FANNCortexM: An Open Source Toolkit for Deployment of Multi-layer Neural Networks on ARM Cortex-M Family Microcontrollers

Performance Analysis with Stress Detection.

Michele Magno, Lukas Cavigelli, Philipp Mayer, Ferdinand von Hagen, Luca Benini
Dept. of Information Technology and Electrical Engineering, ETH Zurich, Switzerland

***Abstract*—We present *FANNCortexM*, an open-source toolkit built upon the *Fast Artificial Neural Network (FANN)* library to run lightweight neural networks on ARM Cortex-M series microcontrollers. The toolkit takes a neural network trained with FANN and generates code targeted at execution on low-power microcontrollers either with a floating-point unit (i.e., ARM Cortex-M4F and M7F) or without a floating-point unit (i.e., ARM Cortex M0-M3). The toolkit is optimized in terms of memory and computational resources. We demonstrate its functionality on the basis of a sample application scenario performing stress detection on a wearable multi-sensor bracelet. Experimental results show a high classification accuracy of 96% for the target application scenario, and low latency of only a few microseconds while keeping the memory requirements (11kB flash storage, 36kB RAM) far below the limitations of the device. Power measurements show a power consumption of only 1.6mW, thus allowing continuous stress detection for 29 days.**

***Keywords*—Machine Learning, Low Power Design, Open-Source framework, Microcontrollers, Wearable, Stress Detection.**

I. INTRODUCTION

IoT devices require to be energy efficient, low cost and non-invasive to be effective, as they are often supplied by small size batteries. Moreover, the new trend of such devices is to be "smart" to make a decision on their own. There are many examples of smart devices that are already a commercial success, for instance, smart clothes, smart sensors in home automation and wearable sensors in healthcare systems [1]. Most of the commercial success of these smart devices is due to the advancements in low power circuits and systems as well as the availability of ubiquitous wireless connectivity [1], [2].

The IoT vision will bring billions of these devices in our life very soon. This "invasion" of devices will create fascinating challenges for industrial and academic researchers. Among others, there is a huge amount of data that these smart devices will produce and that needs to be processed [3]-[8]. Today, powerful cloud-based data analytics techniques are available, but they require an internet connection. However, pushing the analysis to the cloud is not possible in a wide range of application scenarios [3]. Thus, edge computing, where real-time feature extraction, analysis, classification, and local decision-making are carried out close to the sensors are becoming an essential ingredient to achieve a truly scalable and robust IoT infrastructure [9].

A promising approach to improve the intelligence of IoT devices is endowing them with the capability of running neural networks [7]-[10]. Recent results on these machine-learning models demonstrate impressive classification accuracy that in some cases even over-perform human accuracy [19], [20]. One very important feature of neural networks is their flexibility that makes the algorithms suitable for a wide range of applications. Lots of research efforts towards specialized hardware and optimized inference algorithms to run such neural networks on power-constrained devices have been made over the last few years [11]-[18]. However, there are very few examples of neural networks that are running on low power microcontrollers, which are the most common compute engines available at the edge of the IoT [19]-[24]. This is particularly the case, because the majority of approaches are using deep convolutional neural networks, which are extremely accurate and well-suited to classify image frames but require a huge amount of memory and computational resources. However, there are many application scenarios, especially dealing with low-bandwidth, low-power non-frame based sensors, where multi-layer fully connected and sparsely connected networks are more than enough [10][25].

The *Fast Artificial Neural Network* library (FANN) is a free open source neural network library [25], which implements multi-layer artificial neural networks (ANNs) in C with support for both fully connected and sparsely connected networks. This paper presents an open source framework for easy deployment of neural networks trained with FANN on ARM Cortex-M cores, which are the dominant processor core in most microcontrollers. Our framework runs on microcontrollers with a floating-point unit (i.e., ARM Cortex-M4F) or without a floating-point unit (i.e., ARM Cortex-M0+) [26]. We present both the framework and the tools to train artificial neural networks as well as experimental results based on a multi-sensors wearable device designed around an ARM Cortex-M4F processor (TI-MSP432). A working artificial neural network is presented for a challenging application scenario of stress detection on a wearable device [27]-[28]. Experimental results demonstrate the small memory footprint needed and the good accuracy achieved with an ANN with a 6-input, 3-class perceptron with 2 hidden layers. Moreover, we evaluated the biggest network fitting in the 256kB of memory available, with up to 1000 nodes and 47900 weights.

The main contributions of the paper are:

- We present an open source framework based on FANN, which allows building multi-layers artificial neural network-based classifier on all ARM Cortex M family processors both with and without a floating-point unit.

978-1-5386-4980-0/19/$31.00 ©2019 IEEE

- We implement a fully connected network for stress detection using a multi-sensor wearable device based on an ARM Cortex-M4F microcontroller.
- We present experimental results with in-field measurements in terms of memory usage, accuracy, the larger network that can be used and the power consumption.
- We have released the framework as open source software[1] to help engineers and academics to have a powerful and easy to use tool to deploy ANNs on ultra-low power devices and IoT devices.

The rest of the paper is organized as follows. We will start by providing an overview of the framework and discussing the properties of the Cortex-M series. Next, we introduce the emotion and stress detection application scenario. Thereafter, the experimental results are presented before concluding the paper.

II. Neural Network Toolkit Overview

Developing an intelligent sensor device with onboard processing capabilities encompasses many steps (cf. Figure 1). Development starts with specifying a clear target application and which platform and sensors to use, collecting data, and annotating it with the desired labels. In the next step, the data should be preprocessed, identifying and applying suitable feature extractors, optionally performing data augmentation, normalizing the data and preparing it in a suitable data format for training. Then the neural network has to be specified and trained, the network's hyperparameters and its structure have to be explored (number of layers and nodes, which activations), promising networks have to be trained, and the best network has to be identified. If the desired accuracy cannot be obtained, the previous steps have to be revisited (collection of more data, different feature extractors, changing the data augmentation for the training samples). Once the desired accuracy has been achieved, the network must be deployed on the device, converting the neural network to fixed-point (if no hardware floating-point support is available), developing an optimized implementation for the target device, cointegrating it with the sensor read-out and preprocessing, and measuring the resulting performance and power.

A. Artificial Neural Networks

In this work, we focus on multi-layer perceptrons, which are moderately resource-demanding and are used as classifiers after applying suitable non-learnable feature extractors. These multi-layer networks are much less demanding than deep convolutional networks in terms of memory and computing requirements, and they are still widely used and very effective in many application areas such as medical data analysis and in general applications based on sensors that do not produce images.

Multi-layer perceptrons consist of three or more layers of nodes: an input layer, one or more hidden layers, and an output layer. Each layer contains a fixed number of nodes, which (except for the inputs) is a weighted sum of the previous layer's nodes and a bias, followed by a non-linearity such as sigmoid or ReLU functions. The entire network is typically trained end-to-end by optimizing its parameters (weights and biases) using backpropagation and stochastic gradient descent, such that it maps the samples from the dataset to the labels to the best of its abilities (measured by a loss function). The multi-layer perceptron is specified by the number of hidden layers, the number of nodes within each layer, which non-linearities are used (e.g. sigmoid), and the loss function (e.g. cross-entropy loss or mean squared error).

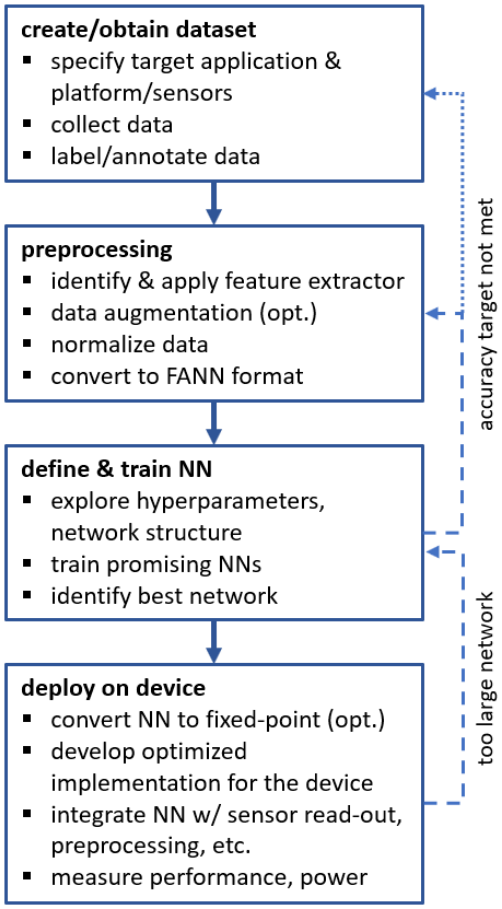

Figure 1: Process flow from concept to deployment.

B. Fast Artificial Neural Network (FANN) Library

FANN[2] [25], [29] is an easy-to-use, mature and well-documented framework to train and perform inference on multi-layer neural networks. It is all written in C, but has bindings to many languages, such as Python, MATLAB, Rust, etc. Due to its popularity, many graphical tools to aid training ANNs and selecting the right architecture and hyperparameters have become available. FANN also includes an automatic hyperparameter tuner and can optimize ANNs for fixed-point inference.

FANN uses a simple file format for storing the dataset and the trained ANN model. It does not support training on GPUs, which simplifies the framework and is not required for networks of a size range suitable for deployment on low-power microcontrollers. The CPU implementation is highly optimized with features such as cache optimizations and approximations of various activation functions as step-linear out-of-the-box.

C. Open-Source FANNCortexM

Our main contribution is to show that multi-layer ANNs can be implemented on ultra-low power microcontrollers, specifically on the ARM Cortex-M family. We have developed an optimized implementation of the inference step

[1] https://github.com/lukasc-ch/FANN-on-ARM

[2] http://leenissen.dk/fann/wp/

for all commonly used functions and layers, making use of ARM M-series-specific instructions. Along with it, we created a script for automatic conversion of the trained network to a directly callable dependency-free C function including a test method which applies samples from the dataset for verification and benchmarking. Notably, the generated code files include all parameters of the network to overcome the need for file system support. This allows for a very simple workflow:

1. Convert the data to the FANN standard format,
2. Train a neural network using FANN and save it,
3. Apply the conversion script to the ANN and dataset,
4. Call the `fann_type* fann_run(fann_type *input)` function from within the code.
5. Build your code together with the generated files and evaluate the resulting application.

Our converter supports fixed-point and floating-point models and can thus run also efficiently on microcontrollers without a floating-point unit. We have released the code as open source software online[3].

D. FANNTool to train the network

To facilitate training the neural network, to select an appropriate network architecture, to determine the number of layers and nodes, and choosing a suitable activation function, the FANN community provides a convenient utility: the FANNTool [11]. Its interface allows for easy modification of the network's hyperparameters, training method, a weight initialization method, and monitoring training progress. It further supports the fully-automated selection of the network's hyperparameters by iteratively modifying them. Figure 2 shows a screenshot of the FANNTool 1.2.

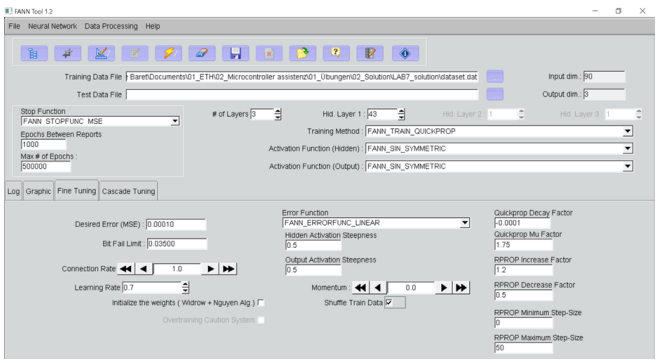

Figure 2: Screenshot of the FANNTool 1.2 user interface

III. Low-Power Embedded Processing: ARM Cortex-M Family

Low-power embedded systems based on microcontrollers are characterized by on-chip SRAM of few kB (256kB-512kB) and non-volatile flash memory of max 1-2MB. Today, most popular low-power microcontrollers are based on ARM Cortex-M processor cores. The ARM Cortex-M (M0; M3, M4; M7) family features different computational capabilities and operating frequencies, and they have power consumption in the mW range. The typical frequency is 16 MHz for the M0 and up to 200 MHz for the recent and more powerful M7. The majority of those processors have no floating-point unit and only the ARM Cortex-M4F and M7 implement a floating-point unit. The flash memory is typically used to preload the program code and static data, while the SRAM is used for the run-time code and main data memory. Thus, one of the constraints to be taken into account is the size of the non-volatile memory. Moreover, as microcontrollers are designed with low power and low cost in mind, operations per second and volatile memory footprint need to be taken into account. Finally, it is important to notice that some of ARM Cortex-M cores have an integrated digital signal processing (DSP) unit and a DSP-enhanced instruction set. This is the case of all the ARM Cortex-M3/basic) and M4 and M7. The DSP unit can be used to accelerate many of the operations used for signal processing and data analysis. Most notably, the Cortex-M4 and Cortex-M7 have integrated single instruction, multiple data (SIMD) instructions and multiply-accumulate operations that might be exploited to accelerate computation in neural networks.

The Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces [26]. CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals, simplifying software reuse, reducing the learning curve for microcontroller developers and reducing the time to market for a new device. Artificial neural networks have a high number of multiplications, so minimizing the time increases the efficiency of the solution. In the FANNCortexM toolkit, we used extensively point multiplication optimization function *arm_dot_prod* for floating point and not floating point, where we evaluated an an increase of execution time of 36% compared with the not using it, that shows the importance of CMSIS. Among others we have also used *arm_fill* and *arm_copy* that also gave an improvement in the range of 30% in execution time. Moreover, CMSIS optimizes the computational time of many functions such as DSP operation. The DSP library includes over 60 digital signal processing related functions that are optimized for the Cortex M processors. DSP functions can be very useful for features extraction (i.e. to perform fast Fourier transformations) but are not required for FANNCortex toolkit, which can run with optimized performance also in Arm-Cortex M0 or other processors without DSP.

IV. Emotion and Stress Detection

In order to evaluate FANNCortexM, we decided to use a challenging application use case where bio-signals are processed: the detection of the stress level of humans. Previous work on stress detection exists using different approaches and feature extractors to detect levels of stress. The authors of the dataset [30] used, achieve 97% accuracy predicting the stress level on 112 samples by using five-minute non-overlapping segments of the data [30]. They calculated in total 22 features of which nine are statistical, four spectral power features, eight skin conductivity features and heart rate variability. Many others reach comparable accuracy using machine-learning approaches on the same dataset with different feature sets. In [28] 8 features from the same dataset were extracted using characteristics of the ECG data. With 10-fold cross-validation and 3 classes, they reach up to 70.15% accuracy and with 2 classes 97.92%. Using only 1 feature and 2 classes they reach 100% accuracy. In [32], 99% accuracy was reached using heart rate, respiration rate,

foot galvanic skin response, and hand galvanic skin response. In [33], 76.93% accuracy is achieved using an ECG-based method. In [34] GSR data are used to perform stress detection. However, the authors conclude that more information besides GSR data is necessary to perform reliable stress detection.

A. Features Extraction

Previous works [30]-[34] are giving a very good evaluation on the most suitable features extraction for stress detection. However, all the features are extracted on a PC and are not considering the device limitation, particularly in terms of memory of a microcontroller. In our application scenario, the available RAM is limited to 64-128kB. Many previous works are recording several minutes of data before extracting features. However, with ECG data sampled at 250Hz, the 64kB of RAM would be filled in less than 90 seconds. Including additional sensors to measure e.g. the galvanic skin response (GSR) further shortens this time. Therefore, one important aspect for selecting suitable features is their compatibility with the resources available on the selected microcontroller unit (MCU).

In this work, we optimized the feature extraction and explored the minimum sampling time required for satisfying stress level classification. To evaluate the performance of the FANNCortexM, we focus only on GSR and ECG data. After an evaluation of different combinations, a total of 6 features based on GSR and ECG were chosen and implemented. The features were taken from the dataset by splitting it into subsets with equal stress level—transitions between different stress levels where generously left out. To generate a constant number of samples in the training and test sets, feature extraction used overlapping intervals for longer sampling periods. Most training sets contain 570 samples and most test sets 70 samples.

For the ECG data, we calculated four features: mean R-peak interval, RMSSD, SDSD, and NN50. Where RMSSD is the root mean square and SDSD the standard deviation of the successive differences between neighboring RR intervals. NN50 is the number of neighboring RR interval pairs which differ by more than 50ms. The R-peak is the most significant peak in an ECG measurement and mostly used for heart rate measurement. For GSR data we used the well-known technique detecting rising edges of the GSR signal and computing the length and height of the slope [34]. The total number of detected edges, the sum of the heights, the lengths of the slopes and the mean GSR value are the four remaining features of our dataset.

Verification showed a high accuracy of up to 96% for these features for split datasets, but accuracy dropped on 10-fold cross validation (cf. Figure 3). We thus normalized the data by subtracting the mean value and limited the range to plus/minus 1.0 to improve the generalization capability of the trained system across all our tests.

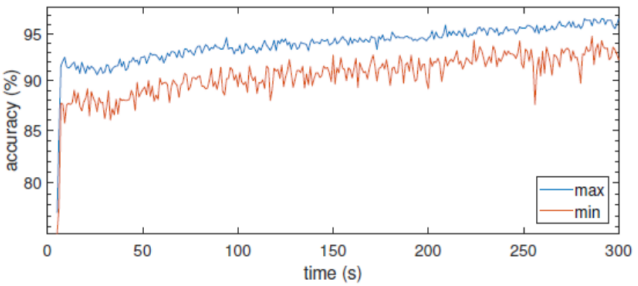

Figure 3. 10-fold cross validation for stress detection for different time frames.

B. Artificial Neural Network

Neural networks of various sizes and layouts were tested using the FANNTool. To find the best settings, we combined our results from Matlab tests with the settings FANNTool recommended, optimized the network size, and picked the solution with the best accuracy. It was found that all 6 input features with only one hidden layer consisting of three hidden neurons achieved the best accuracy. After the selection of the 6 input features and determining that a neural network with only one hidden layer with three neurons yielded best results, we made a performance comparison to determine how the sample period length impacts accuracy. A sweep over the sample periods from 3s to 300s using 10-fold cross-validation was used for accuracy calculation and for finding minimum sample periods required (Figure 3).

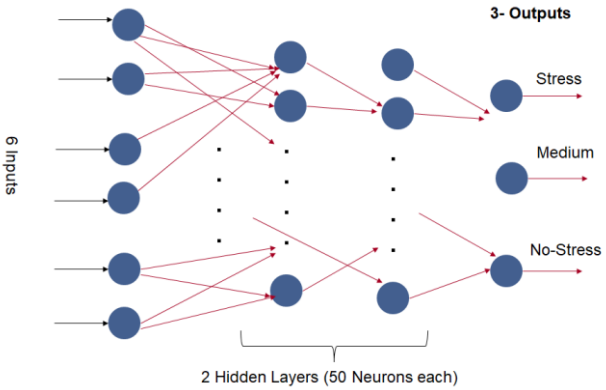

Figure 4: Network architecture designed for classifying 3 levels of stress.

V. EXPERIMENTAL RESULTS

We have conducted several experiments to evaluate the FANNCortexM library deployed on a complete hardware system with ARM Cortex-M4 processors from different companies: Texas Instruments MSP432, Ambiq Apollo 2 and ST STM32L4. Specifically, stress detection has been implemented and evaluated on the EmoTracker platform [38]. Figure 5 shows the block diagram of EmoTracker that uses a TI MSP432 as the MCU and includes electrocardiography (ECG), galvanic skin response (GSR), microphone, skin temperature and a 9-axis inertial measurement unit as sensors and Bluetooth Low Energy (BLE) as the communication interface.

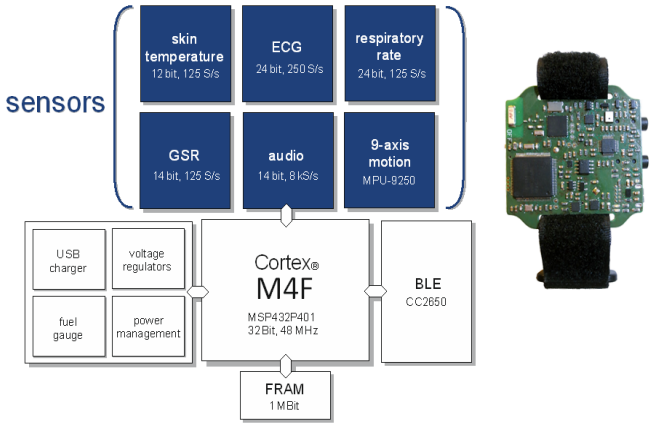

Figure 5: EmoTracker block diagram used for the system evaluation.

A. Memory Requirements

Our FANNCortexM library optimized using CMSIS requires only 11kB of flash storage and 36 bytes of RAM. The remaining memory consumption is dynamic and varies with the size of the network. All nodes are stored in RAM and require 4 bytes each. The memory allocation is done statically such that insufficient memory will be detected at compile time. All other data (weights, biases) are stored in flash memory and are not loaded into RAM. Every neuron requires 16 bytes of additional information which contains e.g. the activation function used. Each weight uses 4 bytes of flash storage. As neural networks are organized in layers and the number of neurons can vary from layer to layer, this consumes 8 additional bytes of flash memory per layer.

The biggest network we tested had 20 hidden layers with 50 nodes each, 5 input and 3 output nodes, for a total of 1000 nodes and 47900 weights. The reported memory consumption was 250 kB of the flash memory and the MSP432's on-chip RAM usage was approximately 50%.

B. Power and Energy Consumption

In the following, we discuss the contribution of the individual components of the system to its overall power (cf. Table 1):

- The *ECG front end* requires 740µW in active mode and is continuously running.
- The *GSR front end* continuously consumes 30µW.
- The *BLE* (CC2650) chip has to transmit very little data in our system. Its duty cycle limited by the maximum effective connection interval of 16s, after which a transfer lasting a minimum of 2.5ms is expected. During this phase the chip is in active mode and consumes 10mW; otherwise, it is in standby mode and dissipates a mere 3µW.
- The *microcontroller* (MSP432) is the most power hungry device, using 25mW in active mode and 10µW in standby. Every 4ms it reads two samples from the ECG front end over SPI at 1 MHz for 72µs. The GSR sensor is read every 8ms, which takes 122µs. The feature extraction takes 25µs per sample and can be performed during the data transfer of the next sample, and thus does not impose additional active time. After enough samples have been collected (e.g. after 6s), the feature extraction is finalized (96µs) and the classification using the neural network is performed (1385µs). It is thus in active mode 3.3% of the time to read the sensor data, which results in a power of 830µW and an additional 37µJ per classification (6µW@1 classif./6s).

The overall power consumption is thus 1.6mW. The coin cell used in the EmoTracker has a capacity of 1110mWh, which allows continuous operation of the device for 29 days without recharging. In many use cases, the stress level would not need to be determined every 6s and the entire device could be duty-cycled by a factor of 10x, providing a lifetime of 290 days.

TABLE I: POWER CONSUMPTION OVERVIEW

	active (48 MHz)	standby	average
ECG front end	740µW	160µW	740µW
GSR front end	30µW	--	30µW
MSP432	25mW	10µW	836µW
BLE (CC2650)	10mW	3µW	5µW
Total			**1611µW**

VI. CONCLUSION

Artificial intelligence and machine learning for low power Internet of Things devices that host microcontrollers are key technologies for near-sensor data analysis and decision making. Among others, deep learning and multi-layer artificial neural networks are showing incredible performance in term of accuracy in many applications. However, very few implementations and measurement results exist for low power microcontrollers. We have presented FANNCortexM, a toolkit to facilitate deployment of neural networks trained using the open source Fast Artificial Neural Network (FANN) library. Our toolkit is optimized for ARM Cortex-M processors, both with and without a floating-point unit, which will be compared in future works. With a case study on stress detection, we have demonstrated that artificial neural networks are suitable for deployment on ultra-low-power microcontrollers in terms of memory usage, compute time, and energy consumption. The toolkit is ready to be used to deploy neural networks for applications on low-power embedded systems.

ACKNOWLEDGMENTS

This work was in part funded by the Swiss National Science Foundation project 'Transient Computing Systems' (Nr.157048), and by the WSL Institute for Snow and Avalanche Research (SLF).

REFERENCES

[1] G Kaewkannate, K., & Kim, S. (2016). A comparison of wearable fitness devices. BMC public health, 16(1), 433.

[2] Rault, T., Bouabdallah, A., Challal, Y., & Marin, F. (2016). A survey of energy-efficient context recognition systems using wearable sensors for healthcare applications. Pervasive and Mobile Computing.

[3] Takhirov, Zafar, Joseph Wang, Venkatesh Saligrama, and Ajay Joshi. "Energy-Efficient Adaptive Classifier Design for Mobile Systems." In Proceedings of the 2016 International Symposium on Low Power Electronics and Design, pp. 52-57. ACM, 2016.

[4] Jeličić, Vana, et al. "An energy efficient multimodal wireless video sensor network with eZ430–RF2500 modules." 5th International Conference on Pervasive Computing and Applications. IEEE, 2010.

[5] Magno, Michele, Davide Brunelli, Lothar Thiele, and Luca Benini. "Adaptive power control for solar harvesting multimodal wireless smart camera." In 2009 Third ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), pp. 1-7. IEEE, 2009.

[6] Kerhet, A., Leonardi, F., Boni, A., Lombardo, P., Magno, M., & Benini, L. (2007, September). Distributed video surveillance using hardware-

friendly sparse large margin classifiers. In 2007 IEEE Conference on Advanced Video and Signal Based Surveillance (pp. 87-92). IEEE.

[7] Rahimi, Abbas, Pentti Kanerva, and Jan M. Rabaey. "A Robust and Energy-Efficient Classifier Using Brain-Inspired Hyperdimensional Computing." Proceedings of the 2016 International Symposium on Low Power Electronics and Design. ACM, 2016.

[8] He K, Zhang X., Ren S., Jian Sun J. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", arXiv:1502.01852.

[9] M. Magno, D. Brunelli, L. Sigrist, et al., InfiniTime: Multi-Sensor Wearable Bracelet with Human Body Harvesting, Sustain. Comput. Informatics, Elsevier, 2016.

[10] Magno, M., Pritz, M., Mayer, P. and Benini, L., 2017, June. DeepEmote: Towards multi-layer neural networks in a low power wearable multi-sensors bracelet. 7th IEEE International Workshop on In Advances in Sensors and Interfaces (IWASI), 2017 (pp. 32-37).

[11] Andri, R., Cavigelli, L., Rossi, D., Benini, L. (2016). YodaNN: An Ultra-Low Power Convolutional Neural Network Accelerator Based on Binary Weights. In Proc. IEEE ISVLSI (pp. 236–241).

[12] Cavigelli, L., Gschwend, D., Mayer, C., Willi, S., Muheim, B., Benini, L. (2015). Origami: A Convolutional Network Accelerator. In Proc. ACM GLSVLSI (pp. 199–204). ACM Press.

[13] Al Bahou, A., Karunaratne, G., el al. (2018). XNORBIN: A 95 TOp/s/W hardware accelerator for binary convolutional neural networks. In 2018 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS) (pp. 1–3). IEEE.

[14] Cavigelli, L., Degen, P., Benini, L. (2017). CBinfer: Change-Based Inference for Convolutional Neural Networks on Video Data. In Proc. ACM ICDSC.

[15] Andri, R., et al. (2018). Hyperdrive: A Systolically Scalable Binary-Weight CNN Inference Engine for mW IoT End-Nodes. In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (Vol. 2018–July, pp. 509–515). IEEE.

[16] L. Cavigelli, M. Magno, L. Benini (2015). Accelerating Real-Time Embedded Scene Labeling with Convolutional Networks. In Proc. ACM/IEEE DAC.

[17] Conti, F., Cavigelli, L., Paulin, G., Susmelj, I., & Benini, L. (2018). Chipmunk: A Systolically Scalable 0.9 mm2, 3.08GOp/s/mW @ 1.2 mW Accelerator For Near-Sensor Recurrent Neural Network Inference. In Proc. IEEE CICC.

[18] Cavigelli, L., & Benini, L. (2018). CBinfer: Exploiting Frame-to-Frame Locality for Faster Convolutional Network Inference on Video Streams.

[19] Rathore, H. "Artificial Neural Network." Mapping Biological Systems to Network Systems. Springer International Publishing, 2016. 79-96.

[20] F. Conti, D. Palossi, R. Andri, M. Magno and L. Benini, "Accelerated Visual Context Classification on a Low-Power Smartwatch," in IEEE Transactions on Human-Machine Systems, vol. 47, no. 1, pp. 19-30, Feb. 2017.

[21] Magno, Michele, Christian Spagnol, Luca Benini, and Emanuel Popovici. "A low power wireless node for contact and contactless heart monitoring." Microelectronics Journal 45, no. 12 (2014): 1656-1664.

[22] Mayer, Philipp, Michele Magno, and Luca Benini. "Self-Sustaining Acoustic Sensor With Programmable Pattern Recognition for Underwater Monitoring." IEEE Transactions on Instrumentation and Measurement (2019).

[23] Wang, X., Magno, M., Cavigelli, L., Mahmud, M., Cecchetto, C., Vassanelli, S., & Benini, L. (2018, September). Rat cortical layers classification extracting evoked local field potential images with implanted multi-electrode sensor. In 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom) (pp. 1-6). IEEE.

[24] S. Murali, F. Rincon, and D. Atienza, "A wearable device for physical and emotional health monitoring"in 2015 Computing in Cardiology Conference (CinC), Sept 2015,pp. 121-124.

[25] "Fann tool," 2007, [Online; accessed 10-December -2018]. [Online]. Available: https://code.google.com/archive/p/fanntool/

[26] ARM. (2017) Cortex microcontroller software interface standard. [Online]. Available: https://developer.arm.com/embedded/cmsis

[27] A. Muaremi, B. Amrich, G. Troster "Towards Measuring Stress with Smartphones and Wearable Devices During Workday and Sleep" in Journal of BioNanoScience, Vol 3, Issue 2, pp 172-183. June 2013.

[28] N. Keshan, P. V. Parimi, and I. Bichindaritz, "Machine learning for stress detection from ecg signals in automobile drivers," in 2015 IEEE International Conference on Big Data (Big Data), Oct 2015, pp. 2661–2669

[29] Nissen, Steffen. "Implementation of a fast artificial neural network library (fann)." *Report, Department of Computer Science University of Copenhagen (DIKU)* 31 (2003): 29

[30] P. R. Healey JA, "Dataset stress recognition in automobile drivers." Available: https://physionet.org/physiobank/database/drivedb/

[31] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet," Circulation, vol. 101, no. 23, pp. e215–e220, 2000.

[32] B. Alic, D. Sejdinovic, L. Gurbeta, and A. Badnjevic, "Classification of stress recognition using artificial neural network," in 2016 5th Mediterranean Conference on Embedded Computing (MECO), June 2016, pp. 297–300.

[33] K. T. Chui, K. F. Tsang, H. R. Chi, C. K. Wu, and B. W. K. Ling, "Electrocardiogram based classifier for driver drowsiness detection," in 2015 IEEE 13th International Conference on Industrial Informatics (INDIN), July 2015, pp. 600–603.

[34] J. Bakker, M. Pechenizkiy, and N. Sidorova, "What's your current stress level? Detection of stress patterns from gsr sensor data," in 2011 IEEE 11th International Conference on Data Mining Workshops, Dec 2011, pp. 573–580.

.