IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 27, 2020, accepted July 25, 2020, date of publication July 29, 2020, date of current version August 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3012900

BioCNN: A Hardware Inference Engine for EEG-Based Emotion Detection

HECTOR A. GONZALEZ[1], (Student Member, IEEE), SHAHZAD MUZAFFAR[2], (Student Member, IEEE), JERALD YOO[3,4], (Senior Member, IEEE), AND IBRAHIM M. ELFADEL[2,5], (Senior Member, IEEE)

[1]Chair of Highly Parallel VLSI-Systems and Neuro-Microelectronics, Technische Universität Dresden, 01069 Dresden, Germany
[2]Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates
[3]Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077
[4]The N.1 Institute for Health, Singapore 117456
[5]Center for Cyber Physical Systems, Khalifa University, Abu Dhabi, United Arab Emirates

Corresponding author: Ibrahim M. Elfadel (ibrahim.elfadel@ku.ac.ae)

The experimental procedures involving human subjects described in this paper have been approved by the Research Ethics Committee at Khalifa University under Protocol #H17-003.

ABSTRACT EEG-based emotion classifiers have the potential of significantly improving the social integration of patients suffering from neurological disorders such as Amyotrophic Lateral Sclerosis or the acute stages of Alzheimer's disease. Emotion classifiers have historically used software on general-purpose computers and operating under off-line conditions. Yet the wearability of such classifiers is a must if they are to enable the socialization of critical-care patients. Such wearability requires the use of low-power hardware accelerators that would enable near real-time classification and extended periods of operations. In this article, we architect, design, implement, and test a handcrafted, hardware Convolutional Neural Network, named BioCNN, optimized for EEG-based emotion detection and other bio-medical applications. The EEG signals are generated using a low-cost, off-the-shelf device, namely, Emotiv Epoc+, and then denoised and pre-processed ahead of their use by BioCNN. For training and testing, BioCNN uses three repositories of emotion classification datasets, including the publicly available DEAP and DREAMER datasets, along with an original dataset collected in-house from 5 healthy subjects using standard visual stimuli. A subject-specific training approach is used under TensorFlow to train BioCNN, which is implemented using the Digilent Atlys Board with a low-cost Spartan-6 FPGA. The experimental results show a competitive energy efficiency of 11 $GOps/W$, a throughput of 1.65 $GOps$ that is in line with the real-time specification of a wearable device, and a latency of less than 1 ms, which is smaller than the 150 ms required for human interaction times. Its emotion inference accuracy is competitive with the top software-based emotion detectors.

INDEX TERMS Emotion recognition, EEG, FPGA, machine learning, hardware accelerator, edge AI, convolutional neural networks, hardware parallelism, pipelining.

I. INTRODUCTION

Patients suffering from Amyotrophic Lateral Sclerosis (ALS) or the late stages of Alzheimer's disease are in a locked-in emotional state that prevents them from using their facial features to express emotions. On the other hand, the brain EEG signals of such patients are not impacted by such state and continue to contain the information needed to detect emotional content. The main goal of this article is to prove the feasibility of a wearable, small footprint, EEG-based machine-learning device to help these patients communicate their emotions in real time to their social environment, particularly their families and care providers. Such device is built around a low-power, FPGA-implemented emotion classifier with competitive classification accuracies for both the valence and arousal of the classified emotion.

Now multiple efforts have been made to design EEG-based *software* classifiers for emotions ranging from shallow [1] to deep models, the latter including hybrid combinations of convolutional neural networks (CNN) for extracting EEG features and recurrent neural networks (RNN) for analysing the EEG time series [2], [3], or even an ensemble of CNN models applied to sub-sampled versions of the same signal to mitigate EEG non-stationarity [4]. More recent approaches have been biologically inspired and used models such as spiking neural networks (SNN) to include spatio-temporal awareness captured from the EEG data [5]. Prior studies have either used raw EEG data [6], [7] or specialized EEG features

The associate editor coordinating the review of this manuscript and approving it for publication was Diego Oliva.

such as differential entropy (DE) [8], higher order crossings (HOC) [9], asymmetrical indices (AIS) [8], or power spectral distribution (PSD) [1]. Depending on the composition of the training set, there are two different training paradigms for emotion classification: subject-independent [10], [11] and subject-dependent [1], [9]. The machine-learning accelerator of this article can of course work in either paradigm. But a distinct contribution of this article is to illustrate the use of a more recent approach to training that combines the advantages of both [12].

Despite the increasing research interest in AI-based edge bio-processors [13], only one single EEG-based hardware classifier of emotions has been reported [14]. Even though it uses a CNN, our approach has several distinguishing features, including:

1) *Hardware architecture:* Our architecture uses aggressive pipelining to minimize memory footprint, to cohesively bind intermediate values between layers during their parallel execution, and to improve throughput with a 14-channel EEG system. The 6-channel EEG online system proposed in [14], uses global buffers for storage instead of SRAM, but no pipelining is reported in the architecture.
2) *Resource re-use:* Our pipelined architecture achieves a smaller relative footprint, by improving compute resource usage with such unique features as CNN kernel queuing and data buffer swapping, which takes full advantage of the low data rate of the EEG sensor.
3) *Feature extraction*: Our approach fully exploits the CNN properties by providing smoothened PSD features with correlation among consecutive frequency bins. On the other hand, the CNN reported in [14] skips these considerations by using traditional frequency spectrum, sample entropy, and Asymetrical Indices [8].
4) *Unbiased validation*: In order to ensure comparable results with the state-of-the-art, we validate our hardware system on the complete DEAP dataset. In [14], samples whose manual ratings are allocated in the center of the emotion circumplex are excluded, which prevents the comparison of the DEAP accuracy with the state-of-the-art.
5) *Diversified validation*: We also validate our hardware system on the Frequency eXpression Dataset (FEXD), which is another byproduct of this article. FEXD constitutes the largest EEG frequency data repository for emotion detection using diverse visual stimuli such as videos or pictures from the International Affective Picture System (IAPS). It includes more isolated evoked experiences and ensures the classifier functionality on different types of visual stimuli.
6) *Hardware accuracies*: To the best of our knowledge, the hardware results we report in this article seems to be the very first in providing hardware accuracies for the arousal binary detection problem. The binary classification of [14] only includes the valence dimension.

One important contribution of our work is the benchmarking of several machine learning algorithms for emotion detection under the same conditions so as to rigorously justify the BioCNN design decisions. Convolutional Neural Networks (CNNs) are selected because of their reliability, flexibility and ability to extract complex features from data. CNNs are extensively used in many applications, including computer vision, signal processing, natural language processing, language translation, and any other machine-learning task, involving large volumes of data with spatially correlated features. For the particular application of EEG-based emotion detection, CNNs provide the highest level of detection accuracy among various machine-learning algorithms, such as support-vector machines, decision trees, and random forests. In this article, we architect, design, implement, and test in hardware a specialized CNN, called BioCNN, that is optimized for biomedical applications. BioCNN embodies all the main characteristics required in an AI-based edge bio-processor. It is robust and reliable, operates in real time, consumes a modest amount of power, and has the algorithmic potential to be applied in ECG diagnostic, blood pressure monitoring, intelligent hearing aid application, and EEG-based emotion detection [13]. The main focus in the BioCNN design is optimizing wearability, reliability, area, low-power, and interactivity. All these features are achieved by exploiting the relaxed latency constraints in biomedical applications. The fine-grained flexibility of the proposed BioCNN architecture and the internal cohesion between its layers are not easily achieved using a High-Level Synthesis (HLS) tool such as Mentor's Catapult.

To summarize, our major contributions in this article are as follows:

1) We rigorously justify the use of CNN for emotion detection using extensive benchmarking against other machine learning algorithms on both public-domain and in-house datasets.
2) We report on BioCNN, a 14-channel, hardware system for emotion detection using EEG signals and a compact, low-power, energy-efficient FPGA implementation of a CNN inference engine. The system is designed to operate in real-time on a constrained edge node.
3) We introduce, implement, and test a novel, holistic architecture for CNN hardware implementation, including input staging, convolutions, maxpooling, and output classification. The architecture is based on aggressive pipelining and hardware parallelism with the dual goal of maximizing resource re-use and minimizing memory footprint.
4) We report on two novel algorithms for pipelined maxpooling and serialized classification processing that are amenable to compact, energy-efficient hardware implementation.

5) We report on the experimental results of implementing BioCNN on a constrained edge node, namely, Xilinx Altys FPGA, showing its competitive features in terms of inference accuracy, low-power consumption, energy efficiency, real-time operation, and re-use of logic resources.

This article is a major expansion of our upcoming, four-page ISCAS 2020 publication [15] with entirely new sections II, III, and VI, full hardware description in Section IV, including *all* the processing stages, and detailed functional and runtime verification in Section V.

The remainder of this article is organized as follows. Section II tackles the emotion detection problem, data collection and the feature extraction process. Section III explains algorithm selection whereas Section IV presents the implementation details of the BioCNN architecture. Section V gives the full details of the BioCNN experimental results and compare them with the state of the art. The proposed BioCNN prototype makes extensive use of resource re-utilization whose various design options are extensively evaluated in Section VI. Section VII discusses the future outlook of BioCNN, and the paper is concluded in Section VIII.

II. THE EMOTION DETECTION PROBLEM

An emotion is defined as an episode of interrelated, synchronized changes in the states of all or most of the organismic subsystems in response to the evaluation of an external or internal stimulus event as relevant to major concerns of the organism [16]. The organismic subsystems are the central nervous system (CNS), neuro-endocrine system (NES), autonomic nervous system (ANS), and somatic nervous system (SNS). Data collection in emotion research is based on eliciting the emotional states through external stimulus using pictures selected from the International Affective Picture System (IAPS), and measuring the changes in the CNS with EEG sensors. The CNS is a good candidate for measuring emotional features because of its involvement in the information processing, the execution and the monitoring of the emotions.

In order to label the samples with the reference classes, a discrete model called ''Circumplex of Emotions'' [17] is used to represent all possible emotional states as a function of two continuous dimensions: (1) Valence, which provides a numerical scale for measuring how pleasant an episode is; and (2) Arousal, which provides a numerical scale for measuring the emotion impact or strength. The evoked experiences are mapped onto the circumplex as illustrated in Fig. 1 using a discretized version of the manual feedback given by each participant during the data collection session. These labels are used for classifier design using supervised learning.

A. DATA COLLECTION

DEAP [1] and DREAMER [10] are two state-of-the-art datasets for emotion detection. DEAP made available the manual ratings and the EEG activity of 32 subjects watching

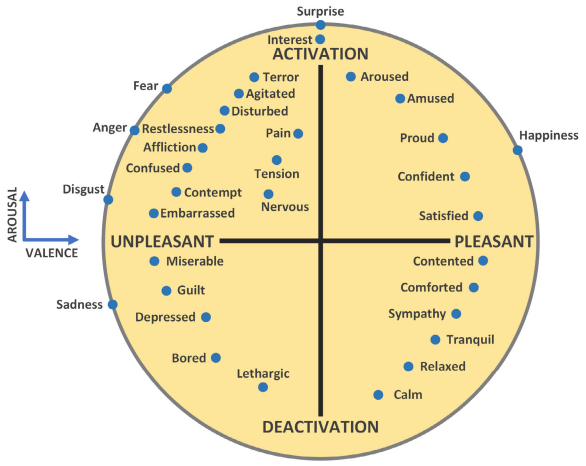

FIGURE 1. Circumplex of the emotions as a function of the Valence and Arousal.

40 one-minute length videos by using a 32-channel EEG cap sampling at 512 Hz. DREAMER provides the manual feedback and the EEG activity of 23 subjects rating 18 videos with variable length (65-393 s) and using the same 14-channel EEG cap from our study at a sampling rate of 128 Hz. Both repositories are used in this article along with our own dataset of labeled EEG waveforms for a set of 5 subjects.

The local EEG records have been collected using a low-cost off-the-shelf device, Emotiv Epoc Plus. The objective of the latter dataset is to increase the number of samples per subject. This is accomplished with a multitude of short experiences evoked using the International Affective Picture System (IAPS) stimulus. Fig. 2 displays one of the 180 subsets composing such experiments. The fully automated protocol starts with a RELAX picture lasting for 5 seconds. Next, it displays, the picture to be classified during 6 seconds. Last, it shows a rating picture called SAM (Self-Assessment Manikini) [18] that captures the valence and arousal experienced during the observation stage.

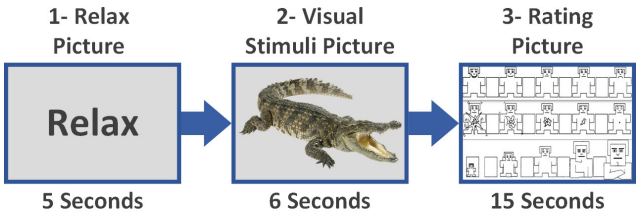

FIGURE 2. Sequence of the visual stimuli.

The three datasets are standardized and normalized into the single FEXD dataset, which is the largest EEG frequency data repository for emotion detection, containing 60 subjects and 2194 samples. DREAMER contributes 414 samples (23 subjects *sdrexx*); DEAP contributes 1280 samples (32 subjects *sdeaxx*), and the in-house collection supplies 500 samples (5 subjects *slocxx*). The three studies used the 10-20 system for the electrode placement, so reducing the density to

the EEG cap with the lowest number (14) of electrodes makes their features comparable. The voltage measurements on the scalp are a collection of multiple neurons firing at different regions, which makes the small mismatches in coordinates across EEG caps justifiable. All the three datasets are down-sampled to a common 128 Hz, the signals are bandpass-filtered to the frequency range 4-45Hz (Brainwaves θ, α, β, and γ), which is also the frequency range adopted by the DEAP dataset. Whereas the delta (δ) waves are somewhat correlated with the arousal and attention attributes [19], they are also highly susceptible to slow motion artifacts such as isolated muscular contractions, which is the main reason for their removal. The eye blinking artifacts are removed using a three-step process based on Independent Component Analysis (ICA). The components are pre-selected based on their scalp plot activation, their frequency spectrum is analyzed in search of a smooth decaying trend after an initial activation, and Event Related Potentials (ERP) images from the same emotion label are used to identify time-independent activation patterns.

B. PSD+Welch FEATURE EXTRACTION

FEXD stands for Frequency Emotion eXpression Dataset because the extracted EEG features are frequency-based. According to Davidson [20] there is strong correlation between the Power Spectral Density (PSD) of the EEG signals and the various EEG frequency bands during the emotion elicitation process. Further higher frequency bands contain more information about positive emotions than lower ones [21]. It has been pointed out that in the context of the DEAP dataset, negative correlation may exist between arousal and some of the EEG frequency bands (θ, α, and γ), [1]. As for valence, a close correlation exists between the PSD and all the EEG frequency bands.

The PSD, denoted $S_{xx}(f)$, describes how the power of the EEG time series is distributed over frequency. The approximation of the true PSD is achieved using

$$S_{xx}(f) = \frac{(\Delta t)^2}{T} \left| \sum_{n=1}^{N} x_n e^{-iwn} \right|^2 \tag{1}$$

which is based on the Discrete Fourier transform of the signal. A finite window $1 \leq n \leq N$ is considered with the signal sampled according to $x_n = x(n\Delta t)$ for a total time interval of $T = N\Delta t$.

Fig. 3 displays the effect of applying PSD to the 14 epochs received from the EEG cap channels. In this article, emphasis is placed on the PSD over handcrafted feature extraction techniques such as Higher Order Crossings (HOC) [9], Differential Entropy (DE) [8], or Asymmetrical Indices (AIS) [8]. This is because PSD facilitates the exploration of training alternatives across the three datasets with different epoch lengths. Non-handcrafted features are also discarded due to epoch length differences, which makes the raw data of the datasets incomparable. In order to estimate the true PSD, Welch's method is used. It is an averaging method

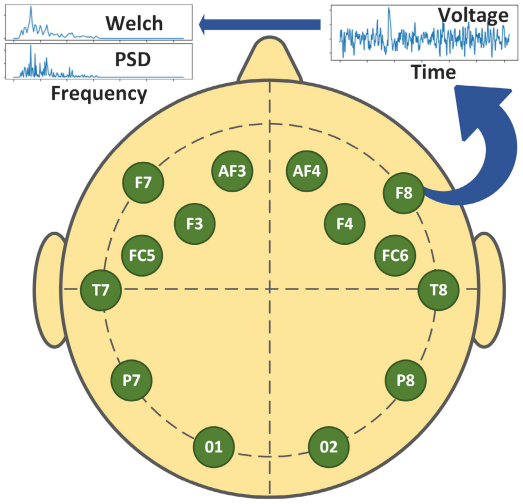

FIGURE 3. Frequency domain transformation of the epochs from each channel.

with overlapping bins that is typically used to reduce the PSD variance and increase the spatial correlation between consecutive bands on the frequency axis.

In order to involve all the relevant frequencies during training, the epoch length per channel is selected to match the duration in each experiment which ranged from 5sec to 393sec. The PSD of these multiple length episodes is down-sampled to a common count of 129 samples for all experiments without compromising their frequency content. As a result, each EEG input frame has a set of 14 frequency series coming from the 14 channels and arranged in a matrix form. The number of the frequency matrices is equal to the number of visual stimuli used in the experiments. The size of each matrix is 14×129, where 14 is the number of electrodes for the Emotiv EPOC+ cap, and 129 is the number of frequency-domain samples at the sampling rate of 128 Hz. Once the classifier is deployed, the input EEG frames are obtained by applying PSD+Welch to 14 rows of a testing epoch of 128 EEG samples, which are collected every second through the available channels.

C. AVERAGE PER FREQUENCY BAND

To reduce the dimensionality of the frequency-domain PSD+Welch features, an average per frequency band (APFB) is performed. The APFB-compressed data is then used for training the shallow models, which are often difficult to train using high-dimensional features. Even though flat averages discard spectral variations within the frequency bands, they still give information about the dominant EEG bands under a particular evoked experience as well as about any significant fluctuations within each band. The transformation process leads to the features expressed in Fig. 4. As can be seen, instead of having a matrix of size 14×129 for each visual stimulus, only a matrix of size 14×4 is used. On Fig. 4 the red, green, blue, and yellow colors refer to the band averages in θ, α, β, and γ bands, respectively.

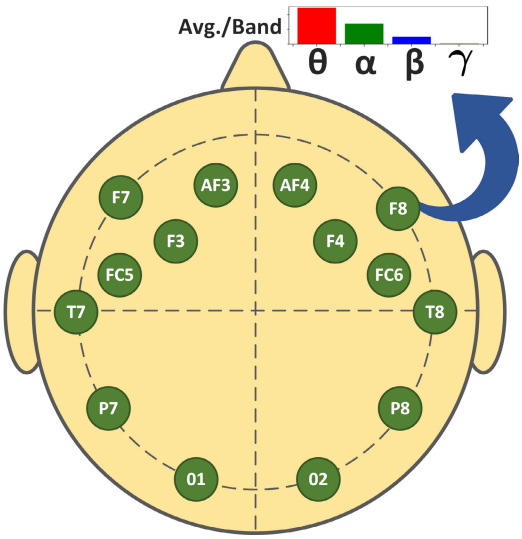

FIGURE 4. Features after the averaging process per frequency band for each of the channels.

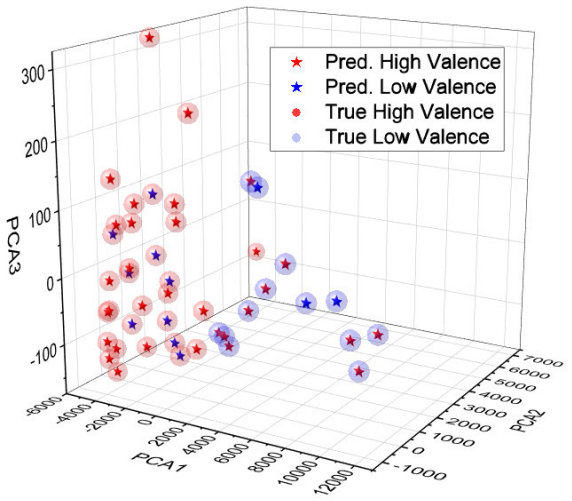

FIGURE 5. PCA of k-mean for two clusters (Left graph) and PCA data with the real valence tags (Right graph).

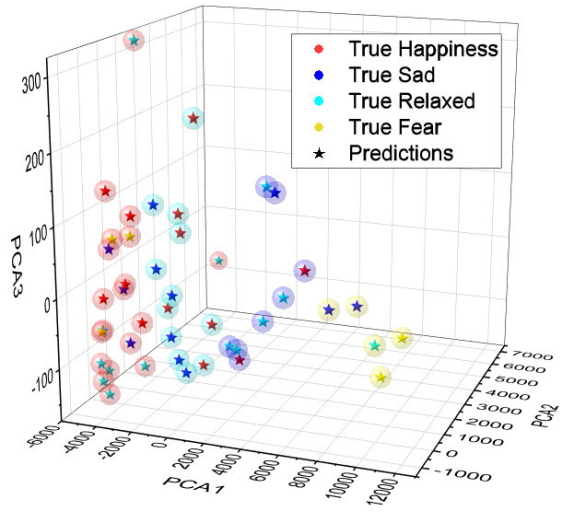

FIGURE 6. PCA of k-mean for four clusters (Left graph) and PCA data with the real emotion tags (Right graph).

III. ALGORITHM SELECTION

A. ALGORITHM DESIGN

Both unsupervised and supervised learning are explored to determine the best algorithm from a hardware implementation viewpoint for the emotion detection problem.

1) UNSUPERVISED LEARNING

Removing the labels from the dataset and exploring its content is a good approach to achieve familiarity with the data, and develop intuition of the emotion detection problem. The first algorithm explored is the k-mean clustering algorithm whose hardware implementation is simpler than other competing clustering algorithms such as Gaussian Mixture Models (GMM) or Density-Based Spatial Clustering of Applications with Noise (DBSCAN). The k-mean algorithm partitions the data into K clusters whose inter-point distances are less than the distances of any two pints belonging to two different clusters. Fig. 5 illustrates the principal component analysis (PCA) of the final clusters for valence along with the actual distribution of the valence classes. PCA is used to visualize the data by plotting the points with respect to the top three components. The k-mean algorithm describes the valence classes with an accuracy of 59%, which is illustrated graphically in Fig. 5 where the predictions are marked as stars, and the actual values as circles whose colors indicate the valence value (low of high). The prediction accuracy of Fig. 5 is not the best that can be achieved but is comparable to several prior studies such as [1], [11], [22].

When the clustering includes both valence and arousal, the k-mean prediction accuracy is significantly degraded as it is clear in Fig. 6, where predictions (Stars) and actual values (Circles) fail to align across the graph.

2) SUPERVISED LEARNING

The shallow models used for the supervised learning approach are Decision Tress, Random Forest, and Support Vector Machines with Radial Basis Function (SVM-RBF) as kernel. A decision tree is a hierarchical graph model whose branches are generated according to a decision parameter. A Random Forest is an ensemble of decision trees, constructed during training phase. The inference phase is based on the majority voting of the various decision tress.

SVM on the other hand is a large margin classifier, which uses kernels to construct a separation hyper-plane in a high-dimensional space that splits the two classes. The kernel used in this work is the Radial Basis Function (RBF), which allows more irregular boundaries in the original feature space. An RBF is given by

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \tag{2}$$

where x and x' are two points in the feature space, and σ is the radius of proximity beyond which the kernel value becomes small. Such RBF kernel is considered a similarity

TABLE 1. Accuracy comparison of the shallow models using single trial of 20% out, and 10-fold CV.

Model Used	Validation Sub[a] (%)		10-fold CV (%)		Dataset (Sub[a])
	Valence	Arousal	Valence	Arousal	
k-mean	56.82	50	N/A	N/A	DEAP[(32)]
Random Forest	68.33	68.33	49.25	49.06	DEAP[(32)]
SVM (RBF)	73.54	68.33	51.62	51.61	DEAP[(32)]
Decision Trees	63.54	53.75	34.65	34.63	DEAP[(32)]

Sub[a]: Subjects. DEAP[(32)]: Including 32 subjects.

measurement as it has a value close to 1 whenever x and x' are close to each other.

Table 1 shows the classification accuracy using the APFB features for a single trial test and a 10-fold cross validation (CV). Among the group of shallow supervised learning, SVM-RBF and Random Forest are the best models for the single trial test. The 10-fold CV is more informative than the single trial test as it takes into account multiple scenarios and gives a more realistic evaluation. Unfortunately, all the shallow algorithms perform poorly in the 10-fold CV, and only SVM slightly outperforms the Random Forest classifier while keeping a more consistent behavior for both validation scenarios. In general, none of the shallow models performed well. One explanation for this may be that the APFB features do not capture important variations within each EEG frequency band that could help in achieving better classification.

B. CNN ALGORITHM

Convolutional Neural networks (CNNs) are inspired by the mammal visual cortex and were originally used for image recognition. In the context of emotion classification, the EEG frequency series are stacked as 2D images so that a CNN can be used for EEG feature recognition. CNNs are particularly suitable for extracting complex features from 2D data, even in the presence of variations.

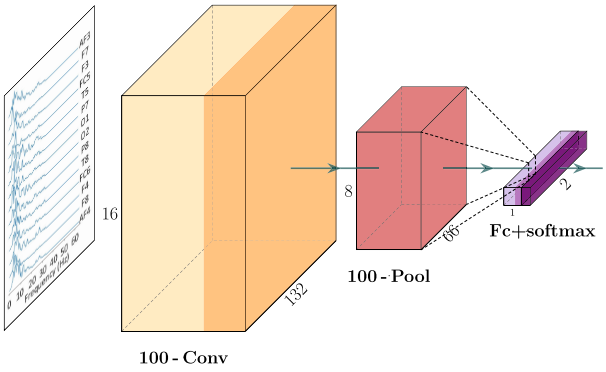

FIGURE 7. Architecture of the Valence classifier implemented in Tensorflow.

Fig. 7 illustrates the architecture of the CNN valence classifier prototyped in TensorFlow [23]. The frequency frame is passed to a convolutional layer containing 100 different kernels, whose convolution results are passed through a rectifier linear unit (ReLU) function, and down-sampled using Maxpooling with a 2×2 pool window. A dropout of 20% is applied to the intermediate activations, which are flattened onto a fully connected network with a single layer to make it hardware friendly. The dense layer has only two neurons representing the HIGH and LOW values of the valence scale. Another CNN architecture similar to Fig. 7 is used for the arousal scale.

TABLE 2. Accuracy comparison of the CNN model using 10-fold cross validation.

Model Used	10-fold CV (%)		Dataset (Subj)
	Valence	Arousal	
CNN	83.12	76.78	DEAP (32)
CNN	55.94	81.41	DREAMER (23)
CNN	66.96	73.34	FEXD (60)

Using 10-fold cross validation, Table 2 shows that CNN achieves a significant improvement in classification accuracy with respect to the shallow models. The model learns a more accurate representation for the valence when trained with the DEAP dataset. DEAP and DREAMER work with different EEG caps and under different experimental setups, but the model and the standardized frequency features display an acceptable performance in FEXD, close to the state of the art. Deeper convolutional neural networks might reach higher accuracy, but given our hardware objective of achieving a resource-constrained implementation, the model in Fig. 7 represents a good trade-off between complexity and performance.

IV. BioCNN HARDWARE IMPLEMENTATION

The hardware implementation of CNN inference engines is highly application-dependent and under severe resource constraints, it is very challenging. Due to the remarkable success that CNN has achieved in solving the object recognition problem, there has been an explosion in academic and industry R & D for optimizing the CNN hardware architecture to achieve stringent specifications on power, performance, bandwidth, area, and inference accuracy. One promising trend to accelerate CNN hardware is to take advantage of the sparsity generated by weight pruning, and rectified linear unit (ReLU) functions. Another approach is to use data compression at the cost of including encoding and decoding stages before and after inference. Further performance gain can be obtained from using special processing units that operate only non-zero values [24]. The use of fast arithmetic algorithms such as the Winograd multiplier [25] has also been considered although it results in a more complex activation stage [26]. A thorough tutorial survey on CNN hardware acceleration with focus on the convolutional layers is given in [27].

In this section, we report on the architecture, logic design, and hardware implementation of a specialized Convolutional Neural Network, called BioCNN, that is optimized for

biomedical applications such as EEG-based seizure detection or EEG-based emotion classification. Such applications have lower bandwidth requirements than for example, computer vision tasks. Although high-level synthesis tools such as Mentor's Catapult can be used for architectural exploration, BioCNN has relied on a custom design with the objective of achieving tight integration between the various CNN stages without sacrificing the flexibility needed for resource re-use. Fig. 8 illustrates the flowchart of the proposed BioCNN, highlighting the re-usability feature of the design as illustrated by the various return paths. In particular, we point out the usage of data buffer queuing, CNN buffer queuing, and pipelined dot products with the goal of maximizing resource usage while increasing throughput. Other usage of pipelining, especially in terms of concurrently running the CNN Phase and the maxpooling phase will be described later in this section.

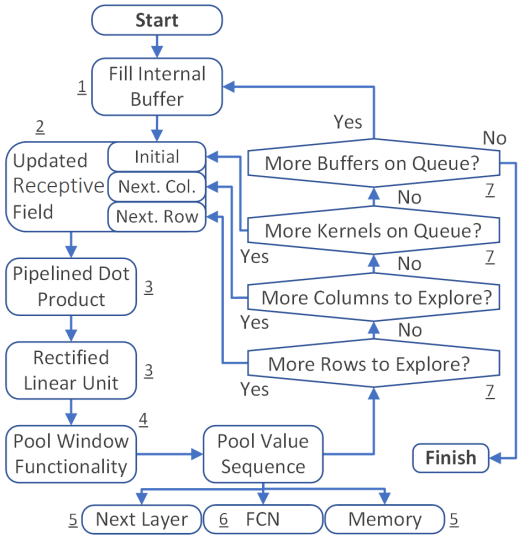

FIGURE 8. Data flowchart of the proposed BioCNN model ([15], Fig. 1(a)).

The overall block diagram of the full system, including training, is presented in Fig. 9. Components such as the signal pre-processing module, the feature extraction unit, and the model training, are implemented offline and off-chip. Model training is based on the subject-dependent paradigm, where the data used for training is collected and tested for only one subject [1], [2].

A. BUFFER FILLING

The pseudo-code describing the filling process is presented in Algorithm 1. The Buffer Filling module in Fig. 8 receives the individual activation value (*AuxReg*) from an internal memory (*Stream*) or from the pre-processed signals of the sensor itself. The first condition (Line 3) is in charge of moving the input values into an internal array ($Struct[0 : M-1]$) of M elements. Once the array is full (Position $x = M$), *Struct* is displaced one level down inside the internal buffer ($IntBuff[0 : N-1][0 : M-1]$) to free resources

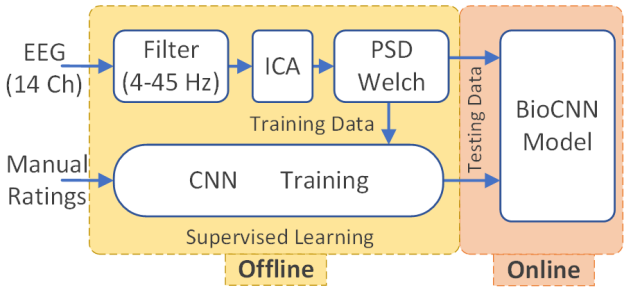

FIGURE 9. High-level block diagram of the emotion classifier ([15], Fig. 1(b)).

Algorithm 1 Pseudo-Code for the Internal Buffer Filling

```
1:  for b=0 to B-1 do                                   ▷ All Buffers
2:      AuxReg=Stream
3:      if x< M then                                    ▷ Shift Right
4:          Struct[0]<=AuxReg
5:          Struct[1]<=Struct[0]
6:          Struct[2]<=Struct[1]
7:          ...
8:          Struct[M-1]<=Struct[M-2]
9:      end if
10:     if y < N and x = M-1 then                       ▷ Shift Down
11:         IntBuff[0][0:M-1]<=Struct
12:         IntBuff[1][0:M-1]<=IntBuff[0][0:M-1]
13:         IntBuff[2][0:M-1]<=IntBuff[1][0:M-1]
14:         ...
15:         IntBuff[N-1][0:M-1]<=IntBuff[N-2][0:M-1]
16:     end if
17: end for
18: // Start Buffer Processing
19: // Conv
20: // ReLU
21: // MaxPool
```

for receiving the next row. The process is repeated until the internal buffer is filled (Position $y = N$), which triggers the next stages marked with green color at the end of the pseudo-code.

The hardware architecture of this block is illustrated in Fig. 10. The horizontal and vertical displacements are triggered by the "Shift Right" (1.2) and "Shift Down" (1.3) signals, respectively. These signals are controlled using the input value coordinates (x, y) so as to ensure proper filling without running into overflow exceptions.

The 128 Hz sampling frequency of the EEG cap is considerably lower than the 100MHz clock used to operate the classifier. The internal buffer module plays a crucial role in matching the source data rate with the CNN computational needs and in achieving the goal of real-time emotion detection.

B. RECEPTIVE FIELD UPDATES

The receptive field in Fig. 11 corresponds to a dynamical structure (2.5) slided and filled up with the individual activations, at every valid position (2.1) of the input frame during the convolutions. The valid positions are represented by the black arrows (2.1) and are determined by the stride of the convolution. The stride movements are selected to be

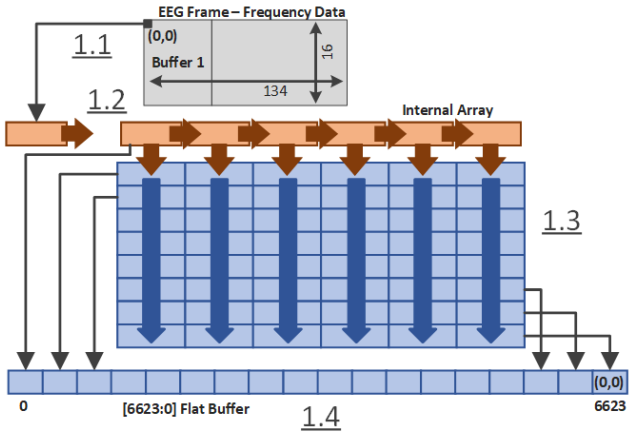

FIGURE 10. Hardware diagram of the internal filling module receiving an input EEG frame.

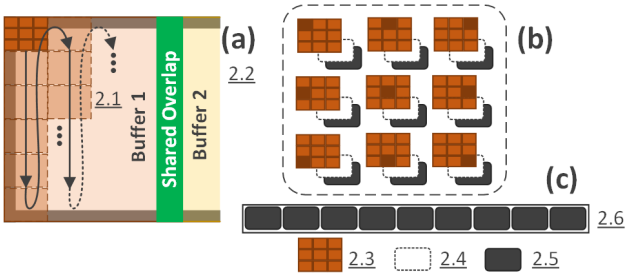

FIGURE 11. Hardware structure of the receptive field (a) Individual activations at valid positions (b) Receptive field updates (c) Flattened window activations.

vertical (2.1) so as to complete all the possible calculations before moving to the next buffer. The calculations at the right edge are executed by extending the buffer to include the shared overlap (2.2) between two consecutive buffers. The overlap required for a 3×3 kernel is one column that is obtained from the adjacent buffer.

The individual selectors (2.4) are in charge of extracting an individual value to fill up the receptive field (2.5). There are as many individual selectors as elements in the convolution kernel. Each selector is responsible for one particular kernel element (2.3). The receptive field values are flattened (2.6) and transferred for further calculation with the kernel values at every stride movement.

C. RECTIFIED CONVOLUTIONAL LAYER

The flattened versions of the kernel and activation window are received by the Pipelined Dot Product and Rectified Linear Unit which constitute the Rectified Convolutional Layer. This is shown in Fig. 8. The pseudo-code in Algorithm 2 illustrates the high degree of resource re-utilization during the convolution with the input frame. Kernel (*k for loop*) and buffer (*b for loop*) queuing over a small number of hardware branches enables a significant reduction in logic resources without impacting data consistency. Each buffer contains one EEG data partition with overlap between

Algorithm 2 Pseudo-Code for the Convolutional Layer

```
for b=0 to B-1 do                                   ▷ All Buffers
    // Buffer Filling
    for k=0 to KS-1 do                              ▷ All Kernels
        for hs=0 to HS-1 do                         ▷ All Horizontal Strides
            for vs=0 to VS-1 do                     ▷ All Vertical Strides
                Conv[vs][hs]<=K^T X_{hs,vs}         ▷ Dot Product
            end for
        end for
    end for
end for
```

adjacent buffers for convolution consistency. As described in Algorithm 2, the system selects one partition ($b = 0$) at a time, and applies all kernels ($[0 : KS - 1]$) over the selected partition. The receptive field described in the previous paragraph is moved across all valid positions (vs, hs), and the corresponding dot product between the kernel and the activation window ($K \cdot X_{vs,hs}$) is computed for each position.

Hardware branches to execute convolutions in parallel can be added to accelerate this step. Lower latencies are achieved by concurrently taking care of pending kernels in the queue. Once the results for all kernels are obtained at every valid position (vs, hs), the next buffer ($b = 1$) is placed in the buffer queue, and the process is repeated until the input frame is fully covered ($b = B - 1$). The return paths in Fig. 8 illustrate the use of the same logic resources to execute convolutions on multiple kernels and multiple buffers.

The hardware architecture of the Convolutional Layer for a 3×3 kernel is illustrated in Fig. 12. The first stage of the pipeline is composed of a stack of parallel multipliers (3.2) that generate the individual multiplications of the dot product operation. The subsequent pipeline stages are the 4 levels of the hierarchical tree adder (3.2) that is used to complete the dot product operation. The actual implementation of the arithmetic operations is sparsity-aware and therefore avoids unnecessary zero-operand cases. The last stage of the pipeline is the rectifier linear unit (ReLU) function (3.3) that replaces negative dot products with zero.

Fig. 13 presents the execution of the six pipeline stages for the first twelve vertical movements of a 3×3 kernel. Fig. 13 further provides a basis for quantifying the throughput of the rectified convolutional layer, which is one rectified dot product per clock cycle. A validity check is made and non-valid results are flagged using the *Conv_Rdy* signal. The number of pipeline stages in the Convolutional Layer is directly related to the number of elements in the kernel, which in turn determines the number of levels of the hierarchical tree adder. Table 3 shows three different scenarios for common kernel sizes, where M is the number of multipliers, A the number of adders at each stage, and R the ReLU stage. Except for the ReLU stage, the numbers on the second row of Table 3 represent the maximum number of additions or multiplications at the various pipeline stages. In some cases, this maximum number is not necessarily required, which translates in further saving of resources. A case in point is the

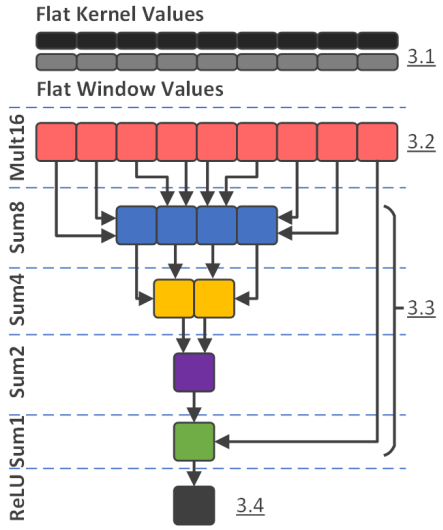

FIGURE 12. Hardware implementation of the rectified convolutional layer ([15], Fig. 1(d)).

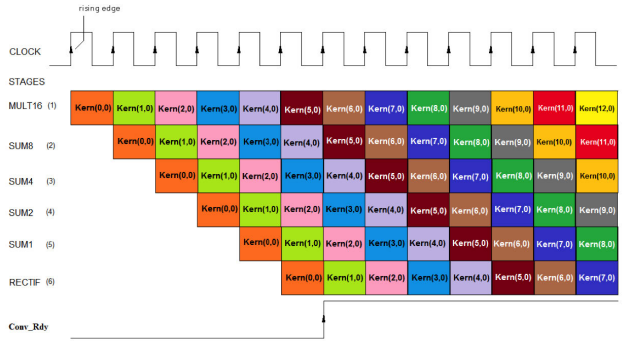

FIGURE 13. Pipeline execution for a 3 × 3 kernel ([15], Fig. 2).

TABLE 3. Convolutional layer dependence on the kernel size.

Kerne Size	Pipeline Stages								
	128	64	32	16	8	4	2	1	ReLU
3x3				M	A	A	A	A	R
4x4				M	A	A	A	A	R
7x7		M	A	A	A	A	A	A	R

3 × 3 kernel where only 9 multiplications are needed instead of the 16 required by the design hierarchy.

D. MAXPOOLING

The results produced in the convolutional layer are sequentially received by the Pool window module in Fig. 8. The pooling process can be initiated even before the first single kernel convolutions are completed. This pipelining of maxpooling with convolution helps improve the throughput of BioCNN while reducing the memory footprint of the intermediate values. Hardware pooling further eliminates the need of an additional CPU core to execute software pooling. The pseudo-code of the hardware pooling algorithm using a 2 × 2 pool size is given in Algorithm 3. The pseudo-code has a hypothetical input of an $N \times M$ matrix composed of the convolution dot products ($Conv[i][j]$). This input is only for explanatory purposes since as is clear from the previous subsection, the convolutional layer generates its results one column at a time.

Algorithm 3 Pseudo-Code for Hardware Maxpooling

```
1:  for i=0 to N-1 do                                   ▷ Conv Columns
2:      a<=0
3:      for j=0 to M-1 do                               ▷ Conv Rows
4:          if j%2==0 then                              ▷ Even Row Number
5:              buffer<=Conv[i][j]
6:          else
7:              selector<=(Conv[i][j]<buffer)? 1: 0
8:              if selector==1 then MaxVertical<=buffer
9:              else MaxVertical<=Conv[i][j]
10:             end if
11:             if i%2==0 then                          ▷ Even Column Number
12:                 FIFO[a]<=MaxVertical
13:                 a<=a+1
14:             else
15:                 selector2<=(FIFO[a]<MaxVertical)? 1: 0
16:                 if selector2==1 then
17:                     MaxPool<=MaxVertical
18:                 else MaxPool<=FIFO[a]
19:                 end if
20:                 a <= a+1
21:             end if
22:         end if
23:     end for
24: end for
```

The pooling module with a 2 × 2 pool size is illustrated in Fig. 14. Consistent with the convolutional layer pipeline, the pooling module stores the first convolution value (4.0) into a buffer (4.2) during the first clock cycle. In the next clock cycle, the next convolution value (4.1) is compared with the buffer (4.2) to produce the selector signal (4.3) used to pass the *MaxVertical* value through the multiplexer (4.4) and place it in the FIFO buffer (4.5a). This pairwise vertical comparison is repeated each vertical stride. All the *MaxVertical* values from the even-numbered columns are stored. Parallel execution of multiple vertical poolings can accelerate the generation of *MaxVertical* values with the FIFO buffer split according to the number of parallel branches. Similarly of the *MaxVertical* values of the odd-numbered columns are generated according to the (4.0)(4.2)(4.1)(4.3)(4.4) sequence of operations. However, the new *MaxVertical* (4.4) is now compared with the old *MaxVertical* (4.5b) stored in the FIFO buffer. This comparison produces the selector (4.6) of the final *MaxPool* value (4.7) for the current Maxpooling window. This even-odd column procedure is repeated until all the pool values from the current activations are generated. The throughput of BioCNN with a single hardware instance is 1 *MaxPool* every 4 clock cycles, which is not as high as the throughput of the convolutional layer, but it is vastly better than waiting for all the convolutions to complete before starting the maxpooling operation. Two concurrent digital counters are used to track of the *MaxPool* window coordinates.

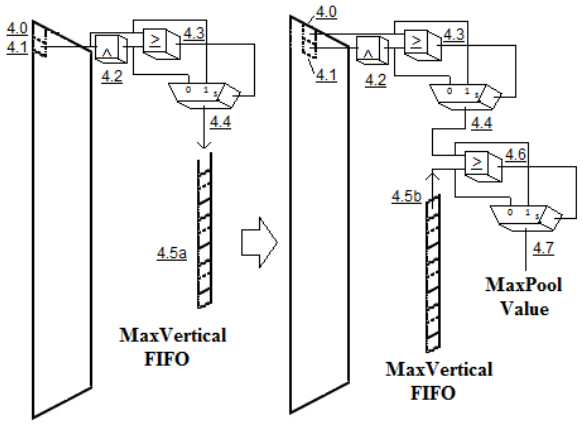

FIGURE 14. Generation of pooled values using a 2 × 2 window ([15], Fig. 3).

E. FULLY CONNECTED OUTPUT NETWORK

The sequential generation of *Maxpool* values requires a special topology for the Dense Network in Fig. 8. The pseudo-code of the proposed dense network is given in Algorithm 4. For the first *Maxpool* value *MaxP*[0][0], only the weights associated with its neuron connections, $W\_L_0\ N_{0\{0,0\}}, \ldots W\_L_0\ N_{Z\{0,0\}}$, where L_0 is the 0-th layer and $N_k, 0 \leq k \leq Z$ are its neurons, are fetched and deployed to the parallel multipliers. The multiplication results are then accumulated in the accumulators $Ac\_L_0\ N_k, 0 \leq k \leq Z$. When the second *Maxpool* value *MaxP*[1][0] is computed, the new set of weights, $W\_L_0\ N_{0\{1,0\}}, \ldots W\_L_0\ N_{Z\{1,0\}}$, for that pooling position is fetched, and the neuron accumulators are updated with the new products. The sequential process is pipelined with the rectified convolutional layer and maxpooling module to form a single coherent data path from the input buffer to the output classification. Once all the *Maxpool* values are processed, the final accumulators are rectified, $ReLU(Ac\_L_0\ N_k)$ with $0 \leq k \leq Z$, and the output layer is activated. The output layer has all its input activations available, which is the typical feed-forward propagation as in Fig. 15(a). The output layer of the dense network is considered in Algorithm 4 under "Dense Layer 1" where the Multiply-Accumulate (MAC) values are obtained directly for the two output neurons. The very final stage is a comparison between the two accumulators to determine the classification label (*ClassTag*). Note that Algorithm 4 presents a more generic case for the dense network where multiple layers are used. The actual hardware implementation has a single-layer dense network with the final classification accumulators activated immediately after "Dense Layer 0" is completed.

As illustrated in Fig. 15(a), if a traditional dense network (6.1) is used, all the *Maxpool* values (6.11), (6.12), (6.13), (6.14), and (6.15) must be processed at the same time, which forces the network to have a high number of weights (6.16) buffered within the FPGA. The intermediate neural layers of the traditional (6.17)(6.18) and the proposed (6.24)(6.25)

Algorithm 4 Pseudo-Code for the Dense Output Network

```
for b=0 to B-1 do                                          ▷ All Buffers
    // Buffer Filling
    for k=0 to KS-1 do                                     ▷ All Kernels
        // All Horizontal Pool Strides
        for ph=0 to (HS/2)-1 do
            // All Vertical Pool Strides
            for pv=0 to (VS/2)-1 do
                // Convolutional Layer
                // ReLU
                // MaxPool
                // Dense Layer 0
                Ac_L0N0 <=MaxP[ph][pv]×W_L0N0{ph,pv}
                                          +Ac_L0N0
                Ac_L0N1 <=MaxP[ph][pv]×W_L0N1{ph,pv}
                                          +Ac_L0N1
                .........
                Ac_L0NZ <=MaxP[ph][pv]×W_L0NZ{ph,pv}
                                          +Ac_L0NZ
                // ReLU
                Ac_L0N0 <=ReLU{Ac_L0N0}
                Ac_L0N1 <=ReLU{Ac_L0N1}
                .........
                Ac_L0NZ <=ReLU{Ac_L0NZ}
                // Dense Layer 1
                Ac_L1N0 <=Ac_L0N0×W_L0N0_L1N0
                          +Ac_L0N1×W_L0N1_L1N0
                          .........
                          +Ac_L0NZ×W_L0NZ_L1NZ
                Ac_L1N1 <=Ac_L0N0×W_L0N0_L1N1
                          +Ac_L0N1×W_L0N1_L1N1
                          .........
                          +Ac_L0NZ×W_L0NZ_L1N1
                // Softmax or Comparison
                ClassTag<=(Ac_L1N0< Ac_L1N1)? 1: 0
            end for
        end for
    end for
end for
```

dense networks look the same to the pooled values of the first layer. The hardware implementation of the proposed dense network of Algorithm 4 is presented in Fig. 15(b). The network accepts a serial stream of pooled values (6.21)(6.22) and processes them one at a time. This single stream requires only one weight connection per each intermediate neuron to be available at the input layer (6.23), which results in a significant reduction in the required FPGA buffer resources. In turn, this serialization requires a specific read sequence to fetch the appropriate weights such as the one described in Algorithm 4.

A full hardware implementation including all the stages is illustrated in Fig. 16. The hardware features two parallel branches to execute convolutions and a dense output network feeding into two neurons for classification. Aside from the hardware parallelism, massive pipelining amongst the individual modules is used to guarantee a coherent, high-throughput data path. Another level of hardware parallelism is further used as the architecture of Fig. 16 is duplicated to classify the valence and arousal concurrently.

V. PROTOTYPE EVALUATION

To the best of our knowledge, BioCNN is the most compact hardware classifier of emotions that has been reported in the

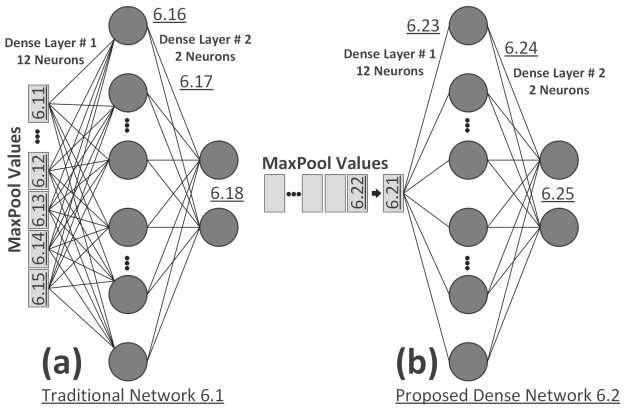

FIGURE 15. (a) Traditional approach to execute dense layers (b) Proposed approach to integrate sequential maxpool values.

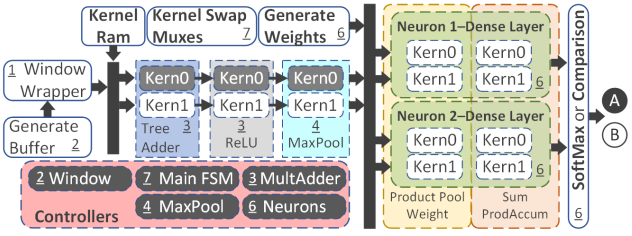

FIGURE 16. Hardware architecture of BioCNN. This architecture is duplicated twice for valence and arousal classification.

literature. It features a highly optimized architecture with maximum module reuse and minimum resource utilization. The optimization is such that the whole BioCNN design, including both valence and arousal classification, fits within the low-end Atlys Spartan-6 FPGA Board. The design has been conceptualized and optimized without using high-level synthesis tools. In this section, the BioCNN hardware implementation is evaluated from the viewpoints of functionality, accuracy, latency, and throughput. The important features of resource re-utilization and data quantization are addressed in the next section.

A. FUNCTIONALITY

The hardware functionality of each architectural module is evaluated on its own first using simulations and then using the experimental setup. The modules being assessed are the Rectified Convolutional Layer, Maxpooling, and the Dense Output Layer.

1) RECTIFIED CONVOLUTIONAL LAYER

As shown in Fig. 13, the six-stage pipeline takes six clock cycles to produce the first result. The simulation of this behavior is illustrated in Fig. 17. Once the internal buffer is filled up, a ready signal (3.1) is broadcast to start the convolutions. The first stage comprises the parallel multipliers between the kernel weights and the activations within the sliding window. It is executed in the first clock

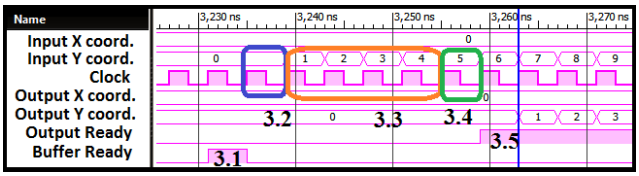

FIGURE 17. Logic simulation of the rectified convolutional layer.

cycle and is identified with a blue square (3.2). The next four clock cycles correspond to the tree adder execution (3.2), whereas the last clock cycle, identified with a green square (3.3), corresponds to the ReLU stage. After the ReLU function is applied, the Output Ready signal (3.5) is asserted to indicate a valid result at the output rectified convolutional layer. The Output Ready signal remains asserted as long as there is a valid output result at the end of the six-stage pipeline.

2) MAXPOOLING

The Maxpooling hardware is based on the pseudo-code presented in Algorithm 3. The logic simulation in Fig. 18 shows how a sequence of rectified values captured in the *rectified_value* trace is processed in this module. In the testing scenario, the rectified signals are intentionally adjusted to have the same values of 160 and 156, for the even- and odd-numbered columns, respectively in order to visualize column changes and illustrate the hardware functionality.

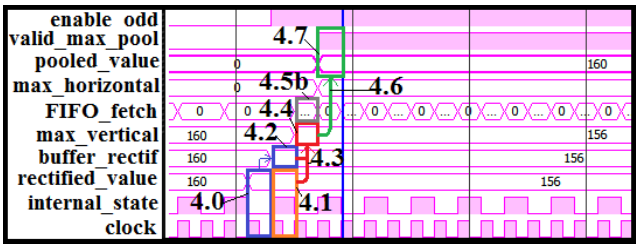

FIGURE 18. Logic simulation of the Maxpooling layer.

The first value (4.0) of a particular column is buffered (4.2) and held for one clock cycle. The following rectified value (4.1) is compared against the previously buffered one (4.2), and a selector signal is generated (4.3). The selector broadcasts the *MaxVertical* value (4.4), which is the greater of the two rectified values (4.0) and (4.1). As stated in Algorithm 3, if the originating column is even, *MaxVertical* is buffered in a FIFO, otherwise it is held for one more clock cycle and subsequently compared against its corresponding pair inside the FIFO. The odd-or-even action is controlled with *enable_odd* signal, which is asserted high for an odd column.

Fig. 18 shows the operations for an odd-numbered column. The *MaxVertical* (4.4) is compared against the value (4.5b) of the FIFO, to generate *MaxHorizontal* (4.7), which is the pooled result. A *valid_max_pool* signal is asserted to indicate the availability of *MaxPool* (4.7). When only one Maxpooling

instance is used, it may become a throughput bottleneck. However, even a single Maxpooling instance can be pipelined with the Rectified Convolutional Layer to form a coherent datapath with the pipeline stages being essentially stall-free.

3) DENSE OUTPUT LAYER

The Dense Output Layer (DOL) is based on the dot product between the weights and the inputs for each connection. The inputs are the serialized *Maxpool* values (6.21) and (6.22) in Fig. 15. The dense multipliers in Fig. 19 are triggered by *MaxPool* (4.7) as previously described, and the product (6.23) is added to the general accumulator (6.235) in the next cycle. In contrast to the dot product implementation of the Rectified Convolutional Layer, the DOL multipliers receive sequences of *MaxPool* values that are staggered in time. The DOL dot product algorithm, shown in Algorithm 3, maintains accumulators to store the final results passing consistent values to the output Softmax stage for classification.

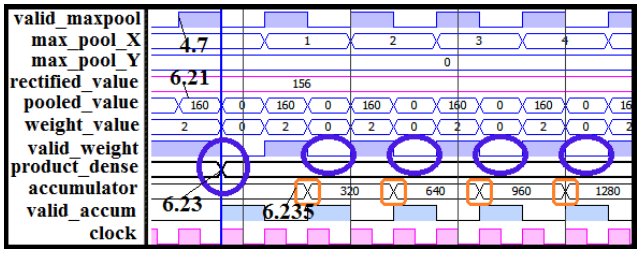

FIGURE 19. Logic simulation of the dense output layer.

The DOL multipliers are edge-triggered and the blue circles (6.23) in Fig. 19 highlight the rising edges that enable them, whereas the *valid_accum* signal indicates when a product value is added to an accumulator. Other accumulator control signals are shown in Fig. 20, including *accumulate* and *StoreRestore*. The final accumulator values (6.25) for each neuron, [180320, 293972], are available only after the last product (6.235) is generated using the last *MaxPool* value (4.7). The final classification result is obtained by comparing the accumulators of the two output neurons and generating the *classif_TAG* signal (6.255) to encode the two classes.

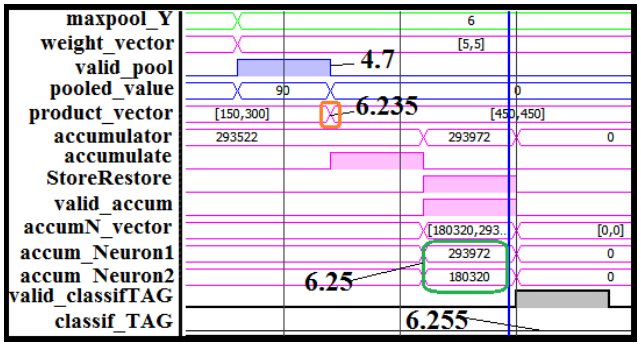

FIGURE 20. Logic simulation of the final accumulators for a given EEG frame.

In addition to the TensorFlow prototype, an emulation program has been written in Python to debug the hardware implementation at every node. Such debug information is given in Fig. 21 for the accumulators of the two output nodes.

```
Partial value of Total_accumulator_per_neuron FROM NEURON 0:  71470
Partial value of Total_accumulator_per_neuron FROM NEURON 1:  46522

 BUFFER 3
Accumulator total for Neuron 0:  71470
Accumulator total for Neuron 1:  46522

TOTAL RESULTS

Total Accumulators:
The total accumulator for Neuron 0 including all three buffers is:  293972
The total accumulator for Neuron 1 including all three buffers is:  180320

Classification TAG:  0

In [4]:
```

FIGURE 21. Final accumulator state for a given EEG frame using the Python reference emulator.

4) EXPERIMENTAL TESTBED

The experimental setup used to test the full system and evaluate hardware accuracy is shown in Fig. 22. The starting point is the collection, labeling, and saving of EEG recordings. Two Python programs are used for this stage, and the data is saved in a CSV file. Subsequently, the signals are filtered, artifacts removed using Independent Component Analysis, and the PSD+Welch features extracted. The 14-channel, frequency-domain EEG frame is dispatched to the FPGA through a UART interface using Matlab. The data inside the FPGA is transformed according to the hardware architecture of Fig. 16.

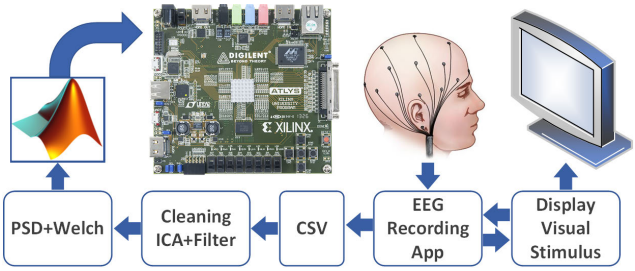

FIGURE 22. BioCNN experimental testbed.

B. ACCURACY

The hardware CNN classifier of this article is tested using the FEXD dataset which comprises 60 subjects. The individual 10-fold cross-validation results are presented in Fig. 23 for all subjects. However, for an apple-to-apple comparison with state-of-the-art, the most popular dataset (DEAP) is used for the figures of merit. All the accuracies of the state-of-the-art classifiers for EEG emotion detection are summarized in Table 4 where the SD acronym refers to the Subject-Dependent approach.

C. LATENCY AND THROUGHPUT

The input frame size without partitions is 14×129, where 14 represents the number of channels and the 129 corresponds to

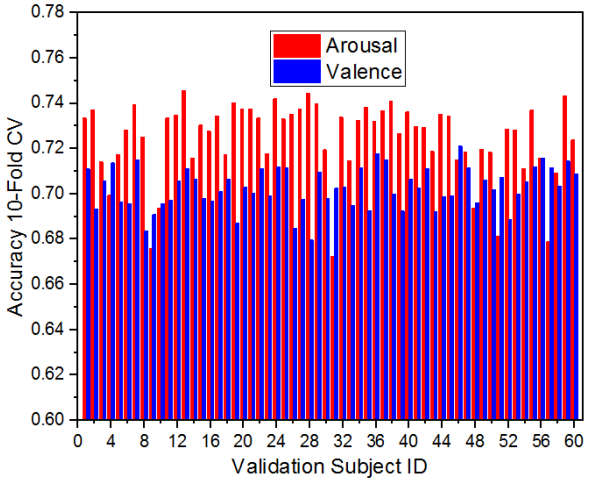

FIGURE 23. Valence and Arousal accuracy for all FEXD subjects using 10-fold cross validation.

TABLE 4. Accuracy comparison with the state of the art using the DEAP database.

Model Used	Valence	Arousal	Sub[a]	Year	Validation
Fusion EEG [1]	57.6	62	32	2012	L1.OUT SD
RAW CNN [6]	81.40	73.36	32	2017	CV N/A SD
Hybrid CNN RNN [3]	90.8	91.03	32	2018	10-fold CV SD
Multiphase CNN SW [14]	78.23 Quaternary		32	2019	CV N/A SD
Multiphase CNN SW [14]	85.06	-	32	2019	CV N/A SD
Multicolumn CNN SW [4]	90.01	90.65	32	2019	CV N/A SD
SNN [5]	78	74	32	2020	CV N/A SD
Our CNN on SW	**83.12**	**76.78**	**32**	**2020**	**10-fold CV SD**

Sub[a]: Subjects

the number of frequency bins. Assume the proposed BioCNN prototype has K kernels, B buffer partitions, and N parallel branches, then its latency L_t in number of clock cycles is given by

$$L_t = \frac{BK}{N}[14 \times 44 + C + M + S + D + (N-1) + R] \quad (3)$$

where the product 14×44 is the number of clock cycles taken by the kernel to move across all valid positions within one buffer, B is the number of partitions on the buffer, C the number of stages in the convolutional layer within each buffer, M the Maxpooling overhead, S the kernel swapping overhead, D the Dense Output Layer overhead, $(N-1)$ the number of clock cycles needed to combine the accumulators of the N parallel branches, and R the overhead of the final classification. In the proposed architecture, $M = S = D = 2$ clock cycles, $C = 6$, $B = 3$ clock cycles, and $R = 1$ clock cycle.

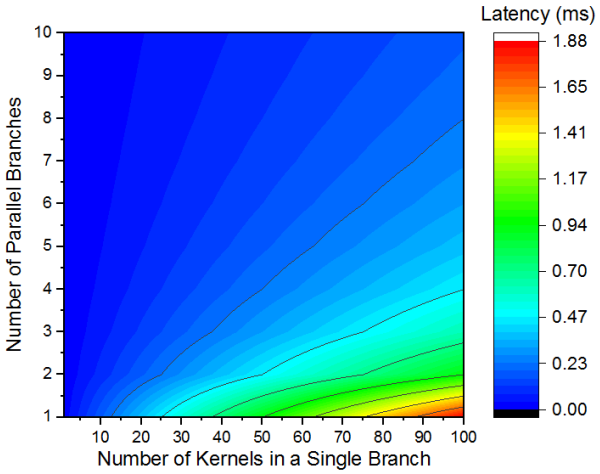

FIGURE 24. Latency contour plots of as a function of parallel branches and convolution kernels.

The latency is plotted in Fig. 24 to show the dependency on the numbers of kernels and parallel branches. As expected, the addition of parallel branches reduces latency, with the worst-case latency obtained for a single branch, and even in this case, the latency is about two orders of magnitude less than the fastest human interactive time, which is $150ms$ seconds for a touch stimulus. For the BioCNN prototype with two parallel branches and 100 kernels, the latency is $0.93ms$.

An estimation of the BioCNN throughput T_h is given by

$$T_h = N\frac{22 \times 7A_{ccum} + 5A_{ccumPrevK}}{14 \times 44 + C + M + S + D} \quad (4)$$

where A_{ccum} represents a single accumulation result and is scaled with 22×7, which is how many of those are generated since BioCNN starts to process one particular kernel through a single branch. $A_{ccumPrevK}$ is an accumulator from the previous kernel and is scaled with 5, which is the number of hierarchical pipeline stages in the convolutional layer in Fig. 12. N corresponds to the number of parallel kernel branches, whose value is 1 for the single-branch estimation. The numerator in (4) expresses the total number of accumulators produced in a single kernel branch. The denominator counts the number of clock cycles it takes to produce those accumulators, which is the same as the time needed for scanning valid buffer locations and executing the pipelined convolutions along with the overhead due to Maxpooling, Kernel Swapping, and Dense Output Layer. Note that only the pipelined stages are considered for throughput calculation because the last stage accounts for only a comparison of the accumulators.

The throughput presented in (4) only accounts for the number of outputs or accumulators produced. However, in order to compare the BioCNN performance with the CNN state of the art, all operations involved need to be reported. Additionally, the clock cycles need to be replaced by time units (seconds). The fixed-point operations involved in the Convolutional and Dense Layers are summarized in Table 5,

TABLE 5. Operations performed in the rectified convolutional and dense output layers.

Layer	Stage	Operations per clk_c
Convolution	Tree Adder	$6 \times N$
	ReLU	$1 \times N$
	MaxPool	$0.25 \times N$
Dense	Products	$0.5 \times N_{eurons} \times N$
	Accumulators	$0.5 \times N_{eurons} \times N$
	Overall Accumulator	$(1\ A_{ccum}/L_t)$
	SoftMax	$(1\ A_{ccum}/L_t)$

where N represents the number of parallel kernel branches available and L_t represents the latency. These fixed-point operations are used to derive the number of operations per second as

$$Op = F_{clk}[6N + 0.25N + 0.5N_{eurons}N + 0.5N_{eurons}N] + F_{clk}(1A_{ccum}/L_t) \quad (5)$$

where F_{clk} is the clock frequency, which is 100 MHz for the Atlys board. With $N = 2$, $B = 1$ and $K = 100$, MAC's and comparison values are generated every $L_t = 630$ clock cycles as can be verified from (3). Higher hardware parallelism (N) will decrease this number. Subsitituing into (5), we get

$$Op = (100 \times 10^6)[6 \times 2 + 0.25 \times 2 + 0.5 \times 2 \times 2 + 0.5 \times 2 \times 2] + (100 \times 10^6) \times 1/630 \quad (6)$$

$$Op = 1.65GOp/s \quad (7)$$

Fig. 25 presents the throughput corresponding to multiple hardware configurations of parallel branches and convolutional kernels. In contrast to the latency, Fig. 24, which is impacted by both the number of branches and the number of kernels, the number of operations per second does not depend on the number of kernels. On the other hand, it increases almost linearly with the number of parallel branches.

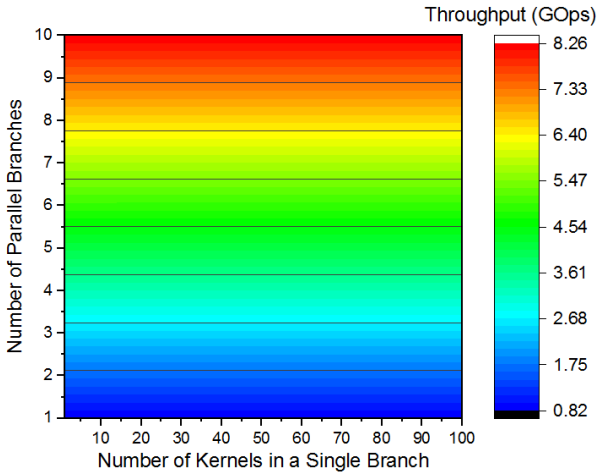

FIGURE 25. Throughput contour plots as a function of hardware parallel branches and kernels in the queue.

VI. MODULE RE-UTILIZATION AND DATA QUANTIZATION

One of the most important principles used in the BioCNN design is that of resource re-utilization. In this section, more details are given regarding the various design options for two important re-utilization features of the BioCNN design, namely, buffer swapping and kernel swapping. The important issue of data quantization and its impact on BioCNN performance is also discussed in this section.

A. BUFFER SWAPPING

Buffer swapping is embedded in the BioCNN architecture by partitioning the input of a given frame into multiple parts as illustrated in Fig. 26. The different partitions are handled according to Algorithm 2, and their results are time-multiplexed to the output of the frame. As described Subsection IV-B, the multiple partitions are overlapped (7.5) to include edge computations and ensure consistency. Fig. 26 also shows the zero paddings (7.6) at the external boundaries of the full frame.

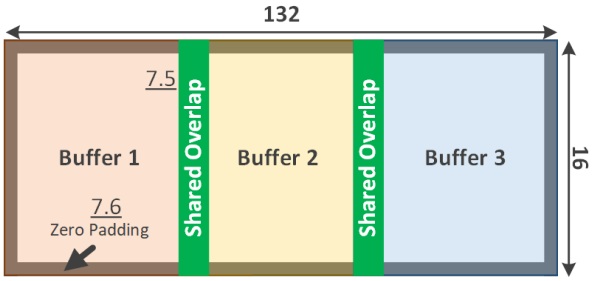

FIGURE 26. Buffer swapping layout.

The impact on logic resource utilization for various partition options is illustrated in Fig. 27. Only the case having three partitions is within the limits of the Atlys board.

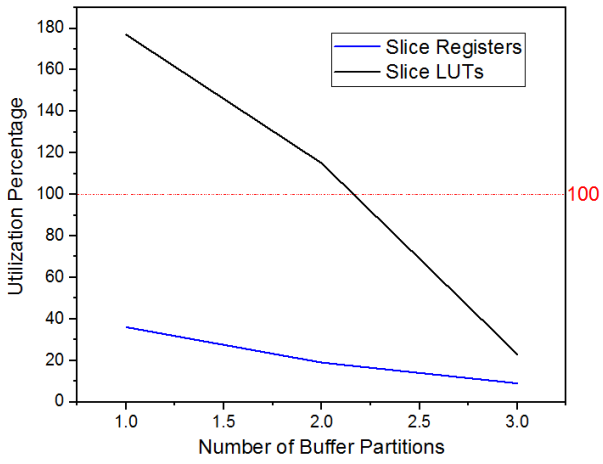

FIGURE 27. Logic utilization for different buffer partitions.

B. KERNEL SWAPPING

As described in Algorithm 2, adding a hardware branch such as (7.1) in Fig. 28 boosts the throughput due to the concurrent processing of pending kernels in the queue. The pending kernel queue is equally divided into two sub-queues

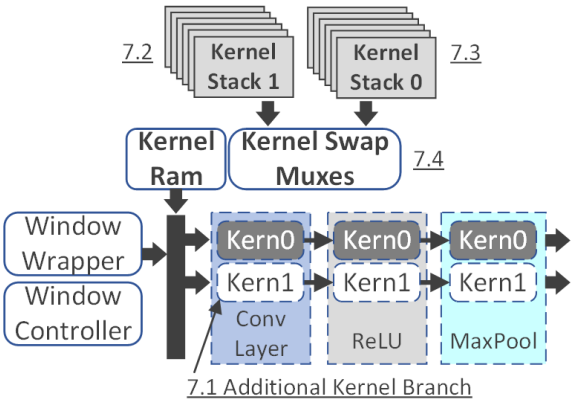

FIGURE 28. Hardware implementation of kernel swapping.

(7.2) and (7.3). The elements from the two sub-queues are sent to the parallel branches through a series of multiplexers (7.4), and the overall BioCNN controller ensures the proper assembly of the output results of each branch.

1) PARALLEL BRANCH OPTIONS

An architecture exploration study has been conducted to evaluate the logic cost of adding parallel branches to BioCNN. Architectures with 1 to 10 parallel branches have been implemented and evaluated. Fig. 29 shows the linear dependence of various FPGA logic resources on the number of hardware branches. The Atlys board imposed a limit of 10 hardware branches for BioCNN. The Lookup Tables (LUTs) usage is 90% when the limit is reached.

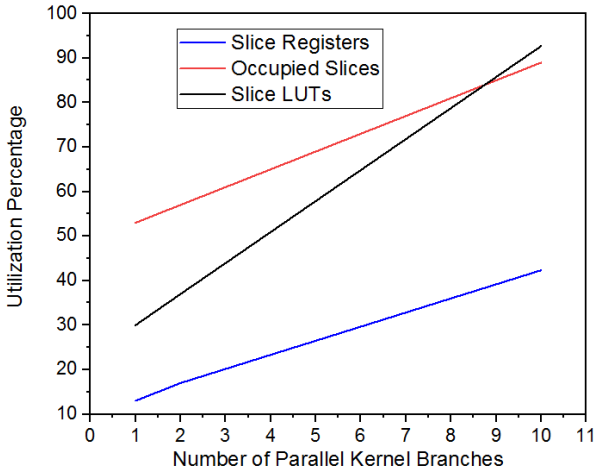

FIGURE 29. Resources impact of adding a parallel branch to the BioCNN architecture.

2) SEQUENTIAL KERNEL OPTIONS

Another BioCNN architectural study has been conducted to evaluate the logic cost of increasing the number of kernels in a single branch. The results have been summarized in Fig. 30 which shows that such impact is quite minimal and that more kernels can be handled with no effect on the logic resources, but of course with a visible impact on the BioCNN throughput.

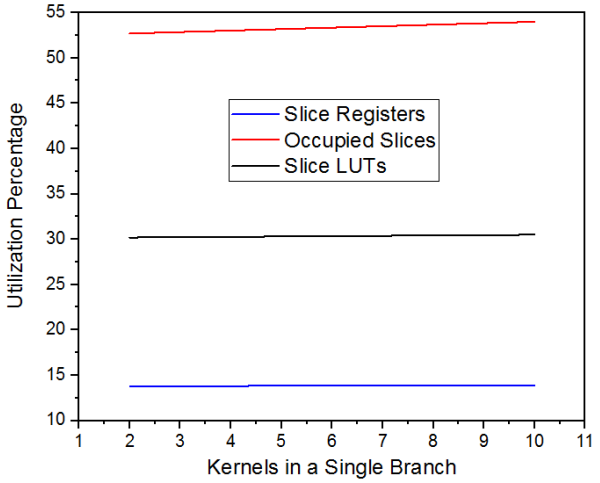

FIGURE 30. Resources impact of adding a sequential kernel to a single branch in the BioCNN architecture.

C. DATA REPRESENTATION IMPACT

The initial TensorFlow prototype of BioCNN used the FP32 option, which is the original TensorFlow option based on the assumption of pure software learning models, and abundant hardware resources. When a hardware implementation on a constrained edge node is sought, such choice may become prohibitive. This is evident in Table 6 for the memory estimation expressed for a single kernel and two cases with different number of representation bits n. The memory required to hold an EEG frame needs to consider 16×132 values, where 16 corresponds to the 14 channels plus the two padding rows, and 132 corresponds to the 129 frequency bins plus 3 padding columns to ensure even distribution in the three buffer partitions. Regarding the convolutional weights, it is required to allocate the size of the kernel (3×3) plus one bias value, all multiplied by the number of kernels K. For the dense weights, it is required to consider a single weight per each unit in the flattened output of the maxpooling layer, which is 8×66 because it is half of the initial EEG Frame plus one bias. This needs to be considered for each kernel value K, and for each neuron in the dense layer.

TABLE 6. Memory estimation for three different modules.

Layer	Detail	K1(Kb) 32 bits	K1(Kb) 16 bits
EEG Frame	$[(16 \times 132) \cdot 1]\frac{n}{8}$	8.45	4.22
Conv Weights #1	$((3 \times 3) \cdot 1 + 1) \cdot K(\frac{n}{8})$	4	2
Dense Weights #1	$((8 \times 66) \cdot 1 + 1) \cdot K \cdot 2(\frac{n}{8})$	422.4	211.2
Total Mem.		434.85	217.42

TensorFlow provides a methodology for data quantization that helps to evaluate the impact of finite-precision arithmetic on the inference phase. The training phase is typically performed using full data representation. In the testing phase, the data representation size may be lowered to a 16-bit or 8-bit integer. Quantization is treated as an additional noise source distorting the input data. Fortunately, the convolutions used in convolutional neural networks often possess filtering properties that can address quantization noise quite appropriately. When data quantization is considered in the context of image recognition, it amounts to assigning high-quality pictures to the training phase, and low-quality pictures to the recognition phase. Table 7 contrasts the BioCNN accuracy on the FEXD dataset when FD16 is used instead of FP32. The accuracy degradation is about 6% for the valence and arousal classification.

TABLE 7. Quantization on classification accuracy for FEXD and DEAP.

Bits	Dataset	Valence	Arousal	Validation
32	FEXD	66.96%	73.34%	10-fold cv
16	FEXD	63.61%	68.93%	10-fold cv
32	DEAP	83.12%	76.78%	10-fold cv
16	DEAP	77.57%	71.25%	10-fold cv

VII. DISCUSSION

To the best of our knowledge, this article is the first detailed report on the fine-grained hardware design of a compact, real-time emotion classifier using EEG signals and the emerging paradigm of intelligent edge computing. Most of the prior research has focused on software implementations under the assumption of abundant hardware resources and off-line operation. The main goals of almost all prior work have been accuracy improvements and the generation of standard benchmarks for evaluating and comparing various emotion detection algorithms. Table 4 summarizes the most significant contributions and compare them with ours using the metrics considered in this article. Even though the present work is focused on a hardware emotion detector using a constrained edge node, its classifier is competitive with the top software emotion classifiers under the DEAP benchmark, and has the potential to provide a low-footprint hardware implementation to the top DEAP classifier reported so far, which is based on an ensemble of CNNs [4].

To the extent of our knowledge, there is only one publication in the open literature that reports on a hardware implementation of an EEG-based emotion classifier [14], in which two approaches for labeling the classes have been considered: (1) Binary, in which only the valence is reported, and (2) Quaternary, in which a single classifier addresses both the valence and arousal dimensions. As presented in Table 4, most of the state-of-the-art contributions report the valence and arousal independently, which is in line with our own BioCNN results. An accuracy comparison with [14] is possible only for the valence classifier. Additionally, in [14] a bias has been introduced on the DEAP dataset by removing samples whose manual ratings were close to the center of the emotion circumplex. Unfortunately, such bias makes the reported DEAP accuracy of [14] unsuitable for comparison with our work or any other state-of-the-art contribution. Setting this bias issue aside, the on-chip valence accuracy of [14] is 83.36% using a fixed point (FP) representation of 24 bits, whereas it is 77.57% in this work where a fixed point representation of 16 bits is used. Although BioCNN uses FP16 vs. FP24 in [14], its accuracy is still quite competitive.

TABLE 8. Figure of merit among hardware classifier of emotions.

Metrics	[14]	This Work
Clock (MHz)	70	100
Platform	ASIC 28nm	Atlys FPGA
Precision (bits)	24	16
Latency (ms)	0.026	0.93
Classes	4 / 2	4
Logic Gates	3.091M	-
Neurons (bits)	10.168K	-
LUTs	-	26229
Flip-Flops	-	15180
Classifier	Multiphase CNN	CNN
Power (W)	0.0295	0.15

As for throughput, power, and energy efficiency, the comparisons between our results and those of [14] are hampered due to the following considerations:

1) Throughput: In [14], the specific throughput (Operations per second) of the classifier is not provided.
2) Energy efficiency: The absence of throughput in [14] further prevents the energy efficiency comparison, which is calculated based on how much power is needed to achieve a particular number of operations per unit of time.
3) Power: The power consumption of the two hardware implementations is not directly comparable since the FPGA typically consumes more power than an equivalent ASIC design.
4) Resources: The number of standard cells in the ASIC design of [14] is not directly comparable with the number of LUTs in our FPGA. However, our BioCNN is implemented in the low-end Spartan-6 LX45, whose total number of equivalent logic gates does not exceed 3840 cells. This number is small relative to the number of neuron bits and logic gates needed in [14] for implementing its CNN training and testing engines.

Among CNN hardware implementations, BioCNN is unique in that it emphasizes module re-utilization to satisfy the stringent resource constraints of the edge node. BioCNN minimizes logic resources using aggressive pipelining not only within the CNN module itself but also across all the BioCNN modules, including the maxpooling and output modules. The result is a coherent data path from the feature map input to the classifier output with minimum memory footprint, competitive energy efficiency, and an inference

accuracy that is at par with software systems. Note that BioCNN does not use any CPU cores anywhere along the data path. Both BioCNN latency and throughput are well within the specifications of a wearable biomedical device for real-time classification. BioCNN has further provided another reference design point for CNN hardware that is based on massive pipelining, data partitioning, and hardware parallelism. This reference point is to be contrasted with the other CNN hardware approaches based on distributed near-memory computing. The comparisons given in Table 9 with the state-of-the-art use a BioCNN clock frequency of 100MHz.

TABLE 9. Figures of merit and BioCNN comparisons with the state-of-the-art.

Metrics	[28]	[29]	[30]	[31]	This Work
Clock[a]	150	100	110	150	100
Platform	Z-7045	Z-7020	VX690T	VX690T	Atlys
Precision[b]	$FP16$	$FP16$	$FP8, 16$	$FP16$	$FP16$
DSP[c]	780	206	1897	2688	32
BRAM[c]	486	144	2715	2365	10
LUT[c]	182616	38136	324793	320203	26229
FF[c]	127653	42618	315465	309227	15180
Throughput[d]	136.97	39.78	213.7	910.2	1.65
Power[e]	9.63	2.03	27.7	30.4	0.15
Efficiency[f]	14.22	19.6	7.71	29.94	11.0

[a] MHz [b] bits [c] Used Resources [d] GOps [e] W [f] GOps/W

Through its features of kernel queuing, data partitioning, and hardware parallelization, BioCNN enables precise quantification of the tradeoff between throughput and FPGA resource utilization. Different applications are expected to use different points of the tradeoff curve, with many biomedical applications such as wearable EEG-based emotion detection, trading off throughput for more compact architectures in line with the constraints of edge nodes. As Table 9 shows, BioCNN uses 10X to 100X fewer hardware resources than competing FPGA-based CNN's, under a competitive energy efficiency of $11 GOps/W$, and achieves a throughput of 1.65 *GOps* that is in line with the real-time specification of the wearable device. Its emotion inference accuracies are competitive with the top software-based emotion detectors.

VIII. CONCLUSION

In this article, BioCNN, a novel FPGA-based CNN implementation specifically geared toward wearable biomedical applications has been presented. In contrast to the mainstream near-memory computing paradigm of high-throughput CNN's, BioCNN emphasizes aggressive pipelining, low memory footprint, and resource re-utilization. The CNN architecture is prototyped on a low-end FPGA board (Xilinx Altys) to mimic the constraints of a wearable, biomedical edge node. It features two novel algorithms for pipelined maxpooling and the serialized processing of maxpooled values in a dense output layer. BioCNN does not require any CPU cores, thus reducing further its area and logic requirements. A crucial step in the application of BioCNN to various biomedical domains is the feature engineering needed in each domain and the feature map that needs to be generated as input to BioCNN. Potential candidates for exploring the generation of feature maps that are BioCNN-compatible include ECG diagnostics, blood pressure monitoring, and hearing aids. In the context of the EEG-based emotion detection application, the BioCNN classifier achieves a classification accuracy of 77.57% and 71.25% for valence and arousal, respectively, using the DEAP dataset. These hardware accuracies are achieved with a reduced FP16 representation and hardware classifier with a small footprint, yet they are competitive with the state-of-the-art software classifiers. BioCNN is the first hardware emotion classifier to report hardware accuracies for the binary arousal problem. The BioCNN throughput of $1.65 GOps$ is in line with the real-time requirement of a wearable device with an energy efficiency of $11\ GOps/W$ and power consumption of $150mW$. Finally, BioCNN latency of less than 1 *ms* is much smaller than the 150 *ms* required for human or human-machine interactions.

REFERENCES

[1] S. Koelstra, C. Muhl, M. Soleymani, J.-S. Lee, A. Yazdani, T. Ebrahimi, T. Pun, A. Nijholt, and I. Patras, "DEAP: A database for emotion analysis ;Using physiological signals," *IEEE Trans. Affect. Comput.*, vol. 3, no. 1, pp. 18–31, Jan. 2012.

[2] X. Li, D. Song, P. Zhang, G. Yu, Y. Hou, and B. Hu, "Emotion recognition from multi-channel EEG data through convolutional recurrent neural network," in *Proc. IEEE Int. Conf. Bioinf. Biomed. (BIBM)*, Dec. 2016, pp. 352–359.

[3] Y. Yang, Q. Wu, M. Qiu, Y. Wang, and X. Chen, "Emotion recognition from multi-channel EEG through parallel convolutional recurrent neural network," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–7.

[4] H. Yang, J. Han, and K. Min, "A multi-column CNN model for emotion recognition from EEG signals," *Sensors*, vol. 19, no. 21, p. 4736, Oct. 2019.

[5] Y. Luo, Q. Fu, J. Xie, Y. Qin, G. Wu, J. Liu, F. Jiang, Y. Cao, and X. Ding, "EEG-based emotion classification using spiking neural networks," *IEEE Access*, vol. 8, pp. 46007–46016, 2020.

[6] S. Tripathi, S. Acharya, R. Sharma, S. Mittal, and S. Bhattacharya, "Using deep and convolutional neural networks for accurate emotion classification on DEAP dataset," in *Proc. Innov. Appl. Artif. Intell. 29th IAAI Conf.*, 2017, pp. 4746–4752. [Online]. Available: https://aaai.org/ocs/index.php/IAAI/IAAI17/paper/view/15007

[7] Y. Wang, Z. Huang, B. McCane, and P. Neo, "EmotioNet: A 3-D convolutional neural network for EEG-based emotion recognition," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–7.

[8] R.-N. Duan, J.-Y. Zhu, and B.-L. Lu, "Differential entropy feature for EEG-based emotion classification," in *Proc. 6th Int. IEEE/EMBS Conf. Neural Eng. (NER)*, Nov. 2013, pp. 81–84.

[9] P. C. Petrantonakis and L. J. Hadjileontiadis, *EEG-Based Emotion Recognition Using Advanced Signal Processing Techniques*. Hoboken, NJ, USA: Wiley, 2015, ch. 11, pp. 269–293, doi: 10.1002/9781118910566.

[10] S. Katsigiannis and N. Ramzan, "DREAMER: A database for emotion recognition through EEG and ECG signals from wireless low-cost off-the-shelf devices," *IEEE J. Biomed. Health Informat.*, vol. 22, no. 1, pp. 98–107, Jan. 2018.

[11] M. Soleymani, J. Lichtenauer, T. Pun, and M. Pantic, "A multimodal database for affect recognition and implicit tagging," *IEEE Trans. Affect. Comput.*, vol. 3, no. 1, pp. 42–55, Jan. 2012.

[12] H. A. Gonzalez, J. Yoo, and I. M. Elfadel, "EEG-based emotion detection using unsupervised transfer learning," in *Proc. 41st Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. (EMBC)*, Jul. 2019, pp. 694–697.

[13] Y. Wei, J. Zhou, Y. Wang, Y. Liu, Q. Liu, J. Luo, C. Wang, F. Ren, and L. Huang, "A review of algorithm & hardware design for AI-based biomedical applications," *IEEE Trans. Biomed. Circuits Syst.*, vol. 14, no. 2, pp. 145–163, Apr. 2020.

[14] W.-C. Fang, K.-Y. Wang, N. Fahier, Y.-L. Ho, and Y.-D. Huang, "Development and validation of an EEG-based real-time emotion recognition system using edge AI computing platform with convolutional neural network system-on-chip design," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 4, pp. 645–657, Dec. 2019.

[15] H. A. Gonzalez, S. M. J. Yoo, and I. M. Elfadel, "An inference hardware accelerator for EEG-based emotion detection," in *Proc. 53th IEEE Int. Symp. Circuits Syst. (ISCAS)*, to be published.

[16] K. R. Scherer, "Trends and developments: Research on emotions," in *Social Science Information*. London, U.K.: SAGE, 2005, p. 0539.

[17] T. Anderson, "Circumplex models of personality and emotion. R. Plutchik & H. R. Conte (Eds). (1997). Washington, DC," *Amer. Psychol. Assoc.*, vol. 8, pp. 359–360, Nov. 2010.

[18] M. M. Bradley and P. J. Lang, "Measuring emotion: The self-assessment manikin and the semantic differential," *J. Behav. Therapy Experim. Psychiatry*, vol. 25, no. 1, pp. 49–59, Mar. 1994.

[19] C. A. Frantzidis, C. D. Lithari, A. B. Vivas, C. L. Papadelis, C. Pappas, and P. D. Bamidis, "Towards emotion aware computing: A study of arousal modulation with multichannel event-related potentials, delta oscillatory activity and skin conductivity responses," in *Proc. 8th IEEE Int. Conf. Bioinf. BioEng.*, Oct. 2008, pp. 1–6.

[20] R. J. Davidson, "Affective neuroscience and psychophysiology: Toward a synthesis," *Psychophysiology*, vol. 40, no. 5, pp. 655–665, Sep. 2003, doi: 10.1111/1469-8986.00067.

[21] M. Soleymani, S. Asghari-Esfeden, Y. Fu, and M. Pantic, "Analysis of EEG signals and facial expressions for continuous emotion detection," *IEEE Trans. Affect. Comput.*, vol. 7, no. 1, pp. 17–28, Jan. 2016.

[22] M. K. Abadi, R. Subramanian, S. M. Kia, P. Avesani, I. Patras, and N. Sebe, "DECAF: MEG-based multimodal database for decoding affective physiological responses," *IEEE Trans. Affect. Comput.*, vol. 6, no. 3, pp. 209–222, Jul. 2015.

[23] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning on heterogeneous systems," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, 2016, pp. 265–284.

[24] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An accelerator for compressed-sparse convolutional neural networks," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 27–40.

[25] A. Lavin and S. Gray, "Fast algorithms for convolutional neural networks," 2015, *arXiv:1509.09308*. [Online]. Available: http://arxiv.org/abs/1509.09308

[26] L. Lu and Y. Liang, "SpWA: An efficient sparse winograd convolutional neural networks accelerator on FPGAs," in *Proc. 55th Annu. Design Autom. Conf.*, Jun. 2018, p. 135, doi: 10.1145/3195970.3196120.

[27] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[28] J. Qiu, S. Song, Y. Wang, H. Yang, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, and N. Xu, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays FPGA*, 2016, pp. 26–35, doi: 10.1145/2847263.2847265.

[29] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "A power-efficient and high performance FPGA accelerator for convolutional neural networks: Work-in-progress," in *Proc. 12th IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synth. Companion CODES*, 2017, p. 16, doi: 10.1145/3125502.3125534.

[30] C. Huang, S. Ni, and G. Chen, "A layer-based structured design of CNN on FPGA," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Oct. 2017, pp. 1037–1040.

[31] L. Gong, C. Wang, X. Li, H. Chen, and X. Zhou, "MALOC: A fully pipelined FPGA accelerator for convolutional neural networks with all layers mapped on chip," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 11, pp. 2601–2612, Nov. 2018.

HECTOR A. GONZALEZ (Student Member, IEEE) received the B.Sc. degree in electronics engineering from the Electrical Engineering and Electronics Department, National University of Colombia, Bogotá, Colombia, in 2011, and the M.Sc. degree in microsystems engineering from Khalifa University (Masdar Institute Campus), in collaboration with MIT, Abu Dhabi, United Arab Emirates, in 2017. He is currently pursuing the Ph.D. degree with the Chair for Highly Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden, Germany. From 2011 to 2015, he held multiple industrial positions as a senior engineer and the maintenance planning leader in the area of instrumentation and control electronics at the Massy Energy and Wood Group PSN. He is working as a Scientific Staff at the Chair for Highly Parallel VLSI-Systems and Neuromorphic Circuits, Technische Universität Dresden. He is the author of several publications covering the areas of digital system verification, digital signal processing for frequency-modulated continuous-wave radar, emotion detection, and industry applications. His current research interests include neuromorphic computing, hardware design for machine learning algorithms, biomedical applications, and digital signal processing for radar systems. His honors and awards include eight fee remission recognition for obtaining the best GPA in the Faculty of Engineering, National University of Colombia, two Enrolments of Honor for getting the best average in the entire Electrical Engineering and Electronics Department, National University of Colombia, and a scholarship given by the Malaysian Technical Cooperation Programme (MTCP) for an immersion training in supervision in the Oil and Gas Industry, Institut Teknologi Petroleum PETRONAS (INSTEP), and the Petronas Leadership Centre, Kuala Terengganu, Malaysia. He was awarded the Richard Newton Fellowship at the Design Automation Conference, in 2018, San Francisco, CA, USA.

SHAHZAD MUZAFFAR (Student Member, IEEE) received the B.S. degree in telecommunication engineering from the Electrical Engineering Department, National University of Computer and Emerging Sciences, Lahore, Pakistan, in 2008, and the M.Sc. degree in microsystems engineering and the Ph.D. degree in interdisciplinary engineering from Khalifa University (Masdar Campus), in collaboration with MIT, in 2015 and 2018, respectively. From 2011 to 2013, he held the position of a senior development engineer at the ESD Division, Mentor Graphics, Lahore. From 2009 to 2011, he was Design Engineer (Team Lead) with the Center for Excellence in FPGA/ASIC Research (CEFAR), National University of Sciences and Technology, Islamabad, Pakistan. Since he joined the Masdar Institute in 2013, he has been a Graduate Research Assistant with the Abu Dhabi SRC Center of Excellence on Energy-Efficient Electronics Systems (ACE^{4}S), where he has been the System Design Lead of one of two ACE^{4}S technology demonstrators. He is currently a Postdoctoral Fellow with the Electrical and Computer Engineering (ECE) Department, Khalifa University, Abu Dhabi, United Arab Emirates. He is the author of more than ten refereed publications and book chapters and the inventor of five U.S. patents (pending). His awards include the Best M.Sc. Thesis Award at the Masdar Institute, for 2014–2015, academic year and the Third Best-Paper Prize at the 2017 Cyber Security Awareness Week, New York University Abu Dhabi, Abu Dhabi.

JERALD YOO (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2002, 2007, and 2010, respectively.

From 2010 to 2016, he was with the Department of Electrical Engineering and Computer Science, Masdar Institute, Abu Dhabi, United Arab Emirates, where he was an Associate Professor. From 2010 to 2011, he was a Visiting Scholar with the Microsystems Technology Laboratories (MTL), Massachusetts Institute of Technology (MIT). Since 2017, he has been with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore, where he is currently an Associate Professor. He has pioneered research on low-energy body-area-networks (BAN) for communication/powering and wearable body sensor network using planar-fashionable circuit board for continuous health monitoring systems. He has authored book chapters in *Bio-Medical CMOS ICs* (Springer, 2010) and *Enabling the Internet of Things: From Integrated Circuits to Networks* (Springer, 2017). His current research interests include low-energy circuit technology for wearable bio signal sensors, flexible circuit board platform, BAN communication and powering, ASIC for piezoelectric micromachined ultrasonic transducers (pMUT), and system-on-chip (SoC) design to system realization for wearable healthcare applications.

Dr. Yoo serves as a Technical Program Committee Member of the IEEE International Solid-State Circuits Conference (ISSCC), ISSCC Student Research Preview (co-chair), IEEE Custom Integrated Circuits Conference (CICC, Emerging Technologies Subcommittee Chair), and IEEE Asian Solid-State Circuits Conference (A-SSCC, Emerging Technologies and Applications Subcommittee Chair). He is also an Analog Signal Processing Technical Committee Member of the IEEE Circuits and Systems Society. He was a recipient or a co-recipient of several awards, including the IEEE International Symposium on Circuits and Systems (ISCAS) 2015 Best Paper Award (BioCAS Track), the ISCAS 2015 Runner-Up Best Student Paper Award, the Masdar Institute Best Research Award, in 2015, and the IEEE Asian Solid-State Circuits Conference (A-SSCC) Outstanding Design Award, in 2005. He was the founding Vice Chair of the IEEE SSCS United Arab Emirates (UAE) Chapter. He is an IEEE Circuits and Systems Society (CASS) Distinguished Lecturer. He also served as the IEEE Solid-State Circuits Society (SSCS) Distinguished Lecturer, from 2017 to 2018.

IBRAHIM (ABE) M. ELFADEL (Senior Member, IEEE) received the Ph.D. degree from MIT, in 1993. From May 2014 to January 2019, he was the Program Manager of the TwinLab MEMS, a joint collaboration with GlobalFoundries and the Singapore Institute of Microelectronics on micro-electromechanical systems. From May 2013 to May 2018, he was the Founding Co-Director of the Abu Dhabi Center of Excellence on Energy-Efficient Electronic Systems (ACE^{4}S). From November 2012 to October 2015, he was also the Founding Co-Director of the Mubadala's TwinLab 3DSC, a joint research center on 3-D integrated circuits with the Technical University of Dresden, Germany. He also headed the Masdar Institute-Center for Microsystems (iMicro), from November 2013 to March 2016. From 1996 to 2010, he was with the corporate CAD organizations at IBM Research and the IBM Systems and Technology Group, Yorktown Heights, NY, USA, where he was involved in the research, development, and deployment of CAD tools and methodologies for IBM's high-end microprocessors. He is currently a Professor with the Department of Electrical Engineering and Computer Science, Khalifa University, Abu Dhabi, United Arab Emirates. His current research interests include the Internet of Things (IoT) platform prototyping, energy-efficient edge and cloud computing, the IoT communications, power and thermal management of multicore processors, low-power, embedded digital-signal processing, 3-D integration, and CAD for VLSI, MEMS, and Silicon Photonics. He was a recipient of six Invention Achievement Awards, one Outstanding Technical Achievement Award and the Research Division Award from IBM, for his contributions in the area of VLSI CAD. He is the inventor or co-inventor of 50 issued U.S. patents with several more pending. In 2014, he was a co-recipient of the D. O. Pederson Best Paper Award from the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN FOR INTEGRATED CIRCUITS AND SYSTEMS. In 2018, he received (with Prof. Mohammed Ismail) the SRC Board of Director Special Award for pioneering semiconductor research in Abu Dhabi. He has also served on the Technical Program Committees of several leading conferences, including ISCAS, DAC, ICCAD, ASPDAC, DATE, ICCD, ICECS, and MWSCAS. He was the General Co-Chair of the IFIP/IEEE 25th International Conference on Very Large Scale Integration (VLSI-SoC 2017), Abu Dhabi. He is the Lead Co-Editor of three books *3D Stacked Chips: From Emerging Processes to Heterogeneous Systems* (Springer, 2016), *The IoT Physical Layer: Design and Implementation* (Springer, 2019), and *Machine Learning in VLSI CAD* (Springer, 2019). From 2009 to 2013, he has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN FOR INTEGRATED CIRCUITS AND SYSTEMS. He is serving as Associate Editor for the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS and on the Editorial Board of the *Microelectronics Journal* (Elsevier).