A Resource-Efficient Embedded Iris Recognition System Using Fully Convolutional Networks

HOKCHHAY TANN, HENG ZHAO, and SHERIEF REDA, Brown University

Applications of fully convolutional networks (FCN) in iris segmentation have shown promising advances. For mobile and embedded systems, a significant challenge is that the proposed FCN architectures are extremely computationally demanding. In this article, we propose a resource-efficient, end-to-end iris recognition flow, which consists of FCN-based segmentation and a contour fitting module, followed by Daugman normalization and encoding. To attain accurate and efficient FCN models, we propose a three-step SW/HW co-design methodology consisting of FCN architectural exploration, precision quantization, and hardware acceleration. In our exploration, we propose multiple FCN models, and in comparison to previous works, our best-performing model requires $50\times$ fewer floating-point operations per inference while achieving a new state-of-the-art segmentation accuracy. Next, we select the most efficient set of models and further reduce their computational complexity through weights and activations quantization using an 8-bit dynamic fixed-point format. Each model is then incorporated into an end-to-end flow for true recognition performance evaluation. A few of our end-to-end pipelines outperform the previous state of the art on two datasets evaluated. Finally, we propose a novel dynamic fixed-point accelerator and fully demonstrate the SW/HW co-design realization of our flow on an embedded FPGA platform. In comparison with the embedded CPU, our hardware acceleration achieves up to $8.3\times$ speedup for the overall pipeline while using less than 15% of the available FPGA resources. We also provide comparisons between the FPGA system and an embedded GPU showing different benefits and drawbacks for the two platforms.

CCS Concepts: • **Computing methodologies** → **Neural networks**; • **Computer systems organization** → *Neural networks*; *Embedded hardware*;

Additional Key Words and Phrases: Iris recognition, biometrics, fully convolutional networks, deep learning, FPGA

ACM Reference format:

Hokchhay Tann, Heng Zhao, and Sherief Reda. 2019. A Resource-Efficient Embedded Iris Recognition System Using Fully Convolutional Networks. *J. Emerg. Technol. Comput. Syst.* 16, 1, Article 8 (October 2019), 23 pages. <https://doi.org/10.1145/3357796>

1 INTRODUCTION

Due to the unique and rich signatures in the irises of each individual, iris recognition has been shown as one of the most secure forms of biometric identification [10]. Unlike other biometric features such as fingerprints and voice, the irises hardly change over the course of an individual's

This work was supported by National Science Foundation grant 1814920.

Authors' address: H. Tann, H. Zhao, and S. Reda, Brown University, 182 Hope Street, Providence, RI 02912; emails: {hokchhay_tann, heng_zhao}@alumni.brown.edu, sherief_reda@brown.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Association for Computing Machinery.

1550-4832/2019/10-ART8 \$15.00

<https://doi.org/10.1145/3357796>

lifetime. Recently, iris recognition has become increasingly common on various wearable and mobile devices. For these systems, a high level of security and an efficient recognition processing pipeline with low computational complexity are the two stringent requirements for deployment.

A variety of algorithms and implementations have been proposed over the years for iris recognition pipelines [9, 27, 38, 39]. For typical processing flows, some of the main difficulties include obtaining a quality iris image and accurately segmenting the iris region. For iris segmentation, several algorithms have been developed [3, 12, 25, 26, 29, 37] using a diverse set of techniques such as circular Hough transform (CHT) and an integrodifferential operator. With the recent success of deep learning, emerging studies on iris recognition have adopted various forms of deep neural networks (DNNs) to replace different parts of traditional pipelines such as segmentation [4, 17, 21] and representation [11, 41]. In particular, using ground-truth datasets such as IRISSEG-EP [16], recent works on fully convolutional network (FCN)-based iris segmentation have shown promising improvements in robustness and accuracy.

Despite the improvements in segmentation accuracy with FCNs, existing studies focus solely on segmentation accuracy without evaluating the impacts of the models on end-to-end iris recognition systems. As evidenced in the work by Hofbauer et al. [15], segmentation accuracy alone may be insufficient when comparing multiple segmentation algorithms. In their study, they experiment with multiple iris recognition flows and demonstrate that segmentation algorithms with higher segmentation accuracy do not always lead to end-to-end flows with a better recognition rate. Thus, when comparing multiple segmentation algorithms or models, it is helpful to evaluate each using the full iris recognition pipeline to select efficient models without sacrificing overall system accuracy performance.

Existing works on FCN-based segmentation also lack evaluation of the model deployments on real HW/SW systems such as embedded systems, which are popular targets for iris recognition applications. As such, the FCN architectures are designed without taking into account the computational overheads in deployment on resource-constrained systems. Instead, the narrow focus on segmentation accuracy also leads to FCN-based designs that are extremely computationally intensive. These models can consist of a large number of layers and parameters and require billions of floating-point operations (FLOPs) for each input, making them unsuitable for embedded systems. To address the current shortfalls, in this work we propose several contributions, which are summarized as follows:

- We propose an end-to-end iris recognition pipeline with FCN-based segmentation. To the best of our knowledge, we are the first to demonstrate a complete iris recognition flow using FCN-based segmentation. To construct this pipeline, we propose an accurate contour fitting algorithm that computes center points and radii of the pupil and limbic boundaries from the FCN segmented mask. The complete flow consists of FCN-based segmentation and a contour fitting module, followed by Daugman normalization and encoding [9]. Compared to previous works, our flow sets a new state-of-the-art recognition rate on the two datasets evaluated while incurring significantly less hardware resources.
- The FCN-based segmentation portion is identified as the major bottleneck in our iris recognition pipeline. With this observation, we propose a three-step SW/HW co-design methodology to obtain a resource-efficient and accurate FCN model suitable for embedded platforms. Our technique consists of FCN architectural exploration, precision quantization using a dynamic fixed-point (DFP) format, and hardware acceleration.
- First, in architectural exploration, we propose and evaluate a large number of FCN architectures and demonstrate that a small decrease in segmentation accuracy can be traded off for an orders-of-magnitude reduction in overall computational complexities. Using the

Fig. 1. Typical processing pipeline for iris recognition applications based on Daugman [9].

end-to-end flow, we highlight the importance of evaluating the impacts of various FCN architectures using overall recognition rates rather than just segmentation accuracy.

- As a second step, we further reduce hardware complexities of the models by introducing quantization to 8-bit DFP for both weights and activations in the FCN models.
- Next, we propose a novel custom, DFP-based hardware accelerator design for the models. To compare to the floating-point format, we also synthesize a floating-point version of the accelerator.
- Finally, we provide a hardware design space exploration and comparisons through implementation of the flow using various hardware configurations and precisions, namely CPU, CPU+Accelerator on FPGA, and CPU+GPU.

The rest of the article is organized as follows. In Section 2, we provide a background of conventional and FCN-based iris segmentation and discussions of related work. In Section 3, we describe our proposed iris recognition flow in more detail. In Section 4, we discuss our resource-efficient SW/HW co-design methodology and hardware accelerator implementation. We discuss our experimental setup, as well as our experimental results, in Section 5. Finally, Section 6 provides the final discussions and concludes the article.

2 BACKGROUND AND RELATED WORKS

To capture the unique features from each individual's irises and construct their corresponding signatures, the iris recognition pipeline typically consists of multiple stages as shown in Figure 1. First, the iris image is captured using a camera, often with near-infrared sensitivity. The input image is then preprocessed for specular reflections removal and contrast enhancement. Next, a segmentation step is applied to detect the pupil, iris, and eyelid boundaries. The segmented iris region is then converted into its polar coordinate form in the normalization step. Finally, a wavelet transform is applied to encode the polar coordinate array into bitstream, which represents the unique signature of the iris [9]. Each encoding is accompanied by a mask bitstream that gives encoding bits corresponding to noniris areas such as those occluded by the eyelids or glare reflection. In this pipeline, the most computationally demanding portions are the preprocessing and iris segmentation [14, 23]. In optimizing the pipeline, it is thus most beneficial to target these first few steps, which is the focus of this work.

2.1 Traditional Iris Segmentation Methodologies

Accurate iris segmentation has been among the most popular and challenging areas in iris recognition. One of the most widely adopted segmentation algorithms was proposed by Daugman [9] using the integrodifferential operator. In this algorithm, the iris center point is located by searching through local-minimum intensity pixels throughout the image in a coarse-to-fine strategy. At each candidate pixel, a circular integrodifferential operator is applied while allowing the radius to change from a minimum to a maximum radius. This radius range is predetermined for the dataset

to contain the limbic boundary. After all of the candidate pixels are evaluated, the pixel location with the maximum in the blurred partial derivative with respect to the increasing radius is used in a fine-grain search. Here, integrodifferential operator is applied to all pixels in a small window surrounding the candidate pixels, which results in a single iris center point with the radius, r . Once the iris radius and center points are determined, a similar step is used to search a small area around the iris center point for the pupil centers. Here, the radius range is allowed to vary from 0.1 to 0.8 of the computed iris radius. The integrodifferential operator is also used to determine the elliptical boundaries of the lower and upper eyelids.

Another popular technique used in many segmentation algorithms is CHT [19, 24, 36, 38]. Typically, the Hough transform operates on an edge map constructed from the input image. The main computation can be written as

$$(x - x_i)^2 + (y - y_i)^2 = r^2,$$

where x_i and y_i are the center coordinates, and r is the circle radius. Similar to integrodifferential operator, the circle radius range for the iris and pupil boundaries are predetermined. A maximum in the Hough space corresponds to a most likely circle at radius r . The operator is used to compute two circles for the limbic and pupil boundaries. Considering that the iris region is often partially occluded by the top and bottom eyelids, two parabolic curves are used to approximate their boundaries.

The assumption of circular or elliptical limbic and pupil boundaries in the segmentation algorithms discussed can be challenging in some cases. For this reason, active contour-based segmentation algorithms were introduced to locate the true boundaries of the iris and pupil [2, 8, 33]. Considering that the segmentation output of active contour can assume any shapes, Abdullah et al. [2] proposed a new noncircular iris normalization technique to unwrap the segmentation region. Gangwar et al. [12] proposed a technique based on adaptive filtering and thresholding. Zhao and Kumar [40] proposed a total variation model to segment visible and infrared images under relaxed constraints.

2.2 FCNs for Iris Segmentation

The challenges with traditional iris segmentation methods stem from the fact that the algorithms tend to be reliant on handcrafted feature extractions and careful parameter tuning such as pre-computed radii ranges for the limbic and pupil boundaries. They can also be highly dependent on certain image intensity profiles and preprocessing steps to function correctly. In addition, separate models are typically deployed to detect the eyelids and iris regions.

With the recent advances in deep learning-based semantic segmentation, FCN-based iris segmentation methodologies have been proposed to solve the challenges facing conventional methods [4, 17, 18, 21]. Similar to successful architectures used in other semantic segmentation problems such as SegNet [5] and U-Net [32], the state-of-the-art FCN models employed in iris segmentation typically have the form of the encoder-decoder format shown in Figure 2. This architecture allows for pixel-wise labeling that conveniently produces output of the same dimensions as the input.

The success of the FCN models stems from their ability to learn and extract increasingly abstract features from the inputs. On the encoder side, it is observed that the hierarchical arrangement of convolutional layers allows earlier layers to learn lower-level features such as edges, whereas the latter layers learn more abstract, high-level concepts from the inputs. The underlying computation of each layer can be summarized as convolution operations followed by a nonlinear function such as the rectified linear unit (ReLU). The operation can be formalized as

$$B_{i,j} = f(b + \sum_m \sum_n \sum_k (A_{i+m,j+n,k} \cdot W_{m,n,k})),$$

Fig. 2. Architecture for encoder-decoder FCNs with skip connections for semantic segmentation.

where A , W , and b are the input tensor, kernel weight matrix, and a scalar bias, respectively, and $f()$ is a nonlinear function. A subset of the layers is also followed by a subsampling operation, which reduces the spatial dimension of the input, allowing the model to be translation invariant. On the decoder side, the low-resolution feature maps output by the encoder are upsampled using successions of transposed convolution layers to produce labeling prediction for each pixel in the original input image.

2.3 Metrics for Iris Segmentation Accuracy

To evaluate segmentation algorithms, there exist multiple ways to compute the segmentation accuracy. A widely accepted metric in iris recognition is the $\mathcal{F}$ -measure, which is aimed at optimizing the precision and recall performance of the segmentation output [31]. The resulting mask from a segmentation operation can be categorized into four different groups: true positive (TP), false positive (FP), true negative (TN), and false negative (FN). TP and TN represent fraction of pixels that were classified correctly as iris and noniris, respectively, with respect to the ground truth segmentation. However, FP and FN correspond to those which are incorrectly classified as iris and noniris. For a dataset with N images, the precision is then defined as

$$\mathcal{P} := \frac{1}{N} \sum_{i=1}^N \frac{TP_i}{TP_i + FP_i},$$

and recall is defined as

$$\mathcal{R} := \frac{1}{N} \sum_{i=1}^N \frac{TP_i}{TP_i + FN_i}.$$

$\mathcal{P}$ measures the fraction of predicted iris pixels that is correct, whereas $\mathcal{R}$ measures the fraction of iris pixels in the ground truth correctly identified or retrieved. $\mathcal{F}$ is then computed by taking the harmonic mean of $\mathcal{R}$ and $\mathcal{P}$:

$$\mathcal{F} := \frac{1}{N} \sum_{i=1}^N \frac{2\mathcal{R}_i\mathcal{P}_i}{\mathcal{R}_i + \mathcal{P}_i}.$$

In iris recognition, other segmentation accuracy metrics also exist, such as the Noisy Iris Challenge Evaluation—Part I [28], where segmentation errors for a dataset of N images, with $c \times r$ dimension, are defined as

$$E^1 := \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{c \times r} \sum_{j=1}^{c \times r} O(j) \otimes C(j) \right).$$

Fig. 3. Proposed FCN-based iris recognition processing pipeline.

Fig. 4. Processing pipeline for contour fitting, normalization, and encoding.

Here, $O(j)$ and $C(j)$ are the pixels from the predicted outputs and ground truth masks, respectively, and $\otimes$ is the XOR operator. A second error measure is also introduced that aims to compensate for the *a priori* probability disproportions between the iris and noniris pixels in the input images:

$$E^2 := \frac{1}{2N} \sum_{i=1}^N (FP_i + FN_i).$$

In our work, we mainly utilize the $\mathcal{F}$ -measure and also report the precision and recall performance. The E^1 and E^2 error rates can also be considered, but they would not affect our FCN optimization.

3 PROPOSED FCN-BASED IRIS RECOGNITION PROCESSING PIPELINE

Traditional iris recognition pipelines consist of multiple computation stages for image preprocessing, segmentation, normalization, and encoding as depicted in Figure 1. In our flow, the segmentation is performed using an FCN model, which allows the preprocessing stage to be eliminated. For normalization and encoding, we employ the well-known rubber sheet model and 1D log Gabor filter from Daugman [10]. To connect the FCN segmentation output with the normalization stage, we propose a contour fitting routine, which will be described next. Figure 3 illustrates our FCN-based processing pipeline that consists of FCN inference, contour fitting, normalization, and encoding.

3.1 Proposed Contour Fitting Algorithm

Daugman's rubber sheet model achieves iris 2D positional and size invariance due to a new coordinate system created by the center points and radii of the iris and the pupil [9]. With FCN-based segmentation, each output mask only identifies the pixels belonging to the iris and not the exact center coordinates or radii of the iris and the pupil. To extract this information, we develop a contour fitting routine as shown in Figure 4. Given a segmentation output mask, we first perform a rough estimate of the iris center point and radius. This is done by analyzing the largest connected

Fig. 5. Proposed SW/HW co-design strategy to achieve an efficient, accurate FCN model and fast inference.

object in the image and computing its centroid, which is the rough iris center point. The iris radius is then approximated by taking the mean of the object's major and minor axis lengths.

Using the approximated center point and radius, we perform a more fine-grain boundary fitting using CHT for circles with similar radii to the rough estimate. After obtaining the final iris radius (r) and center point (x, y), we search for the pupil using CHT for circles with a radius in the range of $[0.1r, 0.8r]$ and whose center points are within a region of interest (ROI) around (x, y). We select this radius range because biologically, the pupil radius can be anywhere between 0.1 and 0.8 of the iris radius [10]. The ROI allows for a less noisy and more computationally efficient localization of the pupil boundary.

3.2 End-to-End Recognition Rate Evaluation

The contour fitting routine produces as output the information regarding the center coordinates and radii of the pupil and limbic boundaries. This result is passed on to the normalization step based on Daugman's rubber sheet model [9], which converts the iris region into a binary grid, which is 16×256 in this work. A 1D log Gabor filter is then used to extract features from the grid producing a 16×256 -bit encoding. A 16×256 -bit mask grid is also produced to identify useful and nonuseful encoding bits. Note that the Daugman normalization used in our current pipelines assumes circular limbic and pupillary boundaries. This assumption may not be suitable for some datasets, such as those explored in Daugman [8] in which the recognition performance may be affected. However, it is a useful first-order approximation, which can be built on to better fit in those cases.

To determine whether there exists a match in the database, the Hamming distance (HD) between the input encoding $\{encodingI\}$ and every stored encoding $\{encodingS\}$ is computed as follows:

$$HD = \frac{||(\text{encoding}I \otimes \text{encoding}S) \cap \text{mask}I \cap \text{mask}S||}{||\text{mask}I \cap \text{mask}S||}, \quad (1)$$

where $\{\text{mask}I, \text{mask}S\}$ are the masks for input and stored encoding, respectively. In our work, the HD is computed for different degrees of rotation in the range of $[-35^\circ, 35^\circ]$ between the two masks. From this, the smallest HD is recorded.

4 PROPOSED SW/HW CO-DESIGN STRATEGY

In Section 4.3, we will show that similar to most iris recognition pipelines, the segmentation step is the most compute-intensive portion and takes up the majority of the overall processing time. In our flow, the segmentation runtime is mostly from FCN inference. Hence, we propose a three-step SW/HW co-design methodology, shown in Figure 5, to reduce the hardware complexities for this processing stage while maintaining high accuracy. Next, we discuss each step in the methodology in more details.

4.1 FCN Architecture Exploration

In developing FCN models to perform iris segmentation, there are many choices for architectural parameters, each of which can lead to drastically different segmentation accuracy and computational complexities. Generally, this design process uses empirical results from training and validating the models to refine the architectures. For this work, we explore network architectures similar

Table 1. Proposed Baseline FCN Architecture

Group #	Layer Type	Filter Size/Stride/Padding	Num. Outputs
-	Image Input	-	-
0	CONV	3×3/1/1	N
1	CONV	3×3/2/1	2N
	CONV	3×3/1/1	2N
2	CONV	3×3/2/1	2N
	CONV	3×3/1/1	2N
3	CONV	3×3/2/1	2N
	CONV	3×3/1/1	2N
4	CONV	3×3/2/1	2N
	CONV	3×3/1/1	4N
3	T-CONV	4×4/2/0	2N
	CONV	3×3/1/1	2N
2	T-CONV	4×4/2/0	2N
	CONV	3×3/1/1	2N
1	T-CONV	4×4/2/0	2N
	CONV	3×3/1/1	2N
0	T-CONV	4×4/2/0	N
	CONV	1×1/1/1	2
-	SOFTMAX	-	2

Each convolution layer (CONV) is followed by batch normalization and ReLU activation layers. The transposed convolution layer (TCONV) is followed by the ReLU activation layer. The arrows denote the shortcut connections, where the outputs of two layers are added together element-wise before passing to the next layer. The layers are grouped with a specific group number. The layers below group number 4 are those belonging to the decoder side, whereas those before and including group number 4 belong to the encoder side. There always exists a shortcut connection between the last encoder-side CONV layer and decoder-side TCONV layer for groups with the same group number. For the baseline model, N is set to be 16.

to U-Net model proposed in Ronneberger et al. [32]. We do not explore other model types such as DeepLab [6], Segnet [5], and Long et al. [22], as they are targeted for more complex scenes with more training examples than our application.

To obtain the most efficient set of FCN architectures with good overall recognition performance, we first create a large pool of candidate FCN models with varying computational costs. Here, the *computational cost* is defined as the number of arithmetic operations, which is the number of FLOPs required per inference. We start by designing a baseline architecture as shown in Table 1. In this model, instead of using pooling layers to downsize the input, we employ strided convolution layers (convolutions with stride greater than 1). This has been shown to have no effect on the models' accuracy performance while offering a reduced number of computations [34]. The larger models with more parameters, such as weights, tend to have the highest segmentation accuracy while requiring significant computational resources. However, the number of parameters must also be selected with care relative to the size of the available training data. Models with too many parameters on a small dataset can overfit and generalize poorly.

With a baseline architecture designed, we iteratively construct different FCN variants by performing a grid search on a few architectural parameters. The choices of parameters are chosen such that they have significant impact on the computational complexities of the models. The three parameters are as follows:

- *Input Image Scaling.* The spatial dimensions of the input iris image directly affect the number of computation required at each layer. Although the original image resolution offers more detailed and fine features, segmentation using a scaled-down version of the input could offer a significant reduction in the number of computations with limited effect on segmentation accuracy. We explore three different scaling factors in this work, namely 1 (original resolution), 0.5, and 0.25. For instance, a scaling factor of 0.5 means that each spatial dimension of the input image is reduced by half.
- *Number of Layers.* We explore FCN models with a wide ranging number of layers for each dataset. The maximum number of layers explored is 18, as shown in Table 1. We obtain models with a smaller number of layers by removing layers in groups. For instance, removing layers with group number 3 would result in a 14-layer network. However, we set a strict constraint that the spatial dimensions of the smallest feature maps in the models, namely the outputs of group 4, are kept fixed at $\frac{1}{16}$ the original dataset resolution.
- *Number of Feature Maps/Channels per Layer.* This parameter is denoted by variable N in Table 1 and quadratically impacts the computational complexity of each FCN layer. For efficiency, we limit the maximum number of output feature maps to be 64 in any layer. Starting from the baseline architecture, we experiment with four different values for N , which are {4, 8, 12, 16}.

However, several architectural choices are kept constant across all of the models. For instance, the filter size of all convolution layers are also kept fixed at 3×3 , except for the last convolution layer, which is 1×1 . The size is 4×4 for all transposed convolution layers. Nonstrided convolution layers are padded to keep the spatial dimensions of the output feature maps the same as their inputs.

Each candidate model is trained using the backpropagation algorithm with stochastic gradient descent (SGD) and momentum weight updates:

$$\Delta \mathbf{W}_{t+1} = \beta \Delta \mathbf{W}_t - \eta \nabla \mathcal{L}(\mathbf{W})$$

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \Delta \mathbf{W}_{t+1},$$

where β and η are the momentum and learning rate, respectively. For loss function $\mathcal{L}(\mathbf{W})$, we use cross-entropy loss where there are two output classes, iris and noniris, for each pixel. This loss can be written as follows:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{c \times r} \sum_{i=1}^{c \times r} (y_i \log p_i + (1 - y_i) \log(1 - p_i)),$$

where $y_i \in \{0, 1\}$ and $p_i \in [0, 1]$ are the ground truth and predicted label for each pixel, respectively. This loss function works well when the number of pixels in each class are roughly equal. In reality, most images captured for iris recognition contain a much smaller iris area compared to noniris. Thus, we introduce an additional parameter to compensate for the disproportionality of the two classes *a priori* probabilities as follows:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{c \times r} \sum_{i=1}^{c \times r} ((1 - \alpha)(y_i \log p_i) + \alpha(1 - y_i) \log(1 - p_i)),$$

where $\alpha \in [0, 1]$ is the ratio of the iris to noniris area and is precomputed from the training set.

Segmentation Accuracy Evaluations. We evaluate two well-known datasets in this work, namely CASIA Interval V4 [1] and IITD [20]. Figure 6 shows the $\mathcal{F}$ -measure performance and computational complexity, defined as the number of FLOPs required per inference, of candidate FCN models evaluated. For each dataset, the models were trained on a training set, and the reported $\mathcal{F}$ -measures in Figure 6 are obtained using a disjoint test set. The training and validation sets

Fig. 6. $\mathcal{F}$ -measure segmentation accuracy and computational complexity of candidate FCN models on CASIA Iris Interval V4 and IITD datasets. The scales refer to the ratio of the model input dimensions to the original image resolutions from the datasets. Smaller resolution inputs can significantly reduce the computational complexity of the models. We label models that make up the Pareto fronts as FCN0-FCN8 for CASIA4 and FCN9-FCN19 for IITD.

are 80% of the original dataset, with the remaining 20% for the test set. For models using scaled-down images, each input is first downsized according to the scale factor. The output segmentation mask is then resized back to the original resolution before the $\mathcal{F}$ -measure is computed. We use the nearest-neighbor approach for the both resizing operations. Note that in our architectural explorations, we train separate networks for the two datasets for fair comparisons with previous works. This does not limit the applicability of our models, as techniques such as domain adaptation [18] can be applied for new unseen datasets.

As illustrated in Figure 6, different $\mathcal{F}$ -measures can result in drastic difference in FCN computational complexities. For the two datasets, our architectural explorations result in models with three orders of magnitude range in complexity, between 0.002 and 2 GFLOPs. The results also show that models using input size closer to the original resolution tend to perform slightly better; however, they are significantly more complex computationally than the lower-resolution counterpart. In addition, for each input size, the different architectural choices can lead to orders of magnitude difference in the number of computations and segmentation accuracy. For both datasets, the accuracy performances for networks using different input scalings saturate at different points, beyond which small additional accuracy improvement require orders of magnitude increase in model complexity. This saturation behavior is also observed when all scaling factors are combined. We provide architectural descriptions of each model from the Pareto fronts (FCN0-FCN8 and FCN9-FCN19) for the two datasets in Table 2.

To compare the efficiency and segmentation performance of our models to previous works, we also evaluate each model using the full dataset. Table 3 shows the results from our best-performing model and those from previous works. The segmentation accuracy of other works reported in the table are obtained from IrisSeg [12] and IrisDenseNet (IDN) [4]. Previously, IrisSeg achieved better segmentation accuracy performance in comparison to other non-FCN segmentation methods such as GST [3], Osiris [26], Masek [25], WAHET [37], and CAHT [29]. This result was outperformed by the FCN-based segmentation method proposed by IDN from Arsalan et al. [4]. In comparison to

Table 2. Architectural Descriptions for FCN Models That Form the Pareto Fronts in Figure 6 for the CASIA Interval V4 and IITD Datasets

Model	CASIA Interval V4		Model	IITD	
	Scaling/ N	Architecture		Scaling/ N	Architecture
FCN0	1/12	0-1-2-3-4-3-2-1-0	FCN9	1/16	0-1-2-3-4-3-2-1-0
FCN1	1/8	0-1-2-3-4-3-2-1-0	FCN10	1/8	0-1-2-3-4-3-2-1-0
FCN2	1/12	0-1-2-4-2-1-0	FCN11	1/6	0-1-2-3-4-3-2-1-0
FCN3	1/4	0-1-2-3-4-3-2-1-0	FCN12	1/4	0-1-2-3-4-3-2-1-0
FCN4	0.5/8	0-1-2-4-2-1-0	FCN13	1/4	0-1-2-4-2-1-0
FCN5	0.5/4	0-1-2-4-2-1-0	FCN14	0.5/8	0-1-2-4-2-1-0
FCN6	1/4	0-1-2-4-2-1-0	FCN15	0.5/4	0-1-2-4-2-1-0
FCN7	0.25/4	0-1-4-1-0	FCN16	0.25/8	0-1-4-1-0
FCN8	0.25/4	0-4-0	FCN17	0.25/4	0-1-4-1-0
			FCN18	0.25/8	0-4-0
			FCN19	0.25/4	0-4-0

Scaling denotes the input image scaling, and N determines the number of channels as in Table 1. The architectures are shown as a list of group numbers, each of which corresponds to the various layers shown in Table 1. The layers to the right of group number 4 belong to the decoder side, whereas those to the left of and including group number 4 belong to the encoder side. Similar to Table 1, there exist shortcut connections between the last encoder-side CONV layers and decoder-side TCONV layers for groups with the same group number, which are not shown here.

Table 3. Segmentation Accuracy Comparison to Previous Works

DB	Method	R		P		F	
		μ	σ	μ	σ	μ	σ
CASIA4 Interval	GST [3]	85.19	18	89.91	7.37	86.16	11.53
	Osiris [26]	97.32	7.93	93.03	4.95	89.85	5.47
	WAHET [37]	94.72	9.01	85.44	9.67	89.13	8.39
	CAHT [29]	97.68	4.56	82.89	9.95	89.27	6.67
	Masek [25]	88.46	11.52	89.00	6.31	88.30	7.99
	IrisSeg [12]	94.26	4.18	92.15	3.34	93.10	2.65
	IDN [4]	97.10	2.12	98.10	1.07	97.58	0.99
	Our FCN0	99.41	0.40	98.93	0.75	99.17	0.40
IITD	GST [3]	90.06	16.65	85.86	10.46	86.60	11.87
	Osiris [26]	94.06	6.43	91.01	7.61	92.23	5.80
	WAHET [37]	97.43	8.12	79.42	12.41	87.02	9.72
	CAHT [29]	96.80	11.20	78.87	13.25	86.28	11.39
	Masek [25]	82.23	18.74	90.45	11.85	85.30	15.39
	IrisSeg [12]	95.33	4.58	93.70	5.33	94.37	3.88
	IDN [4]	98.00	1.56	97.16	1.40	97.56	1.04
	Our FCN9	98.92	0.87	98.33	1.13	98.62	0.65

Best recall, precision and f-measure results are shown in bold.

the IDN model, which requires more than 100 GFLOPs per inference, both of our FCN architectures need fewer than 2 GFLOPs, as shown in Table 2, which is 50 $\times$ more efficient. This large difference in computational overhead can be attributed to the fact that our network architectures are significantly shallower with far fewer numbers of feature maps per layer. In addition, our models utilize a few shortcut connections instead of the costly dense connectivity.

4.2 Quantization to DFP

As demonstrated by Hashemi et al. [14] and Tann et al. [35], reducing the data precision in DNNs can significantly lower the computational overheads of the models. With the Pareto front models identified in Figure 6, we co-design their data precision such that they can be run using lower-cost computational units on the targeted hardware platform. Considering that quantization is a time-consuming process, we do not target other models that are not on the Pareto fronts.

The numerical ranges of the weights and activations in DNN models can vary drastically between different layers. Previous works [13] have shown that even quantizing the weights and activations to a 16-bit uniform fixed-point format significantly degrades the accuracy of models in comparison to the original floating-point representation. To represent these different ranges using a small number of bits, we propose to quantize the FCN models to DFP [7] for both the weights and activations. Within a layer, DFP behaves exactly like a normal fixed-point format. However, the radix location is allowed to vary between different layers for DFP. In this format, each layer in the FCN models is represented by five hyperparameters, namely (w_{bw} , a_{bw} , w_{fl} , a_{in} , a_{out}), for bitwidths of the weights and activations/feature maps, and fractional lengths of the weights, input feature maps, and output feature maps, respectively. We fix the bitwidths of both weights and activations of all layers to be 8 bits.

To determine the proper fractional lengths for the weights and feature maps of each layer, we first perform profiling for the weights and activations of the trained floating-point models. For the weights, we select layer-wise fractional lengths such that no overflow exists during the quantization. For the activations or feature, the profiling is done by using a randomly selected subset of training data to perform forward passes with the models. During this inference process, we record the largest activation for each layer. Similar to the weights, we then select layer-wise fractional lengths such that there is no overflow. With these hyperparameters in place, we then quantize the floating models to DFP by employing a procedure similar to that of Hashemi et al. [13] using the straight-through estimator.

4.3 Hardware Acceleration Iris Recognition Pipeline on Embedded SoC

So far, the majority of work on iris recognition focuses mostly on algorithmic designs such as segmentation and feature extraction. There exist only a few studies on the system design and implementation aspect. Hashemi et al. [14] and López et al. [23] implemented full recognition pipelines on an embedded FPGA platform and showed that careful parameters optimization and SW/HW partitioning are required to achieve acceptable runtime. For iris recognition with FCN-based segmentation, existing studies so far are only concerned with achieving state-of-the-art segmentation accuracy without considerations for computational costs of the proposed designs. As such, full system analysis and implementation of these processing pipelines have not been demonstrated. In this section, we describe our implementation of the FCN-based iris recognition pipeline targeting an embedded FPGA SoC. We provide analysis of the system runtimes and bottlenecks. We also propose a hardware accelerator, which is able to achieve significant speedup computations relative to the on-board CPU core.

4.3.1 Runtime Profiles for Iris Recognition Pipeline. As an initial step, we implement the iris recognition pipeline in software running on the physical CPU core on the FPGA SoC. Our pipeline consists of four main modules, namely segmentation, contour fitting, normalization, and encoding. The segmentation step can be performed using different FCN models, which can lead to vastly different runtimes. However, the runtimes for the remaining three components stay approximately constant across different input images and FCN models. This is because the dimensions of the input and output images for these three modules are constant.

Fig. 7. FCN-based iris recognition pipeline runtime breakdown for FCN0–FCN8 models from CASIA Interval V4 Pareto front in Figure 6. From left to right, the FCN models are arranged in increasing computational complexity.

Table 4. Runtime Profile for FCN Inference Using the On-Board CPU

Function	Init	Im2Col	GEMM	Activation (ReLU)
Percentage	1.31	10.58	80.77	7.34

With this setup, we profile the runtime of the different components in the pipeline, which is shown in Figure 7. Here, we observe that the majority of the runtime is spent in the segmentation stage. This is especially true for larger FCN models where segmentation takes up more than 95% of the total runtime. Therefore, it is reasonable to focus our efforts on accelerating the segmentation component of the pipeline, which essentially is the inference process of the FCN model. To effectively speed up this operation, next we explore the runtime profiles for FCN model components.

4.3.2 FCN Processing Components. In this work, our FCN models are implemented and trained using the Darknet framework [30]. Each model consists of multiple layers with different computational requirements, and each layer consists of multiple components as listed in Table 4. Here, the Init function is responsible for ensuring that the output matrices are properly initialized and zeroed out. Note that batch normalization layers are used in training, but they are not shown here because the trained normalization parameters ($\mu, \sigma^2, \gamma, \beta$) can be folded into the network parameters in inference as such:

$$\hat{\mathbf{w}} = \gamma \cdot \mathbf{w} / \sigma^2$$

$$\hat{\mathbf{b}} = \gamma \cdot (\mathbf{b} - \mu) / \sigma^2 + \beta,$$

where $\mathbf{w}$ and $\mathbf{b}$ are the trained weights and biases of the preceding convolution layer. With this, the forward computation can be carried out using $\hat{\mathbf{w}}$ and $\hat{\mathbf{b}}$ without the batch normalization layers. The Im2Col function is a standard operation that converts the input images/feature maps into column format. With this, the convolution operations can be carried out using a general matrix-to-matrix multiplication (GEMM) routine. For the transposed convolution layer, a similar operation is used to convert column data to an image instead. The GEMM unit is essentially responsible for the multiplication of two matrices: the weights and input feature maps. The results in Table 4 show that the GEMM unit is the most time-consuming portion, taking up more than 80% of the module

Fig. 8. Overall system integration and the hardware accelerator module for the GEMM unit. The code representing the operations of the hardware module is shown in the bottom left, where A and B are the multiplicand and multiplier matrices, and C is the resulting output matrix. For the DFP version of the accelerator, A and B are 8 bit and C is 16 bit. A, B, and C are all 32-bit floats for the floating-point version. The accelerator module is connected to the Zynq Processor Unit via the ACP.

runtime. The remaining 20% is spent mostly on Im2Col and the activation function, which is the ReLU in this case.

The resources on board the SoC allow for multiple choices for accelerating the pipeline, including parallelization and vectorization using embedded CPU cores and a custom hardware accelerator on the programmable logic (PL) fabric. In comparison to the PL, parallelization and vectorization on the CPU offer a limited number of arithmetic processing units; however, accelerators on the PL side can face challenges in the limited on-chip buffer and memory bandwidths. Thus, to efficiently utilize the available hardware resources, we leave the control logic and memory access-intensive component, Im2Col, in software and move the computational-intensive module, GEMM, to PL by synthesizing a custom accelerator. For the activation function, we process it using the CPU core in parallel to the accelerator unit. Next, we describe our accelerator architecture in detail.

4.3.3 Hardware Accelerator Architecture. For FCN models, the GEMM operation is carried out in every layer between the weight and input feature matrices. The dimensions of the two matrices can be represented by a 3-tuple, (M, K, N) , where the weight matrix is $M \times K$ and the input feature matrix is $K \times N$. The output feature matrix is then $M \times N$. Between different layers of an FCN model, (M, K, N) vary significantly depending on the sizes and number of the input and output feature maps. Evidence of this can be observed in our network architecture shown in Table 2 for CASIA Interval V4. In this architecture, after the Im2Col operation, the (M, K, N) dimensions would be (16, 9, 76800) for Layer 1, whereas for Layer 2, these dimensions become (32, 144, 19200). Among FCN models that use different input image scaling factors, these dimensional differences are even more drastic. As such, the accelerator unit must be able to accommodate these dimensional variations and maximize utilization across all models explored.

Figure 8 shows the overall system integration and the architecture of the accelerator core. We implement tiling buffers for the weights (Buffer A), input features (Buffer B), and output features (Buffer C). The sizes of these buffers are selected based on the greatest common divisor among the

Fig. 9. A closer look at the datapaths of the buffers in the DFP accelerator unit.

models. For the candidate models in Figure 6, these turn out to be 8×9 for matrix A, 9×224 for B, and finally 8×224 for matrix C. Note that since we do not target a specific model, the sizes for A, B, and C may not be optimal for any specific architecture. In the final system deployment, such dimensions can be further optimized according to the chosen FCN model. We used Vivado High-Level Synthesis (HLS) to develop the GEMM accelerator, which is connected to external memory via an AXI4-Full interface to Accelerator Coherency Port (ACP). A DMA is used to communicate with the ACP and fill the accelerator buffer. Here, we use the ARM CPUs as the control unit through a separate AXI-Lite connection. The CPU is responsible for preparing and feeding correct addresses of the input and output matrices, as well as sending the start signal. Once this start signal is received, the accelerator unit accesses the input matrices, performs computations, and writes the output matrix to the designated address in the DDR RAM.

The accelerator in Figure 8 utilizes nine parallel floating-point multipliers, each of which is connected to different banks of block RAM containing portions of input from matrices A and B. This matrix partitioning helps improve the throughput of the design. The outputs of the multipliers are then summed together using an adder tree consisting of nine adders. If the output is a partial sum, it is written to Buffer C for accumulation until completion before being written back to the DRAM. For the floating-point version, all datapaths and buffers are 32 bits wide. For the DFP version, Figure 9 provides a closer look at the datapaths. Considering that DFP representation may result in different radix-point locations for the feature maps between different FCN layers, we need to shift the output results accordingly. Afterward, the output feature maps are converted to 8 bit and saturated if necessary.

5 EXPERIMENTAL RESULTS

In this section, we discuss the segmentation and recognition performance of our proposed processing pipeline. We also report the runtime performance for the embedded FPGA implementation, speedup achieved using our hardware accelerator, and comparison to an embedded GPU.

5.1 Experimental Setup

All of our experiments are performed using two well-known and publicly available iris datasets: the CASIA Interval V4 [1] and IITD [20]. Both datasets are captured using near-infrared range sensors and reflect real-world deployment conditions. The ground truth segmentation masks used in all of our experiments are obtained from IRISSEG-EP [16]. We use segmentation from Operator A for the CASIA Interval V4 dataset. For FCN training and deployment, we use the Darknet framework [30]. We fully implement our processing pipeline on the ZedBoard with Xilinx Zynq 7020 FPGA SoC with a 28-nm process node and 512 MB of DDR3 memory. The chip contains two ARM Cortex

A9 cores and PL fabric with 85K logic cells and 4.9-Mb block RAM. We also implement the flow on an embedded GPU, namely the NVIDIA Jetson TX1, for comparison. The TX1 SoC has four ARM Cortex A57 cores and 256 Maxwell CUDA cores, all with a 20-nm process node, and 4 GB of LPDDR4 memory. For both systems, our iris recognition flow is run inside an embedded Linux operating system.

5.2 Recognition Performance Comparisons to Previous Works

Although isolated evaluation of FCN models using the segmentation accuracy can be helpful in narrowing down to the most efficient set of models, they are not a sufficient indicator of the true overall recognition performance. The true trade-off between FCN model computational complexity and recognition performance can only be analyzed using an end-to-end flow. In other words, each model must be evaluated based on performance metrics such as the equal error rate (EER) and its receiver operating characteristics (ROC). Considering that end-to-end evaluation on all models explored is extremely time consuming, we select only the models from the Pareto fronts from Figure 6, which represent the most efficient models across the segmentation accuracy levels. The models on the Pareto fronts are labeled FCN0–FCN8 and FCN9–FCN19 for CASIA Interval V4 and IITD datasets, respectively. For each dataset, the labels are in decreasing order of computational complexity, as well as segmentation accuracy.

To evaluate the recognition performance of each FCN model, we perform all possible combinations of intra-class, which are different instances of the same iris, and inter-class matchings. For CASIA Interval V4, this results in approximately 9K intra-class and 6.9M inter-class comparisons. For IITD, approximately 4.8K intra-class and 5M inter-class comparisons are performed. In each matching, the HD is computed as described in Section 3.2.

Figure 10 shows the ROC curves for the two datasets. Here, the ground truth results are obtained by using the segmentation from IRISSEG-EP [16] along with the rest of our flow, which includes contour fitting, normalization, and encoding. As evidenced here, our best-performing models achieve ROC close to the ground truth. The EER along with the $\mathcal{F}$ -measure results for the models are reported in Table 5. We also provide comparison to two previous methods: CAHT [29] and IrisSeg [12]. We observe that the ground truth EER for each dataset computed using our flow is slightly lower than that reported in IrisSeg. Although we cannot provide an exact explanation for this result without full access to their experimental setup, we suspect that our contour fitting step might be the reason for the difference, as both of the studies use Daugman's normalization and encoding methods.

The results in Table 5 show that a few of our FCN models in each dataset outperform previous state-of-the-art EER results from IrisSeg [12]. For CASIA Interval V4, FCN0–FCN3 outperform IrisSeg, with FCN0 reducing the EER by almost half. For the IITD dataset, FCN9–FCN11 surpass the previous methods, with FCN9 reducing EER by more than half. However, it is interesting to note that some of our models achieve significantly higher segmentation accuracy than both CAHT and IrisSeg, while at the same time, these models underperform the previous method's recognition performance. This discrepancy can be attributed to the nature of FCN-based segmentation, which does not strongly account for fine-grain pupil and limbic boundaries labeling. This problem can throw off the contour fitting module in the next stage, producing inaccurate center points and radii. This highlights the necessity to evaluate FCN-based design using end-to-end flow rather than segmentation accuracy alone. In future work, this problem may be remedied by assigning larger loss to boundary pixels in comparison to other pixels.

Additional evidence for the necessity to perform end-to-end evaluation is between FCN9 and FCN10, where the model with more than $3\times$ computational complexity and higher segmentation accuracy performs worse in overall recognition performance. This observation is also true for

Fig. 10. ROC curves of FCN-based iris recognition pipelines with ground truth segmentation and different FCN models for the CASIA Interval V4 and IITD datasets. The x-axis shows the false acceptance rate, and the y-axis shows the false rejection rate. In the legend of each dataset, the FCN models are arranged in increasing FLOPs from bottom to top. The zoom-in axis range is [0 0.02] for both the x and y directions.

between FCN12 and FCN13. Figure 10 also verifies this observation where the ROC curves for FCN10 and FCN13 fall below those of FCN9 and FCN12, respectively.

5.3 Comparisons between DFP and Floating Point

Table 6 shows the segmentation accuracy and end-to-end recognition rate comparisons between our floating-point FCN-based pipeline and its DFP counterpart. The DFP version of each FCN model is obtained by analyzing and fine tuning the trained floating-point weights. From the results in the table, it is evident that the quantization process negatively impacts the segmentation accuracy of the models. However, in many cases, the quantization, in fact, improves the overall recognition rates. For instance, for FCN11 and FCN13, the EER improves significantly after the quantization to DFP.

5.4 Runtime Performance and Hardware Acceleration Speedup

We report the runtime performance of our FCN-based iris recognition pipelines using various FCN models in Figure 11. Due to space constraints, we only report results for FCN9–FCN16 for the

Table 5. EER and Segmentation Accuracy ($\mathcal{F}$ -measure) Comparison between Previous Approaches, Our FCN-Based Pipeline, and Ground Truth (GT)

CASIA Interval V4				IITD			
Approach	EER (%)	$\mathcal{F}$ -measure	GFLOPs	Approach	EER (%)	$\mathcal{F}$ -measure	GFLOPs
CAHT [29]	0.78	89.27	—	CAHT [29]	0.68	86.28	—
IrisSeg [12]	0.62	93.10	—	IrisSeg [12]	0.50	94.37	—
FCN5	0.94	98.75	0.016	FCN16	6.96	97.94	0.014
FCN4	0.64	98.93	0.060	FCN15	1.13	98.15	0.038
FCN3	0.50	99.06	0.132	FCN14	0.82	98.24	0.054
FCN2	0.43	99.09	0.380	FCN13	0.50	98.35	0.117
FCN1	0.42	99.14	0.513	FCN12	0.60	98.38	0.154
FCN0	0.38	99.17	1.143	FCN11	0.41	98.50	0.335
				FCN10	0.19	98.59	0.453
				FCN9	0.29	98.62	1.791
GT	0.31	—	—	GT	0.16	—	—

In each dataset, FCN models are arranged in increasing FLOPs and $\mathcal{F}$ -measure from top to bottom. Best EER and F-measure results are shown in bold.

Table 6. EER and Segmentation Accuracy ($\mathcal{F}$ -measure) Comparison between the Ground Truth (GT), Floating-Point, and DFP FCN-Based Recognition Pipelines Using the IITD Dataset

Model	Floating Point		DFP	
	EER (%)	$\mathcal{F}$ -measure	EER (%)	$\mathcal{F}$ -measure
FCN13	0.50	98.35	0.46	97.23
FCN12	0.60	98.38	0.68	96.49
FCN11	0.41	98.50	0.22	97.24
FCN10	0.19	98.59	0.23	96.97
FCN9	0.29	98.62	0.37	97.14
GT	0.16	—	—	—

Fig. 11. Runtime results for end-to-end FCN-based iris recognition pipelines based on different FCN segmentation models for the IITD dataset. Five platform configurations are reported: pure nonvectorized floating-point software (CPU Float), vectorized float-point and fixed-point software using ARM NEON instructions (CPU Vectorized Float, CPU Vectorized DFP), and hardware accelerated with floating-point and DFP accelerators (CPU+Accelerator Float, CPU+Accelerator DFP). The speedup relative to SW Float is reported on top of each bar.

Table 7. FPGA Resource Utilization for the Floating-Point and DFP Accelerators

	LUT	LUTRAM	Flip-Flop	BRAM	DSP	BUFG
Floating Point	15%	3%	9%	5%	21%	3%
DFP	13%	2%	7%	5%	5%	3%

The resources include Look-Up Tables (LUT), LUT as memory (LUTRAM), Flip-Flop Registers, Block RAM (BRAM), Digital Signal Processing units (DSP), and Global Clock Buffers (BUFG).

Fig. 12. FPGA floorplans of our synthesized accelerators and system modules.

IITD dataset. Similar trends and conclusions are observed for FCN0–FCN8 for the CASIA4 Interval dataset. Each runtime result is composed of four components, namely segmentation, contour fitting, normalization, and encoding. For each FCN model, we report results for three configurations, namely pure software, vectorized software, and hardware accelerated design using our custom accelerator. As discussed in Section 4.3, contour fitting, normalization, and encoding are always run using pure software. For contour fitting, there are small variations between different input images and FCN models; however, the average runtime is approximately constant across the different runs. Hence, the bulk of the differences among the pipelines stem from the segmentation runtimes using different FCN models.

5.4.1 Runtime Results for FPGA SoC. In comparison to nonvectorized software, vectorization using the NEON instruction allows between 2.5× and 2.8× speedup. Using our accelerator design, we achieve between 2.4× and 6.6× speedup. We observe that higher speedup is realized for larger FCN models since the fraction of runtime spent in segmentation far exceeds that of other components. For the hardware-accelerated implementation, the runtime differences between different FCN pipelines vary by up to two orders of magnitude, ranging from 0.05 to 5.3 seconds.

The resource utilization of our accelerators is reported in Table 7, and the floorplans of the designs are shown in Figure 12. As discussed earlier, since our target models vary significantly in architecture and computational requirement, we implement the accelerators using only the greatest common divisor among them, which explains the low resource utilization. However, with this design, we demonstrate that significant speedup can be achieved while only utilizing a fraction

Table 8. Runtime (in Seconds) and Energy (in Joules) Comparison for End-to-End Iris Recognition Flows between Embedded FPGA and GPU Platforms

Model	CPU+Accel (Float)		CPU+Accel (DFP)		CPU+GPU (Float)	
	Runtime (s)	Energy (J)	Runtime (s)	Energy (J)	Runtime (s)	Energy (J)
FCN13	0.67	3.35	0.57	2.85	0.77	11.55
FCN12	0.89	4.45	0.78	3.90	0.79	11.85
FCN11	1.79	8.95	1.51	7.55	0.76	11.4
FCN10	1.73	8.65	1.43	7.15	0.83	12.5
FCN9	5.32	26.6	4.20	21.0	1.06	15.9

The best runtime result for each model is shown in bold. The GPU SoC is based on a 20-nm process node, and the FPGA SoC is based on 28 nm.

of the available resource. Once a specific model is chosen, a potentially larger speedup can be achieved by optimizing the accelerator design and parameters.

As expected, we observe that overall the floating-point accelerator consumes more resources than the DFP counterpart. Specifically, the floating-point accelerator requires 4× more DSP resources than the fixed-point one. Although there is a smaller difference in LUT counts, this is due to the required shifting and saturation logic required in the DFP accelerator. For BRAM, the two accelerators utilize the same amount, as we require multiple ports for parallel multiplications and accumulations.

5.4.2 Comparison with Embedded GPU. For comparison, we also implement our iris recognition pipeline on a Jetson TX1 embedded GPU platform. Table 8 provides the runtime comparisons for the end-to-end flow between the embedded FPGA and GPU systems. The results show that the GPU performs significantly better than the FPGA platform for larger models such as FCN9 and FCN10. This performance difference can be attributed to the higher operating frequency and more computational resources such as cores and memory bandwidth on the GPU platform. This, however, results in the GPU consuming more than double the power requirement for the FPGA platform. In this case, the platform of choice is therefore dependent on the runtime and energy constraints of the target deployment. For smaller models, surprising runtime results are observed for the GPU platform. From FCN11 to FCN13, the runtime did not decrease as the models became simpler. Our profiling using Nvidia's *nvprof* and *Ninsight Systems* shows that most of the runtime is spent in GPU memory allocation and movement. This results in the GPU having better energy efficiency for larger models but significantly less efficiency for smaller ones. However, an important note is that the GPU SoC was fabricated with a more recent process node of 20 nm, which means that for the same 28-nm technology node as the FPGA system, the GPU would consume more energy than the results reported in Table 8.

6 CONCLUSION

In this work, we proposed an end-to-end iris recognition application with FCN-based segmentation. Through our profiling of the overall processing pipeline, we identified that the majority of the runtime is spent on the segmentation step, which was the FCN inference. Targeting this processing stage, we introduced a three-step SW/HW co-design methodology to cut down its runtime. First, we introduced a design space exploration for the FCN architecture to select the most efficient set of models. The exploration was performed through a grid search on several architectural parameters including the spatial dimensions of the input image. For each architecture, we evaluated its segmentation accuracy performance and the computational overheads of each FCN model. We then identified the most efficient set of models, which formed a Pareto front. Compared

to the FCN architectures from previous works, our best-performing models set new state-of-the-art segmentation accuracy on two well-known datasets, namely CASIA Iris Interval V4 and IITD, while being 50 $\times$ more resource efficient. Furthermore, we evaluated the true recognition rate of each model using the end-to-end pipeline and showed that the models outperformed the recognition rate from previous works on the two datasets. Our architectural exploration in this design process showed that a small EER increase of 0.7% can be traded off for orders of magnitude reduction in computational complexities and latency. With this set of models, we co-designed their datatype to DFP formats for hardware-friendly execution. Finally, we introduced a novel FPGA-based DFP accelerator and demonstrated a full implementation of an accelerated processing flow on an embedded FPGA SoC. We also synthesized a floating-point version of the accelerator for runtime and resources comparisons. In comparison to the onboard CPU, our accelerator is able to achieve up to 8.3 $\times$ speedup for the overall pipeline while using only a small fraction of the available FPGA resource. Finally, we provided comparisons between the FPGA system and an embedded GPU showing the different benefits of the two platforms and interesting insights for smaller FCN models. We released a MATLAB version of our iris recognition flow on Github.¹

The design methodology proposed in this work opens many doors for research on end-to-end systems similar to the demonstrated iris recognition. Future work includes further FCN optimization through pruning, accelerator support for sparse matrix, and improved contour fitting for non-circular irises using active contour.

ACKNOWLEDGMENTS

We would like to thank Dr. Soheil Hashemi for helpful discussions. We also thank NVIDIA Corporation for their generous GPU donation.

REFERENCES

- [1] Biometrics. 2010. CASIA Iris V4 Dataset. Retrieved September 1, 2018 from <http://biometrics.idealtest.org/dbDetailForUser.do?id=4>
- [2] Mohammed A. M. Abdullah, Satnam S. Dlay, Wai L. Woo, and Jonathon A. Chambers. 2017. Robust iris segmentation method based on a new active contour force with a noncircular normalization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 47, 12, 3128–3141.
- [3] Fernando Alonso-Fernandez and Josef Bigun. 2012. Iris boundaries segmentation using the generalized structure tensor. A study on the effects of image degradation. In *Proceedings of the IEEE International Conference on Biometrics: Theory, Applications, and Systems*. 426–431.
- [4] Muhammad Arsalan, Rizwan Ali Naqvi, Dong Seop Kim, Phong Ha Nguyen, Muhammad Owais, and Kang Ryoung Park. 2018. IrisDenseNet: Robust iris segmentation using densely connected fully convolutional networks in the images by visible light and near-infrared light camera sensors. *Sensors* 18, 5, 1501.
- [5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2015. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. arXiv:1511.00561.
- [6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. 2018. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40, 4, 834–848.
- [7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. arXiv:1412.7024.
- [8] John Daugman. 2007. New methods in iris recognition. *IEEE Transactions on Systems, Man, and Cybernetics* 37, 5, 1167–1175.
- [9] John Daugman. 2009. How iris recognition works. In *The Essential Guide to Image Processing*. Elsevier, 715–739.
- [10] John G. Daugman. 1993. High confidence visual recognition of persons by a test of statistical independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 11, 1148–1161.

¹<https://github.com/scale-lab/FCNiris>.

- [11] Abhishek Gangwar and Akanksha Joshi. 2016. DeepIrisNet: Deep iris representation with applications in iris recognition and cross-sensor iris recognition. In *Proceedings of the IEEE International Conference on Image Processing*. 2301–2305.
- [12] Abhishek Gangwar, Akanksha Joshi, Ashutosh Singh, Fernando Alonso-Fernandez, and Josef Bigun. 2016. IrisSeg: A fast and robust iris segmentation framework for non-ideal iris images. In *Proceedings of the IEEE International Conference on Biometrics*. 1–8.
- [13] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R. Iris Bahar, and Sherief Reda. 2017. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *Proceedings of the IEEE Design, Automation, and Test in Europe Conference and Exhibition*. 1474–1479.
- [14] Soheil Hashemi, Hokchhay Tann, Francesco Buttafuoco, and Sherief Reda. 2018. Approximate computing for biometric security systems: A case study on iris scanning. In *Proceedings of the IEEE Design, Automation, and Test in Europe Conference and Exhibition*. 319–324.
- [15] Heinz Hofbauer, Fernando Alonso-Fernandez, Josef Bigun, and Andreas Uhl. 2016. Experimental analysis regarding the influence of iris segmentation on the recognition rate. *IET Biometrics* 5, 3, 200–211.
- [16] Heinz Hofbauer, Fernando Alonso-Fernandez, Peter Wild, Josef Bigun, and Andreas Uhl. 2014. A ground truth for iris segmentation. In *Proceedings of the IEEE International Conference on Pattern Recognition*. 527–532.
- [17] Ehsaneddin Jalilian and Andreas Uhl. 2017. Iris segmentation using fully convolutional encoder–decoder networks. In *Deep Learning for Biometrics*. Springer, 133–155.
- [18] Ehsaneddin Jalilian, Andreas Uhl, and Roland Kwitt. 2017. Domain adaptation for CNN based iris segmentation. In *Proceedings of the 2017 International Conference of the Biometrics Special Interest Group (BIOSIG'17)*. 1–6.
- [19] W. K. Kong and D. Zhang. 2001. Accurate iris segmentation based on novel reflection and eyelash detection model. In *Proceedings of the IEEE International Symposium on Intelligent Multimedia, Video, and Speech Processing*. 263–266.
- [20] Ajay Kumar and Arun Passi. 2010. Comparison and combination of iris matchers for reliable personal authentication. *Pattern Recognition* 43, 3 (2010), 1016–1026.
- [21] Nianfeng Liu, Haiqing Li, Man Zhang, Jing Liu, Zhenan Sun, and Tieniu Tan. 2016. Accurate iris segmentation in non-cooperative environments using fully convolutional networks. In *Proceedings of the IEEE International Conference on Biometrics*. 1–8.
- [22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.
- [23] Mariano López, John Daugman, and Enrique Cantó. 2011. Hardware-software co-design of an iris recognition algorithm. *IET Information Security* 5, 1, 60–68.
- [24] Li Ma, Yunhong Wang, and Tieniu Tan. 2002. Iris recognition using circular symmetric filters. In *Proceedings of the IEEE International Conference on Pattern Recognition*. 414–417.
- [25] L. Masek and P. Kovesi. 2003. *MATLAB Source Code for a Biometric Identification System Based on Iris Patterns*. The School of Computer Science and Software Engineering, The University of Western Australia.
- [26] D. Petrovska and A. Mayoue. 2007. *Description and Documentation of the BioSecure Software Library*. Project No IST-2002-507634-BioSecure, Deliverable.
- [27] Ahmad Poursaberi and Babak N. Araabi. 2005. A novel iris recognition system using morphological edge detector and wavelet phase features. *ICGST International Journal on Graphics, Vision and Image Processing* 5, 6, 9–15.
- [28] Hugo Proença and Luís A. Alexandre. 2007. The NICE. I: Noisy iris challenge evaluation-part I. In *Proceedings of the IEEE International Conference on Biometrics: Theory, Applications, and Systems*. 1–4.
- [29] Christian Rathgeb, Andreas Uhl, and Peter Wild. 2012. *Iris Biometrics: From Segmentation to Template Security*. Vol. 59. Springer Science & Business Media.
- [30] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. Retrieved September 23, 2018 from <http://pjreddie.com/darknet/>.
- [31] C. J. Van Rijsbergen. 1979. *Information Retrieval* (2nd ed.). Butterworth-Heinemann.
- [32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*. 234–241.
- [33] Samir Shah and Arun Ross. 2009. Iris segmentation using geodesic active contours. *IEEE Transactions on Information Forensics and Security* 4, 4, 824–836.
- [34] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. arXiv:1412.6806.
- [35] Hokchhay Tann, Soheil Hashemi, R. Iris Bahar, and Sherief Reda. 2017. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *Proceedings of the IEEE Design Automation Conference*. 1–6.
- [36] Christel-Loïc Tisse, Lionel Martin, Lionel Torres, and Michel Robert. 2002. Person identification technique using human iris recognition. In *Proceedings of the Vision Interface Conference*, Vol. 294. 294–299.

- [37] Andreas Uhl and Peter Wild. 2012. Weighted adaptive Hough and ellipsopolar transforms for real-time iris segmentation. In *Proceedings of the IEEE International Conference on Biometrics*. 283–290.
- [38] Richard P. Wildes, Jane C. Asmuth, Gilbert L. Green, Stephen C. Hsu, Raymond J. Kolczynski, James R. Matey, and Sterling E. McBride. 1994. A system for automated iris recognition. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*. 121–128.
- [39] Guangzhu Xu and Zaifeng Zang. 2008. An efficient iris recognition system based on intersecting cortical model neural network. *International Journal of Cognitive Informatics and Natural Intelligence* 2, 3, 43–56.
- [40] Zijng Zhao and Ajay Kumar. 2015. An accurate iris segmentation framework under relaxed imaging constraints using total variation model. In *Proceedings of the IEEE International Conference on Computer Vision*. 3828–3836.
- [41] Zijng Zhao and Ajay Kumar. 2017. Towards more accurate iris recognition using deeply learned spatially corresponding features. In *Proceedings of the IEEE International Conference on Computer Vision*. 3809–3818.

Received February 2019; revised May 2019; accepted August 2019