPR), pp. 598-603.
41. Albericio J, Judd P, Hetherington T, Aamodt T, Jerger NE, Moshovos A (2016) Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In: 43rd Intl. Symposium on Comp. Arch. (ISCA), pp. 1-13.
42. Jesorsky O, Kirchberg KJ, Frischholz RW (2001) Robust Face Detection Using the Hausdorff Distance. In: 3rd Int. Conf. on Audio- and Video-Based Biometric Person Authentication (AVBPA), pp. 90-95.
43. Pantic M, Valstar M, Rademaker R (2005) Web-based Database for Facial Expression Analysis. In: Intl. Conf. on Multimedia and Expo (ICME), pp. 317-321.
44. Phillips PJ, Moon H, Rauss PJ, Rizvi S (2000) The FERET evaluation methodology for face recognition algorithms. IEEE Trans. on Pattern Anal. and Machine Intelligence 22(10):1090-1104.
45. Thomaz CE, Giraldi GA (2010) A New Ranking Method for Principal Components Analysis and its Application to Face Image Analysis. Image and Vision Computing 28(6):902-913.
46. Lee PH, Hung JY, Hung YP (2010) Automatic Gender Recognition Using Fusion of Facial Strips. In: 20th Int. Conf. on Pattern Recogn., pp. 1140-1143
47. Leng XM, Wang YD (2008) Improving generalization for gender classification. In: 15th Int. Conf. on Image Processing, pp. 1656-1659.
48. Moghaddam B, Yang MH (2002) Learning gender with support faces. IEEE Trans. on Pattern Anal. and Machine Intelligence 24(5): 707-711.
49. Lu L, Shi P (2009) A novel fusion-based method for expression-invariant gender classification. In: Int. Conf. on Acoustics, Speech, and Signal Processing, pp. 1065-1068.
50. Baluja S, Rowley HA (2007) Boosting sex identification performance. Int. J. of Computer Vision 71(1): 111-119.
51. Buchala S, Loomes MJ, Davey N, Frank RJ (2005) The role of global and feature based information in gender classification of faces: a comparison of human performance and computational models. Int. J. of Neural Systems 15: 121-128
52. Sahin I, Saritekin NK (2016) A Data Path Design Tool for Automatically Mapping Artificial Neural Networks on to FPGA-Based Systems. J Elec. Eng. Tech. 11(5):1921-1929.

Andrew Tzer-Yeu Chen received the BE(Hons) degree in Computer Systems Engineering with First Class Honours and the BCom degree from the University of Auckland, New Zealand, in 2015. He is currently pursuing a Ph.D. degree in Computer Systems Engineering, at The University of Auckland, New Zealand. His research interests include embedded systems, computer vision, hardware/software co-design, robotics, data analytics, and engineering education.

Morteza Biglari-Abhari received the M.Sc. degree in electrical and electronic engineering from Sharif University of Technology, Tehran, Iran, and the Ph.D. degree from the University of Adelaide, Australia. He is currently a Senior Lecturer with the Department of Electrical and Computer Engineering, The University of Auckland, Auckland, New Zealand. His research interests include computer architecture, hardware/software co-design of real-time and secure embedded systems, energy-efficient multiprocessor systems on chip, reconfigurable, and parallel architectures. He has been a reviewer for many conferences in the embedded systems research area and several journals, including the IEEE Transactions on Computers, IEEE Transactions on Circuits and Systems for Video Technology, ACM Transactions on Embedded Computing Systems, Journal of Microprocessors and Micro Systems, Journal of Real Time Image Processing, and Journal on Embedded Systems.



Kevin I-Kai Wang received the BE(Hons) degree in Computer Systems Engineering and the Ph.D. degree in Electrical and Electronics Engineering from the University of Auckland, New Zealand, in 2004 and 2009 respectively. He was an R&D Engineer designing home automation systems and traffic sensing systems from 2009 to 2010. From 2011 to 2013, he was with The University of Auckland as a Post-Doctoral Research Fellow and a Professional Teaching Fellow. Since 2013, he has been a Lecturer with the Department of Electrical and Computer Engineering, at the University of Auckland. His current research interests include distributed computational intelligence, pervasive healthcare systems, industrial monitoring and automation systems, and bio-cybernetic systems.

Abdesselam Bouzerdoum graduated with MSEE and Ph.D. degrees from the University of Washington, Seattle, USA. In 1991, he joined Adelaide University, South Australia, and in 1998 he moved to Edith Cowan University, Perth. In 2004, he was appointed Professor of Computer Engineering and Head of School of Electrical, Computer & Telecommunications Engineering at the University of Wollongong. From 2007 to 2013, he served as Associate Dean (Research), Faculty of Informatics. In addition, Dr. Bouzerdoum held several Visiting Professor Appointments at Institut Galile, Universit Paris-13, LAAS/CNRS, Toulouse, France, Villanova University, USA, and the Hong Kong University of Science and Technology. From 2009 to 2011, he was a member of the ARC College of Experts and Deputy Chair of the EMI panel (2010-2011).

Dr. Bouzerdoum is the recipient of the *Eureka Prize for Outstanding Science in Support of Defence or National Security* (2011), the *Chester Sall Award* of IEEE *Transactions on Consumer Electronics* (2005), and a *Distinguished Researcher Award (Chercheur de Haut Niveau)* from the French Ministry (2001). He served as Associate Editor for 4 International journals, including IEEE Transactions on Systems, Man, and Cybernetics (1999-2006). He has published over 300 technical articles and graduated many Ph.D. and Research Masters students. His research interest including radar imaging and signal processing, image processing, vision, machine learning, and pattern recognition.

Fok Hing Chi Tivive received the BE(Hons) in Telecommunications from Edith Cowan University and the Ph.D degree in computer engineering from the University of Wollongong, Australia, in 2001 and 2006 respectively. Since 2006, he has been with the School of Electrical Computer and Telecommunications Engineering, University of Wollongong, as a Postdoctoral Research Fellow. His research interests include machine learning, pattern recognition, image processing, and through-the-wall radar imaging.