ECG-TCN: Wearable Cardiac Arrhythmia Detection with a Temporal Convolutional Network

ECG-TCN: Wearable Cardiac Arrhythmia Detection with a Temporal Convolutional Network

Thorir Mar Ingolfsson*, Xiaying Wang*, Michael Hersche*, Alessio Burrello‡, Lukas Cavigelli†, Luca Benini*‡

*ETH Zürich, D-ITET, Switzerland ‡University of Bologna, DEI, Italy †Huawei Technologies, Zurich RC, Switzerland

***Abstract*—Personalized ubiquitous healthcare solutions require energy-efficient wearable platforms that provide an accurate classification of bio-signals while consuming low average power for long-term battery-operated use. Single lead electrocardiogram (ECG) signals provide the ability to detect, classify, and even predict cardiac arrhythmia. In this paper we propose a novel temporal convolutional network (TCN) that achieves high accuracy while still being feasible for wearable platform use. Experimental results on the ECG5000 dataset show that the TCN has a similar accuracy (94.2%) score as the state-of-the-art (SoA) network while achieving an improvement of 16.5% in the balanced accuracy score. This accurate classification is done with 27× fewer parameters and 37× less multiply-accumulate operations. We test our implementation on two publicly available platforms, the STM32L475, which is based on ARM Cortex M4F, and the GreenWaves Technologies GAP8 on the GAPuino board, based on 1+8 RISC-V CV32E40P cores. Measurements show that the GAP8 implementation respects the real-time constraints while consuming 0.10 mJ per inference. With 9.91 GMAC/s/W, it is 23.0× more energy-efficient and 46.85× faster than an implementation on the ARM Cortex M4F (0.43 GMAC/s/W). Overall, we obtain 8.1% higher accuracy while consuming 19.6× less energy and being 35.1× faster compared to a previous SoA embedded implementation.**

***Index Terms*—healthcare, time series classification, smart edge computing, machine learning, deep learning.**

I. Introduction

Wearable devices are becoming increasingly popular for health monitoring. They enable to continuously track vital parameters, assist their users to live healthy, and potentially detect health disorders [1]. For example, hard-to-diagnose diseases such as sleep apnoea or intermittent cardiac arrhythmia can be identified more easily, reliably, and earlier. Particularly high-risk patients can benefit from always-on cardiovascular monitoring where the early detection of cardiac arrhythmia, or a potential infarction, increase the chances of full recovery; and thus, can be life saving. Electrocardiogram (ECG) signals are commonly used to identify different types of heart diseases and particularly detect acute disease states. Traditionally, the acquired signal is either recorded but not analyzed in case of diagnostic devices, processed by a larger non-wearable device available in medical care facilities, or in the rare cases of wearable monitoring transmitted to a remote cloud where it is processed and analyzed. While the first have clear limits in capability and usability, the latter exhibits a high energy consumption for the data transmission and gives rise to privacy concerns associated with personal healthcare data.

With the recent technological advancements in low-power microprocessors, e.g., microcontrollers (MCUs) based on ARM Cortex M family or RISC-V parallel ultra-low power (PULP) platform [2], smart edge computing is becoming the new compute paradigm to overcome these limitations. Here, the sensor data is directly processed locally on the smart wearable device, right where it is collected [3]. On the data analysis side, deep learning has shown impressive performance leaps across many fields. For time series classification tasks, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are now setting the state-of-the-art (SoA) accuracy while simplifying the overall engineering process by omitting the development of application-specific engineered features [4]. With the increasing popularity of deep learning, many frameworks have become available to efficiently deploy neural networks not only in the cloud, but also to various edge devices. However, current comparison efforts on their effectiveness and usability have been limited, often hindered by device-specific deployment flows.

However, many deep learning models are also known to require notoriously large amounts of resources in terms of compute, power, and memory, which remain out of reach for low-power MCUs. Searching for smaller networks that fit the constraints of the platforms, engineers must accept a lower accuracy [5]. Hence, active research is on-going to design resource-efficient models that are compact. A viable option are temporal convolutional networks (TCNs) [6], [7] which have shown to achieve high accuracy in biosignal classification even with small model size and limited demand for resources [8].

In this paper, we propose a compact TCN for accurate time-series classification on embedded devices, demonstrated with the deployment and characterization of an ECG-based heartbeat classifier on multiple embedded platforms. The main contributions are:

- We propose a novel TCN for accurate heartbeat classification. On the ECG5000 dataset, our proposed model achieves similar accuracy as the SoA LSTM-FCN network [4] (94.2% vs. 94.1%) and higher balanced accuracy (89.0% vs 72.5%), while requiring an order-of-magnitude lower number of parameters (15 k vs 405 k) and multiply-accumulate operations (1 MMACs vs 37 MMACs).
- We quantize to INT-8 and deploy the model on an ARM Cortex M4F using both TFLite and CUBE.AI deployment framework, and on a RISC-V PULP-based GreenWaves Technologies (GWT) GAP8 [2] MCU using both the GWT-proprietary and closed-source GAPflow and the academic, free open-source NEMO/DORY tools [9].
- We report extensive throughput and power measurements, comparing various deployment frameworks and target

Corresponding emails: {thoriri, xiaywang}@iis.ee.ethz.ch

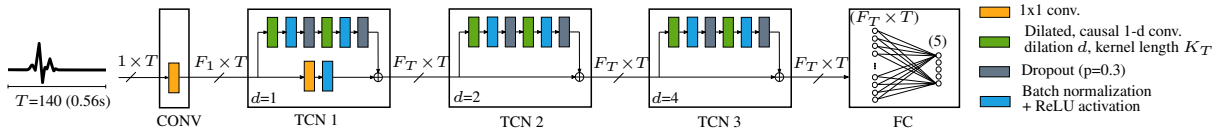

Fig. 1. TCN architecture for arrhythmia detection processing 140 ECG samples (0.56 s). The first convolutional block (CONV) has $F_1 = 2$ filters. TCN blocks 1–3 use a same number of filters $F_T = 11$, the same kernel length $K_T = 11$, and a variable dilation $d \in \{1, \ldots, 4\}$.

platforms. The proposed TCN achieves highest energy efficiency (9.91 GMAC/s/W) when being deployed on a GAP8 using NEMO and DORY. Compared to a prior art CNN+GRU deployed on a Cortex M4F using CMSIS-NN [5], our deployed TCN consumes 19.6× less energy (i.e., 6.0× more energy-efficient) and is 35.1× faster, while being 8.1% more accurate.

We open-source release our codes[1], including the enhancements made to the open-source DORY tool.

II. Methods

A. Temporal Convolutional Networks

The network proposed in this paper is commonly referred to as a temporal convolutional network (TCN) and is depicted in Fig. 1. Before passing the ECG signal through the TCN blocks, we expand the channels of the signal by passing them through a 1x1 convolutional layer. The residual block of our TCN consists of two layers of dilated convolutions, with batch normalization, non-linearity, and a dropout layer in-between the convolutions. Even though TCNs feature only 1D convolutions, they can still process 2D feature maps by considering the second channel dimension as the depth dimension. The residual skip connection adds the input to the output feature map, with the check that if the depth of the input and output is different, a 1×1 convolution is inserted. See Fig. 1 for an illustration of a residual block and the stacking of three residual blocks together.

By stacking residual blocks, the receptive field size (RFS) increases exponentially with each residual block, as the dilation in each subsequent block grows exponentially larger by the factor $d = 2^{L-1}$, where L is the number of the residual block. The RFS of the TCN is then determined by

$$\mathrm{RFS} = 1 + 2 \cdot (K_T - 1) \cdot (2^L - 1),$$

where K_T is the kernel size.

The residual block of our TCN differs from the one initially proposed by [6]:

- Batch normalization is used between convolutions instead of weight normalization as batch normalization has been shown to provide higher accuracy than weight normalization on various large scale networks [10].
- We use batch normalization on the residual branch wherever a 1×1 convolution is inserted.
- Instead of spatial dropout, normal dropout is used. As the TCN is applied after various convolutions, the adjacent samples within feature maps are not strongly correlated, hence it is beneficial to drop individual elements instead of entire 1D feature maps to regularize the activations.

[1]https://github.com/pulp-platform/ECG-TCN

B. Training Procedure

The model was developed, trained, and tested on an Nvidia GTX 1080 Ti GPU using both a TensorFlow (TF) and a PyTorch environment, to adapt to the different deployment frameworks. When training the proposed TCN, we use categorical cross-entropy loss and uniformly initialize filter kernels with the procedure introduced in [11]. The TCN model is trained on batches of size 30 using an Adam optimizer with a learning rate of 0.001 for 20 epochs. The final network is chosen via cross-validation on the training set and the final accuracy is reported on the test set.

C. Hardware Deployment

We target two platforms to be able to compare performance and find the best-suited microprocessor.

1) GAP8: The first platform is GWT GAPuino with GAP8, a RISC-V-based PULP platform with two compute domains: a fabric controller with one CV32E40P core for control tasks and 512 kB L2 memory extendable via a HyperBus interface and a cluster domain with 8 CV32E40P cores for parallel computation of highly demanding workloads and 80 kB directly accessible L1 memory. The frequency ranges from 32 kHz to 250 MHz. We compare two deployment frameworks:

- *GAPflow*: We quantized our network using TFLite and deployed it using GAPflow. It comprises of NNTool, which parses and prepares TFLite models, and AutoTiler, which converts the parsed model into optimized C code that can be executed efficiently in parallel by the 8 cores in the cluster. Since dilated causal convolutions are currently not supported, we replaced the dilated causal convolutional blocks as seen in Fig. 1 in the already trained model with the functionally identical sequence of a padding layer which pads the signal on the left side with $d \cdot (K_T - 1)$ zeros (causal padding), followed by a 1D convolution layer with a zero-stuffed weight kernel to imitate dilation. Moreover, the batch normalization layers are also removed without affecting the accuracy.
- *NEMO/DORY*: We used NEMO [12] to quantize our network to INT-8 and deployed it using DORY [9]. The open-source tool automatically generates C code to manage the two-level memory, L1 and L2, and execute convolutional kernels while maximizing the data locality. Similar to AutoTiler, DORY uses tiling and double-buffering to hide memory movements during kernel computation, acting as a software cache and minimizing the memory latency. While DORY employs PULP-NN kernels [13] optimized for 2D convolutions, in our work, we extended DORY with a novel, optimized 1D kernel version of the original 2D kernels, inserting a variable dilation parameter and optimizing the parallelization together with

the internal loop execution. These kernels plug into the DORY templates for automatic network code generation. Further, we adapted the DORY memory allocation to support convolutions on two parallel branches of the network, as present in our residual blocks.

2) ARM Cortex M4F: The second platform is the B-L475E-IOT01A STM32L4 Discovery kit, which is based on an ARM Cortex M4F core with 1 MB of flash memory, 128 kB of SRAM, and clocked at 80 MHz. We explore three different ways of deploying networks on the Cortex M4F and discuss their advantages and disadvantages:

- *TFLite Micro:* It provides a library and a tool flow to convert trained TF models to MCU-compatible code with some limitations for non-frequently used layers. We built TFLite Micro from source with flags for the Cortex M4F architecture and indicating that TFLite should use the optimized CMSIS-NN kernels provided by ARM. The model is built by calling layers from the resulting TFLite Micro library. With this method, it is possible to run quantized INT-8 TFLite models on the ARM Cortex M4F.
- CUBE.AI *with TFLite:* The use of CUBE.AI allows the automatic conversion of pre-trained neural networks and integration of generated optimized library into project code. The latest version of CUBE.AI takes a quantized INT-8 TFLite model and automatically parses the network graph, and generates an optimized code to run the network on inputs. It also can take a floating-point model made with Keras as an input and generates code to run the model on the MCU.
- CUBE.AI *with Keras:* This method is similar to the previous CUBE.AI-based flow but starts from a Keras model instead. However, not all the layers are quantized to an INT-8 representation in this mode.

As for GAPflow, manual padding is added to the dilated causal convolutions for the deployments on ARM since this operation is currently supported by neither CUBE.AI nor TFLite.

D. Performance Metrics

We evaluate the models according to the classification accuracy, which is the ratio between correctly classified trials and the total number of trials in the test set. The ECG5000 dataset we use in this paper is highly imbalanced, having only two samples of one of the classes in the whole training set. Therefore, we also consider the balanced accuracy score defined as $(\text{Sensitivity} + \text{Specificity})/2$. To measure how computationally intensive each network is we also compute the number of MACs for inference.

III. EXPERIMENTAL RESULTS

We evaluate our methods on the ECG5000 dataset introduced in [14] and based on data from the BIDMC Congestive Heart Failure Database (chfdb) [15], [16]. The record has 17 998 834 data points, containing 92 584 heartbeats. The data was then pre-processed in two steps. Each heartbeat was extracted and then made equal length using interpolation. Five classes are annotated, corresponding to the following labels: Normal (N), R-on-T Premature Ventricular Contraction (R-on-T PVC), Premature Ventricular Contraction (PVC), Supraventricular Premature or Ectopic beat (SP or EB), and unclassified beat (UB). Five thousand heartbeats were then extracted at random. The dataset was then split into a training set (500 data points) and a testing set (4500 data points) to report accuracy.

A. Accuracy Results

Table I summarizes the accuracy and balanced accuracy scores of our proposed network and the comparison to related works. The previously best performing network architecture on the ECG5000 dataset was the LSTM-FCN [4] which is a concatenation of an LSTM and a CNN network. We reproduced the results from the authors and further investigated the two branches of the network by splitting them, training them, and optimizing them as separate networks to fully evaluate and compare to the SoA. Four other network architectures from [17] are then included in the comparison in Table I as well as a baseline of one nearest neighbor classifier. Our model is 1.7% more accurate and excels by 34.1% in balanced accuracy compared to the baseline proposed by the original authors of the selected dataset [18].

Our TCN has similar accuracy results as the previous SoA LSTM-FCN. By looking at the balanced accuracy score, we see that our TCN achieves much better results (16.5% better) than the previous SoA and robustly classify the underrepresented classes in the dataset. Also, when considering the size and complexity of the models in Table I, we observe that our TCN requires 27× fewer parameters (15 k vs 405 k) and 37× lower number of MACs (1 MMACs vs 37 MMACs), compared to the LSTM-FCN. Using only the LSTM reduces the number of parameters and MACs, but also yields to a lower accuracy (93.1%) and balanced accuracy (68.9%).

Next, we consider the classification performance of the quantized TCN models, provided in Table II, and compare it to a quantized CNN+GRU that was trained on a different dataset which distinguishes between five classes too [5]. Quantizing the TCN weights and activations from float-32 to INT-8 representation yields a negligible accuracy loss of 0.2% and 0.4% using TFLite and NEMO, respectively. Similarly, the work in [5] has seen a 0.4% accuracy loss after quantizing the CNN+GRU model form floating-point to INT-8 representation. The TCN quantized with NEMO has a 8.1% higher accuracy than the quantized CNN+GRU.

B. Power and Energy

We perform power measurements to assess the energy consumption of our TCN implementations using five different

TABLE I
ACCURACY SCORES OF METHODS ON THE ECG5000 DATASET

Architecture	Acc. [%]	Bal. Acc. [%]	#Param.	MACs
TCN (ours)	**94.2**	**89.0**	**14 883**	1 030 260
LSTM-FCN [4]	94.1	72.5	404 741	36 982 656
CNN [4]	93.4	81.5	266 373	36 844 160
LSTM [4]	93.1	68.9	138 373	**138 496**
1-NN (L2 dist.) [18]	92.5	54.9	70 000	140 000
AttLSTM-CNNs [17]	85.4	-	-	-
Attention LSTM [17]	85.3	-	-	-
CNNs1D [17]	84.8	-	-	-
LSTM [17]	85.4	-	-	-

TABLE II
COMPARISON BETWEEN TCN ON GAP8, TCN ON ARM CORTEX M4F, AND RELATED WORK

Network	TCN (ours)					CNN+GRU [5]
Platform	GAP8		STM32L475			nRF52832
MCU	1+8×CV32E40P @100 MHz		1×Cortex M4F @80 MHz			1×Cortex M4F @64 MHz
Deployment framework	NEMO/DORY	GAPflow	TFLite	CUBE.AI TFLite	CUBE.AI Keras	CMSIS-NN
Dataset	ECG5000	ECG5000	ECG5000	ECG5000	ECG5000	[5]
Input size	1×140	1×140	1×140	1×140	1×140	1×256
MACs	1 030 260	2 348 925	2 339 994	2 339 994	2 354 114	3 221 244
Memory (weights & act.)	26.63 kB	25.03 kB	35.86 kB	35.86 kB	113.40 kB	210.00 kB
Full-prec. accuracy [%]	94.2	94.2	94.2	94.2	94.2	86.1
Quantized accuracy [%]	93.8	94.0	94.0	94.0	not quantized	85.7
Time/inference [ms]	2.7	20.2	188.0	126.5	374.3	94.8
Throughput [MMAC/s]	381.58	116.28	12.45	18.50	6.29	33.98
MACs/cycle	3.795	1.157	0.162	0.231	0.078	0.531
Power [mW]	38.52	41.67	45.54	42.75	47.68	20.65
Energy/inference [mJ]	0.10	0.84	8.56	5.41	17.85	1.96
En. eff. [GMAC/s/W]	9.91	2.79	0.27	0.43	0.13	1.64

toolchains on two MCUs. Table II shows the comparison of the measurement between these methods; additionally, we include the power measurements from [5]. When looking at the ARM Cortex M4F implementation of the TCN, using a quantized TFLite model with CUBE.AI provides the best performance. It has a speedup of around $1.5\times$ with respect to using TFLite barebone on the MCU. Both GAP8 implementations outperform the ARM ones, with NEMO and DORY tools, augmented by our optimized 1D kernels, consuming the least amount of energy per inference (0.10 mJ) at highest energy efficiency (9.91 GMAC/s/W), while also providing the highest performance (381.58 MMAC/s) by executing one inference in only 2.7 ms. The best GAP8 implementation is approximately $23.0\times$ more energy-efficient than the best deployment of the TCN onto the ARM Cortex M4F. Additionally, comparing to [5], it is around $6.0\times$ more energy-efficient than the CMSIS-NN method used to implement their network on the ARM Cortex M4F, while consuming $19.6\times$ less energy.

IV. CONCLUSION

This paper presents a novel, lightweight TCN architecture for cardiac arrhythmia classification. The network achieves similar accuracy results as the SoA on the ECG5000 dataset while improving the balanced accuracy score by 16.5%. The TCN is then further quantized to INT-8 using two different quantization tools and implemented on two publicly available platforms, STM32L475 and GAP8. The INT-8 quantized TCN implementation on the GAP8 platform with the NEMO quantization scheme and DORY tool for deployment shows superior results, being $23.0\times$ more energy-efficient than the implementation on the ARM Cortex M4F. The implementation has a throughput of 381.58 MMAC/s, consumes 0.10 mJ per inference, and takes only 2.7 ms to output a prediction. Integrating an ADC such as [19] with GAP8 would allow us to build an ECG smartpatch consuming 382.23 μW on average. Assuming a real-world scenario with one inference per second, i.e an average heartbeat rate of 60bpm, the system would last two years on a CR2450 coin cell battery.

ACKNOWLEDGMENT

This project was supported in part by the Swiss National Science Foundation (Project PEDESITE) under grant agreement CRSII5_193813 and in part by the Swiss Data Science Center PhD Fellowship under grant ID P18-04.

REFERENCES

[1] J. Dunn *et al.*, "Wearables and the medical revolution," *Personalized Medicine*, vol. 15, 09 2018.
[2] E. Flamand *et al.*, "GAP-8: A RISC-V SoC for AI at the Edge of the IoT," in *Proc. IEEE ASAP*, 2018.
[3] M. Magno *et al.*, "InfiniWolf: Energy Efficient Smart Bracelet for Edge Computing with Dual Source Energy Harvesting," in *Proc. DATE*, 2020.
[4] F. Karim *et al.*, "LSTM Fully Convolutional Networks for Time Series Classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.
[5] A. Faraone *et al.*, "Convolutional-Recurrent Neural Networks on Low-Power Wearable Platforms for Cardiac Arrhythmia Detection," in *Proc. IEEE AICAS*, 2020.
[6] S. Bai *et al.*, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *arXiv:1803.01271*, 2018.
[7] M. Carreras *et al.*, "Optimizing temporal convolutional network inference on fpga-based accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, pp. 348–361, 2020.
[8] T. M. Ingolfsson *et al.*, "EEG-TCNet: An Accurate Temporal Convolutional Network for Embedded Motor-Imagery Brain–Machine Interfaces," in *Proc. IEEE SMC*, 2020.
[9] A. Burrello *et al.*, "DORY: Automatic End-to-End Deployment of Real-World DNNs on Low-Cost IoT MCUs," *arXiv:2008.07127*, 2020.
[10] I. Gitman *et al.*, "Comparison of Batch Normalization and Weight Normalization Algorithms for the Large-scale Image Classification," *arXiv:1709.08145*, 2017.
[11] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *Proc. IEEE CVPR*, 2016.
[12] F. Conti, "Technical Report: NEMO DNN Quantization for Deployment Model," University of Bologna, Tech. Rep., 2020.
[13] A. Garofalo *et al.*, "PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190155, 2020.
[14] Y. Chen *et al.*, "A general framework for never-ending learning from time series streams," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, pp. 1622–1664, 2015.
[15] D. S. Baim *et al.*, "Survival of patients with severe congestive heart failure treated with oral milrinone," *Journal of the American College of Cardiology*, vol. 7, no. 3, pp. 661–670, 1986.
[16] A. L. Goldberger *et al.*, "Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals," *circulation*, vol. 101, no. 23, pp. e215–e220, 2000.
[17] Q. Du *et al.*, "Attention-based LSTM-CNNs For Time-series Classification," in *Proc. ACM SenSys*, 2018.
[18] Y. Chen *et al.*, "UCR Time Series Classification Archive," p. 14.
[19] M. Yip *et al.*, "A 0.6V 2.9µW mixed-signal front-end for ECG monitoring," in *Proc. IEEE VLSIC*, 2012.