An FPGA-based Hardware Accelerator for Scene Text Character Recognition

Luiz Antonio de Oliveira Junior
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
laoj2@cin.ufpe.br

Edna Barros
Centro de Informática
Universidade Federal de Pernambuco
Recife, Brazil
ensb@cin.ufpe.br

***Abstract*—Scene text character recognition is a challenging task in Computer Vision since natural scene images usually have cluttered background and the character's size, font, orientation, texture, brightness, and alignment in the picture are variable and non-predictable. Furthermore, most systems including scene text character recognition are usually embedded in a system on a chip (SoC), which has critical requirements, such as low latency, low area, mobility, and flexibility, at the same time that they require high accuracy. In this context, in this work we propose a heterogeneous system for embedded applications with time, area and power constraints, that combines hardware and software to accelerate a technique for scene text character recognition, based on Histogram of Oriented Gradients (HOG) for feature extraction and a neural network Extreme Learning Machine (ELM) as a classifier. The system was prototyped and experimented in the Terasic embedded platform DE2i-150 and the results showed that the system has accuracy of 65.5% in the Chars74k-15 dataset and is able to process up to 11 frames per second, having a good trade-off between processing time and accuracy in embedded environments. Moreover, it occupies only 11% logic elements of the Altera Cyclone IV FPGA, enabling its use in embedded systems.**

***Index Terms*—Character Recognition, FPGA, HOG, ELM, Computer Vision.**

I. Introduction

Scene text character recognition (STCR) approaches in Computer Vision aim to develop systems that can automatically recognize characters information in natural scene images. Although this kind of systems have been widely used in industrial and commercial applications, such as mobile navigation and scene understanding [1] and license plate recognition, STCR still is a challenging computational problem, since natural images usually have cluttered background and the character's size, font, orientation, texture, brightness, and alignment in the image are variable and non-predictable. Figure 1 shows how characters from different classes look similar due to the natural scene image characteristics aforementioned.

Three common steps of an STCR algorithm are (1) pre-processing, (2) feature extraction and (3) character classification.

During the pre-processing phase, algorithms are used to improve the quality of visual information by keeping only the information related to the character itself and removing noise-related information. The most commonly used preprocessing techniques are color space conversion and threshold, morphological operations such as erosion and dilation for noise removal and reconstruction of objects; analysis of connected components used to remove noise or to isolate the character of other objects in the image. Also, most feature extraction algorithms, such as Histogram of Oriented Gradients (HOG), [2] and Convolutional Neural Networks (CNN) [3] require that all input images are of a specific size. However, character images in actual scenes can range from tens to thousands of pixels, depending on the distance between the character and the camera and the type of the source, for example. Therefore, it is necessary to apply some image resizing method, such as the spline cubic interpolation algorithms and bicubic interpolation [4].

Fig. 1. Examples of natural scenes character images from the Chars74k dataset.

Given an input preprocessed image, the step of extracting features obtains image characteristics that are intrinsic to the character class to which it belongs. In this context, a good extractor must produce similar attributes for all characters of the same type, even if there are: (1) variations in the size, (2) rotation and (3) translation. Three main types of feature extractors have been used [5]: Histogram of Oriented Gradients (HOG); Convolutional Neural Networks (CNN); and Structural character characteristics [6].

In the step of character recognition, a classifier receives the image characteristics vector and classifies the image in its respective class. The most used classification techniques are *Support Vector Machine* (SVM), Convolutional Neural

978-1-5386-4756-1/18/$31.00

Networks (CNN), Random Forests, and Neural Networks.

In this work, we propose a heterogeneous architecture (processor and FPGA) to accelerate an STCR technique, to address the limitations of the existing software-based and hardware-based techniques. The main contributions of this work include: (1) a novel method for SCTR, (2) an FPGA-based architecture for bicubic interpolation algorithm and (3) an STCR architecture that is suitable for embedded applications.

This paper is organized as follows: in section 2, we describe related works. In section 3, we introduce our proposed technique for STCR. In section 4 we present the proposed CPU/FPGA-based architecture implementing the STCR technique. Finally, we conclude the in section 5.

II. Related Works

To tackle the challenges of the STCR problem, many efforts have been devoted to developing accurate techniques. Campos et al. [7] introduced the scene text characters benchmark Chars74K and showed that commercial Optical Character Recognition (OCR) tools do not have a good performance when used to classify natural scenes characters. In this work, the authors addressed the STCR problem on the Chars74k-15 dataset by testing combinations of feature extractors and classifiers, such as Multiple Kernel Learning (MKL). The techniques were implemented only in software and the processing times were not mentioned. Yi et al. [1] proposed a scene text recognition method for Android applications that employs stroke configuration map as feature representation and a cascade AdaBoost learning scheme to train the classifier. The system was evaluated on the Chars74k dataset and was implemented on a Samsung Galaxy II, and the authors reported an approximated time of 1 second to process each frame.

The authors in [8] proposed a new method for scene character recognition that uses a CNN to extract features from the input image and uses these features to train stroke detectors. Although the authors reported an accuracy of 76.1% on the Chars74k-15 database, this method may not be suitable for embedded systems, since CNNs are both memory and computationally intensive [9], such that the execution time can be very high if there are not enough resources.

The authors in [10] employ Fisher Vectors derived from Gaussian Mixture Models (GMM) and a linear classifier on character recognition. The results showed that this method is able to achieve up to 74.8% of accuracy on Chars74k-15 and has a inference time of 12.5 ms. However, the time analysis was performed using a PC with a Intel i7 CPU and 12GB of RAM, therefore this method can not be fairly compared with the ones implemented in embedded environments.

Most of the STCR techniques proposes a combination of a feature extractor and a classifier aiming to achieve high accuracy, and they do not regard to reduce the processing time, a critical metric for real-time applications. Moreover, these techniques are usually executing on a high-performance CPU, which can be not appropriated for embedded applications. On the other hand, a growing number of systems that need scene text character recognition are usually embedded and have critical requirements. Such systems must have low latency, low area, mobility, and flexibility, at the same time that they require high accuracies, such as autonomous vehicles [11] and walking assistants for blind people [12].

Some recent works have used FPGA to accelerate character recognition techniques [13], [14] and the processing times are indeed lower than the software approaches. In [13], the proposed architecture can classify about 12 characters per second, whereas in [14] the proposed architecture processes one image per 6ms. These works, however, have been evaluated using MINIST dataset, a very basic dataset of handwritten digits or some proprietary datasets with a uniform background, size, and fonts.

III. The STCR Proposed Method

The biggest challenge in STCR is to choose the algorithms for each STCR step maintaining a compromise between accuracy and processing time. In this context, we propose a novel technique for STCR, based on resize, HOG and ELM neural network. Figure 2 illustrates the execution flow of the proposed approach. The proposed system receives as input

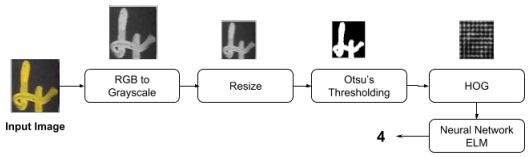

Fig. 2. Stream of execution of the proposed STCR system

an RGB image. In the pre-processing step, the input image is first converted to the gray-scale color space. The gray-scale image is then resized to a default size of 128×128 pixels, to standardize the size of the input images. Finally, the image size of 128×128 pixels is thresholded, using the Otsu [15] technique, to segment the *background* character. In the next step, a Characteristic Extractor based HOG [2] is applied to the threshold image with the purpose of generating a 1296 dimensionality vector containing the characteristics of the image character. Finally, this feature vector is the input of a neural network Extreme Learning Machine (ELM) [16], which classifies the character.

In order to verify the accuracy of the proposed approach we used the Chars74K-15 benchmark. In this sense, we implemented the proposed approach in C ++, using a fixed point in Q1.8.8 format to represent the real numbers, under an Ubuntu 16.04 LTS OS, running on an Intel i7-4500U dual-core 3GHz, with 8GB of RAM and 1TB hard drive.

The parameters in the ELM neural networks are the number of neurons in the hidden layer, L and the activation function g. Therefore, we performed two experiments for finding the number of neurons in the hidden layer and the activation function that resulted in the best performance of the proposed technique: (1) ELM with linear activation function, ie $g(x) = x$, and (2) ELM with hyperbolic tangent activation

function that is $g(x) = tanh(x)$. The results indicate that the best accuracy occurs when we use an ELM neural network with 18000 neurons in the hidden layer, $L = 18000$, and the hyperbolic tangent activation function, $g(x) = tanh(x)$. In this configuration, the system has the accuracy of 67.2%, similar or higher accuracy in comparison with related works.

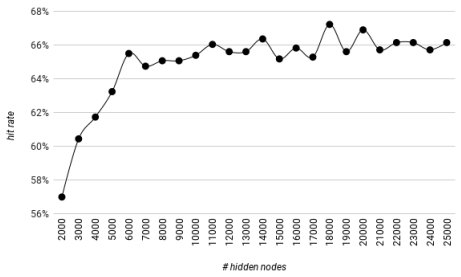

Fig. 3. System accuracy as a function of the number of neurons of the ELM hidden layer

IV. The Heterogeneous Architecture for the STCR Technique

The first step was to define which tasks to perform in software (in the CPU) and which ones to execute in hardware (in the FPGA). For this purpose, we implemented the STCR method in C ++ in the target architecture and measured the average time each step took to execute on the Intel Atom N2600 processor, to identify the more time-consuming functions to be hardware accelerated. For this purpose we used *profiling* tool Gprof [17] and the functions of the *time.h* Library. The result of the analysis shows that the *Resize* function is the bottleneck, consuming 71.75% of the total execution time.

In this way, we developed a hardware module, the Bicubic Interpolation Accelerator (BIA), to accelerate the *Resize* step and integrated it into the target platform for accelerating the STCR algorithm.

Figure 4 also illustrates the execution flow of the STCR algorithm and the division of tasks between hardware and software. The tasks running on the ATOM were implemented in C++, while the tasks running on the FPGA were specified in RTL-level SystemVerilog.

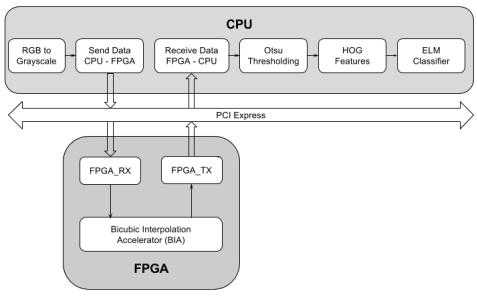

Fig. 4. Proposed Architecture

According to figure 4, the proposed system receives an input RGB image, which is converted in the Grayscale color system. In the second step, *Send Data CPU - FPGA*, the *grayscale* image is converted into a stream of untyped bytes, which are sent to the FPGA via PCI-E bus. The FPGA_RX module controls reading transactions on the PCI-E bus to receive the stream of bytes. Then, the BIA module resizes the gray-scale image to an image size of 128×128 pixels, through the bicubic interpolation algorithm. The resized image is sent back to the software in through the FPGA_TX module. The module *Receive Data FPGA - CPU* converts the stream of bytes sent by the FPGA back to a typed image. Finally, the HOG-based feature extractor is applied to the image, and the classifier based on an EML neural network classifies the input image into one of the 62 possible characters.

A. Resize Technique

As mentioned, we use the bicubic interpolation technique to resize input images of any size to a standard size of 128×128 pixels. An interpolation function in one dimension can be defined as a convolution between a discrete function g and an interpolation kernel r (equation 1). The general form of a resized image through an interpolation function is given by equation 1:

$$f(x') = [r * g](x') = \sum_{m=-\infty}^{+\infty} r(dx - m).g(x + m) \quad (1)$$

where x' is the point where we want to estimate the value of f, where $x = \lfloor x' \rfloor$ and $dx = x' - x$.

The main difference between the various interpolation functions is the convolution *kernel*. After analyzing different kernels we chose the bicubic interpolation method, which is composed of third degree polynomials defined in the intervals: $(-2, -1)$, $(-1, 0)$, $(0, 1)$ and $(1, 2)$. Outside this interval, it assumes the value zero (equation 2).

$$r_{cub}(x, a) = \begin{cases} (-a+2)|x|^3 + (a-3)|x|^2 - 1, & \text{if } 0 \leq |x| < 1, \\ -a|x|^3 + 5a|x|^2 - 8a|x| + 4a, & \text{if } 1 \leq |x| < 2, \\ 0, & \text{if } |x| \geq 2 \end{cases} \quad (2)$$

such as $r_cub = 0$ when $|x|2$. Only four discrete values of $g(u)$ are required for the convolution operation. Therefore, the equation 1 can be rewritten as described below.

$$f(x') = [r * g](x') = \sum_{m=-1}^{+2} r_{cub}(dx - m).g(x + m) \quad (3)$$

Equation 3 defines the one-dimensional case. As digital images are two-dimensional, we need to redefine the interpolation function for the 2D case. So let I be an input image of $a \times b$ pixels and $R_{cub2D}(x, y) = r_{cub}(x)r_{cub}(y)$ or *kernel* of the 2-D bicubic interpolation, we can define the resized image I_{red} (of $p \times q$ pixels) through the following equation:

$$I_{red}(x', y') = \sum_{m=-1}^{+2} \sum_{n=-1}^{+2} I(x+m, y+n) r_{cub}(dx-m) r_{cub}(dy-n) \quad (4)$$

in which (x', y') are the coordinates of any one pixel in the output image, where $(x, y) = (\lfloor x' * \frac{a}{p} \rfloor, \lfloor y' * \frac{b}{q} \rfloor)$, $dx = x' * \frac{a}{p} - x$ e $dy = y' * \frac{b}{q} - y$.

This resizing method is described in algorithm 1. In this work, the number of rows and columns in the output images,O_x and O_y, are 128. Algorithm 1 describes a convolution between the input image and the kernel using a neighborhood of 4×4 pixels (window) to produce each pixel of the image output.

Algorithm 1 Image resizing using the bicubic interpolation method

Input: I, the original image; I_x e I_y, the number of rows and columns of the original image, respectively; O_x e O_y, the number of rows and columns in the output image
Output: O, the resized image;

```
for i ← 0 : O_x do
    for j ← 0 : O_y do  x ← ⌊i * I_x/O_x⌋   y ← ⌊j * I_y/O_y⌋   O(i,j) ← 0
        for m ← −1 : 2 do
            for n ← −1 : 2 do  dx ← i * I_x/O_x − y   dy ← j * I_y/O_y − x
                if x + m < I_x and y + n < I_y then  O(i,j) ← O(i,j) + I(x+m, y+n) * r_cub(dx−m)|_{a=0.5} * r_cub(dy−n)|_{a=0.5}
                end if
            end for
        end for
    end for
end for
```

Figure 5 illustrate the process described by algorithm 1. For each pixel of the output image, a region of 4×4 pixels adjacent to the input image (window) is selected, and each pixel in that region is multiplied by a corresponding weight, as defined by the interpolation kernel.

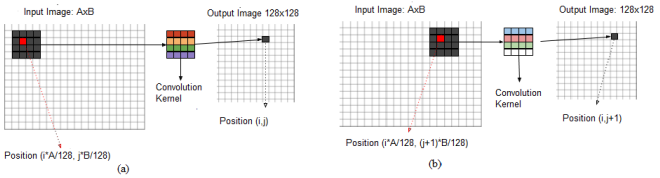

Fig. 5. Bicubic Interpolation Method

Although interpolation is quite similar to filtering, there are two main differences to be considered. In bicubic interpolation for large input images, distant points in the input image may result in points near the output image (see figures 5-a and 5-b). In the case of filtering, near points in the input image always generate near points in the output image. Additionally, in bicubic interpolation the convolution kernel values, r_{cub}, vary at each iteration depending on the values of dx and dy, which can be observed in line 15 of the algorithm 1. In the case of filtering, the filter values are static.

The shaded region over the input image represents the neighborhood of 4×4 pixels (window) used to produce a pixel in the output image. It can be seen in Figure 5 that near points in the output image ((i, j) and $(i, j+1)$) are generated by distant points in the input image. Additionally, for two different output pixels, the kernel values are not the same.

B. The BIA Module

The Bicubic Interpolation Accelerator is the proposed hardware module for resizing an input image of any size to the size of 128 x 128 pixels, through the bicubic interpolation algorithm. This module was designed in RTL-level and prototyped in FPGA.

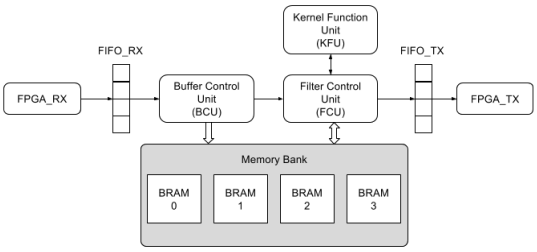

Fig. 6. Bicubic Interpolation Accelerator architecture.

The proposed architecture for the BIA module consists of three processing units: (1) Buffer Control Unit (BCU), (2) Filter Control Unit (FCU) and (3) Kernel Function Unit (KFU) and four on-chip memory banks of the BRAM type. This architecture was designed to be integrated with the processor as a standalone unit, flexible and able to adapt to various sizes of input images. The module was implemented as a pipeline and can provide one output pixel every six cycles of a 50 MHz clock. The BIA exploits the parallelism of the bicubic interpolation algorithm by processing 4 pixels (of the 16 pixels required to compute an output pixel) at each clock cycle. According to algorithm 1, given an input image I of any size, the method constructs the output image iteratively, calculating each output pixel, through the equation 4.

Two fundamental features increase the complexity when designing the bicubic interpolation hardware accelerator. In bicubic interpolation, the filter weights, i.e., the values of r_{cub}, are calculated dynamically for each input pixel, so it is not possible to use optimizations based on static filters. To cope with this, we have developed the *Kernel Function Unit*. For computing a given pixel of the output image, the bicubic interpolation algorithm accesses 16 pixels of the input image, which is the neighborhood of 4×4 pixels described before and indicated by the loops of algorithm 1. From this point, we will refer to this region as a window.

Unlike a filtering process, the coordinates of the pixels contained in the window are calculated dynamically for each output pixel, since they depend on the dimensions of the input image and the coordinates of the output pixel. Therefore, the use of simple input image storage mechanisms (i.e *shift registers*), widely used in filtering operations, is not suitable for an interpolation algorithm.

The Buffer Control Unit controls the storage of the input image. Input pixels are stored in 4 BRAMs since this storage structure are on-chip memories, and their read latency is only one clock cycle. At each clock cycle, this unit receives 4

consecutive pixels from the input image, and stores each of them in a different BRAM. That is, let p_i be the i-th pixel of the input image, p_i is stored in the $(i \mod 4)$th BRAM, in the address $i/4$. This pixel storage arrangement allows the Filter Control Unit module to access four consecutive pixels of the input image in parallel since two or more consecutive pixels will never be stored in the same BRAM. Since each BRAM stores, at maximum, one fourth of the input image, we set their sizes to $\lceil \frac{M*N}{4} \rceil$ bytes, where M and N are the height and width of the largest image in the training and test set. Therefore, in our experiments each BRAM has 81KB.

The Kernel Function Unit (KFU) is the unit responsible for dynamically computing weights for each input pixel, according to equation 2, where $a = 0.5$. That is, given an input value (x, y), this module returns the value of $r_{cub}(x) * r_{cub}(y)$. The input values of the function r_{cub}, $dx - m$ and $dy - n$ are real numbers. Therefore, in the hardware module, we represent the numbers in the fixed-point format $Q1.8.8$. That is, 17-bit registers are used to store numerical values, in which the first most significant bit is used as the signal bit, the next eight most significant bits represent the integer part of the number, and the least significant 8 bits represent the fractional part. Since we are using a fixed-point representation for $dx - m \in [-2, 2]$ and $dy - n \in [-2, 2]$, all possible input values of the KFU belong to the range $-512..512$ ($512 = 2 << 8$), so there are only 1025 possible outputs of r_{cub} (1025 is the number of integers in the range $-512..512$). For this reason, we are using a look-up table (LUT) to implement the kernel function, $r_{cub}(x)$. The main advantages of using a LUT in this unit are: (1) to avoid performing a large number of multiplications and additions for each input value and (2) the output of the module is available with the latency of one clock cycle.

As the BIA module exploits the 4-pixel parallelism in a window row, the KFU unit inputs are: (1) 4 ordered pairs, which correspond to the values of (dx, dy) for each pixel, and (2) the value of $m \in [-1, 2]$, which depends on which line is currently being processed. The value of n is known and fixed for all entries since we are handling all the pixels of a line in parallel. The Filter Control Unit (FCU) calculates for each output pixel the coordinates of the pixels belonging to the window; reads the BRAM memory pixels, based on the coordinates calculated; calculates the input values of the Kernel Function Unit, dx, dy and m; and calculates the input values of the Kernel Function Unit, dx, dy and m. In the design of this unit, we explore hardware parallelism by computing a line (i.e., 4 out of 16 pixels) of the 4×4 pixel window in a clock cycle. The operation of the FCU is implemented as a finite state machine. It controls the reading of the pixels of the input image, i.e., how the FCU controls access to BRAM memories and how to control the multiplication between the pixels read and the corresponding kernel weights.

V. Results

The proposed approach was evaluated considering the accuracy and execution time. We performed experiments in two different scenarios. In the first one, we implement the proposed STCR only in software (C++) under Ubuntu 14.04, running on an Intel ATOM dual-core 1.6GHz, with 2GB of RAM at the DE2i-150 platform. For a fair comparison, the software implementation was also based on fixed point of format Q1.8.8 and the values of $r_{cub}(x)$ were computed through a *hash table*, to simulate a *look-up table*. In the second scenario, the part in software (C ++) runs on the ATOM processor. The hardware part was prototyped in a Cyclone IV. Communication between the processor and the FPGA is done through a PCI-E bus. Besides, we use the RIFFA framework [18] to interface the FPGA with the CPU. Both scenarios were evaluated using the Chars74K-15 benchmark [7]. It contains 930 training images and 930 test images, divided into 62 classes. Therefore, in all subsequent tests, it is assumed that Chars74K-15 was used.

The ELM parameters must be adjusted for the DE2I platform. To chose the number of neurons that results in a better performance and accuracy trade-off, we evaluate the impact of the number of neurons in the hidden layer on the average time for classifying an image in the target platform. Figure 7 summarizes the result. According to figure 7, the classification time of the ELM neural network grows approximately linearly with the number of neurons in the hidden layer. A good performance trade-off was obtained using 6000 neurons in the hidden layer. With this configuration, the system accuracy was 65.5%, a better result in a comparison of recent works, but also only 1.7 percentage points lower than the rate considering 18,000 neurons (for I7 based platform). Moreover, in this configuration, the average time for classifying an image was 50.7 ms, which is three times smaller than the average classification time of an ELM neural network with $L = 18000$ (151.9 ms).

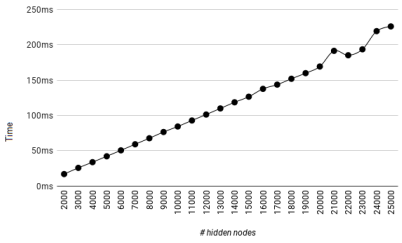

Fig. 7. Classification timing as a function of number of neurons.

For evaluating the average execution time of the proposed hybrid architecture for the STCR method, the individual times of each task were measured using the profiling tool Gprof [17] and the C library time.h. The execution time for the module *Resize* shown in table I is the CPU-FPGA communication time plus the BIA execution time. The results indicate that the BIA module is able to resize images up to 137 times faster than the C++ implementation.

Since we use a fixed point for representing real numbers in the *Resize* step, we evaluated its impact on the system's accuracy. Figure 8 indicates that there is no significant impact on the accuracy with the use of fixed point.

STCR Task	Atom N2600 (ms)	Atom + FPGA (ms)
RGB to Grayscale	1.1	1.1
Resize	220.34	**1.6**
Otsu	1.2	1.2
HOG	30.9	30.9
ELM	50.7	50.7

TABLE I
AVERAGE TIME EXECUTION FOR SOFTWARE ONLY AND HYBRID ARCHITECTURE

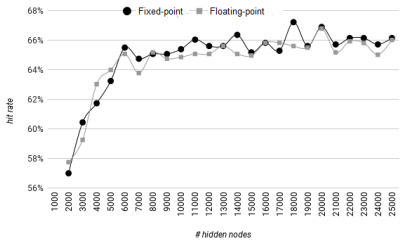

Fig. 8. System Accuracy for fixed-point and floating-point implementations

The proposed system was also compared to related works for the Chars74K-15 database. The table II indicates that the proposed architecture has a good trade-off between processing time and accuracy in embedded environments, being able to process up to 11 frames per second. The proposed CPU-FPGA hybrid architecture is 3.6 times faster than software-only implementation and up to 11.7 times faster than the another method implemented in an embedded system [1].

Meth	time (s)	Arch	Accuracy(%)
MKL [7]	-	-	55.8
Tensor+NN [19]	-	-	57.1
Fisher Vectors [10]	0.012	Intel i7	74.8
CNN + Stroke [8]	-	-	76.1
HOG+Adaboost [1]	1	ARM-A9 +GPU	60.0
Proposed in SW	0.308	Intel Atom	65.5
Proposed in HW-SW	0.085	Atom+FPGA	65.5

TABLE II
COMPARISON STCR APPROACHES USING DATASET CHARS74K-15

VI. CONCLUSION

In this work, we present a novel technique based on resize, HOG and ELM for character recognition in natural scenes (STCR). This technique achieves an accuracy of 65.5% in the dataset Chars74K-15, a good result compared with related works in an embedded environment. For accelerating the frame rate, we developed the BIA module, an accelerator for image resizing based on Bicubic Interpolation. The whole system was prototyped on the DE2i-150 embedded platform, which includes an ATOM processor and FPGA. The heterogeneous architecture is up to 11.7 times faster than the related works and can process up to 11 frames per second. The BIA module occupies only 11% of the FPGA area so that it can be used it in embedded systems with time and area constraints.

As future work, more FPGA accelerators can be developed for the other time-consuming parts of the algorithm, such as the ELM Neural Network and the HOG method. The proposed STCR system can be integrated to a Scene Text Localization system to accelerate text recognition in natural end-to-end scenes. Finally, because the BIA is a very flexible unit, it can be used to accelerate any computer vision system that needs image resizing.

REFERENCES

[1] C. Yi and Y. Tian, "Scene text recognition in mobile applications by character descriptor and structure configuration," *IEEE Transactions on Image Processing*, vol. 23, no. 7, pp. 2972–2982, 2014.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[4] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE transactions on acoustics, speech, and signal processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[5] C. Chen, D.-H. Wang, and H. Wang, "Scene character and text recognition: The state-of-the-art," in *International Conference on Image and Graphics*. Springer, 2015, pp. 310–320.

[6] C. Shi, C. Wang, B. Xiao, S. Gao, and J. Hu, "End-to-end scene text recognition using tree-structured models," *Pattern Recognition*, vol. 47, no. 9, pp. 2853–2866, 2014.

[7] T. E. de Campos, B. R. Babu, and M. Varma, "Character Recognition in Natural Images," *Visapp (2)*, pp. 273–280, 2009.

[8] Z. Zhang, H. Wang, S. Liu, and B. Xiao, "Deep contextual stroke pooling for scene character recognition," *IEEE Access*, vol. 6, pp. 16 454–16 463, 2018.

[9] C. Zhang and V. Prasanna, "Frequency domain acceleration of convolutional neural networks on cpu-fpga shared memory system," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 35–44.

[10] C. Shi, Y. Wang, F. Jia, K. He, C. Wang, and B. Xiao, "Fisher vector for scene character recognition," *Pattern Recogn.*, vol. 72, no. C, pp. 1–14, Dec. 2017. [Online]. Available: https://doi.org/10.1016/j.patcog.2017.06.022

[11] X. Rong, C. Yi, and Y. Tian, "Recognizing Text-based Traffic Guide Panels with Cascaded Localization Network," *European Conference on Computer Vision*, pp. 1–14, 2016.

[12] M. Aggravi, A. Colombo, D. Fontanelli, A. Giannitrapani, D. Macii, F. Moro, P. Nazemzadeh, L. Palopoli, R. Passerone, D. Prattichizzo, and Others, "A Smart Walking Assistant for Safe Navigation in Complex Indoor Environments," in *Ambient Assisted Living*. Springer, 2015, pp. 487–497.

[13] K. Sanni, G. Garreau, J. L. Molin, and A. G. Andreou, "FPGA implementation of a Deep Belief Network architecture for character recognition using stochastic computation," in *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, 2015, pp. 1–5.

[14] H. Zho, G. Zhu, and Y. Peng, "A RMB optical character recognition system using FPGA," in *2016 IEEE International Conference on Signal and Image Processing (ICSIP)*, 2016, pp. 539–542.

[15] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, jan 1979.

[16] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[17] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *ACM Sigplan Notices*, vol. 17, no. 6. ACM, 1982, pp. 120–126.

[18] M. Jacobsen, D. Richmond, M. Hogains, and R. Kastner, "RIFFA 2.1: A reusable integration framework for FPGA accelerators," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 8, no. 4, p. 22, 2015.

[19] M. Ali and H. Foroosh, "Natural Scene Character Recognition Without Dependency on Specific Features." in *VISAPP (2)*, 2015, pp. 368–376.