An Embedded CNN Implementation for On-Device ECG Analysis

Alwyn Burger
Embedded Systems
University of Duisburg-Essen
Duisburg, Germany
alwyn.burger@uni-due.de

Chao Qian
Embedded Systems
University of Duisburg-Essen
Duisburg, Germany
chao.qian.1119@stud.uni-due.de

Gregor Schiele
Embedded Systems
University of Duisburg-Essen
Duisburg, Germany
gregor.schiele@uni-due.de

Domenik Helms
OFFIS Institute for Computer Science
Oldenburg, Germany
domenik.helms@offis.de

***Abstract*—IoT systems that employ AI and neural networks for processing sensor data are usually dependent on an active connection to the cloud, or restricted to simplified models and techniques. By employing a highly optimised Convolutional Neural Network that can be executed directly on our heterogeneous IoT device, we can address both these problems without sacrificing accuracy. Additionally, leveraging the cloud when unknown or uncertain samples are seen allows us to continuously improve the model.**

Capable of running a "normal" model trained using frameworks such as TensorFlow, we show that our system can achieve accuracy of up to 97% in medical applications such as ECGs. This is done using our device that uses less than 200mW but can locally process more than 300 heart beats per second.

***Index Terms*—Internet of Things, neural networks, FPGA, Biomedical signal processing**

I. Introduction

Modern IoT often depend on small battery-powered devices to collect data directly from a user in their environment. For example, in medical applications a wearable is used that needs to continuously monitor the user's heart condition. That creates a strict power budget that limits which hardware components can be used when creating such a device.

The microcontrollers that are low powered enough to operate long term on a battery usually cannot perform complex algorithms such as inference with a Convolutional Neural Network (CNN) fast enough. Therefore, most of the currently used systems offload most of the calculations to the cloud, or compromise accuracy by using a less complex technique or neural network. Some applications such as medical monitoring cannot afford the risk of downtime when the connection to the cloud service is interrupted. In this case, a local processing solution is essential for ensuring the continuous operation of the system. High accuracy must still be maintained, or the health of the user could be at risk. That is why a less accurate solution – or one that cannot process incoming data at least as fast as it is collected – is unacceptable for such a critical application case.

Our approach is to use highly efficient local hardware acceleration. This allows us to perform CNN inference directly on an embedded device at very high accuracy. By deploying a trained model to a tiny local embedded FPGA, our device can locally analyse data from the sensors with no communication required with the cloud. An FPGA offers numerous Digital Signal Processing (DSP) components for the complex mathematics required, and Look-up Tables (LUTs) and registers for efficient data logic – without sacrificing flexibility. Another advantage is that it can be updated whenever required when new information becomes available and a better model can be created.

The first contribution of this work is a pervasive IoT system that uses this FPGA-based embedded device to monitor users' electrocardiograms (ECGs), but can be applied to other applications. It leverages the local intelligence of the device to classify every beat of the user's heart into various categories. It also uses cloud services to log a user's condition over time and continuously improve the system's performance. The second contribution is that we use a conventional CNN to allow developers an easy to way use our system. By using a generic framework such as TensorFlow [13], anyone could create a model that can be deployed to our system.

The requirements that we set for our system are as follows:

1) It should be applicable to a wide variety of applications,
2) It should provide industry standard inference accuracy of at least 95%,
3) The local device should be able to process incoming data at least at real-time, and
4) Its power consumption should be low enough to realistically be battery powered.

The term "real-time" here implies that the computation should be faster than the data is captured. In this case, that varies with the heart rate of the patient, which should not commonly exceed an average of 180 beats per minute. However, linking Requirement 3 to the timing requirement implies that we want

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the KI-Sprung LUTNet project (project number 16ES1125).

faster processing – so the device may save power by going into a sleep state.

We present our work by first discussing some related work in Section II, followed by an overview of the entire system in Section III. After explaining the details of how we optimised the CNN in Section IV, we present an evaluation of the system in Section V and some concluding thoughts in Section VI.

II. Related Work

One example of using cloud computing resources to augment a similar IoT system was created by Ke, Hengjin, et al. [1], who proposed a cloud-aided online EEG classification system for brain healthcare. It has also been used in other fields, e.g. [2] where researchers presented a cloud-based object detection algorithm which can detect not one or two but hundreds of objects in near real-time.

Other work has used local device intelligence with CNNs, such as Bettoni, Marco, et al. who performed image recognition with a ZYNQ FPGA platform [3]. Their project only needs 1.943MB memory but offers an inference runtime of almost 1.33 s per sample. Furthermore, a binarized neural network (BNN) has been used on an Intel Curie with only 15 KB of usable memory. By using this stripped down version of the neural network they achieve an inference runtime of under 50 ms per sample [4].

Instead reducing the size of the CNN by quantizing weights and biases as is done in a BNN, the CNN can be compressed by altering its mathematical model. In [5], the researchers proposed an XNOR-Network in which both the filters and the input to convolutional layers are binary. Moreover, they approximate convolutions using primarily binary operations. This results in $58\times$ faster convolutional operations (in terms of number of the high precision operations) and $32\times$ memory savings. Another approach rethinks the inference task with FPGA soft logic [6]. The FPGA implementations following the later approach achieved a mean area reduction of $1.81\times$ vs the state-of-the-art BNN architecture with unrolling and pruning. However, such radical compression creates a risk of loss of inference accuracy.

Our approach aims to create an IoT system that uses a conventionally created CNN model, can achieve very high inference accuracy, and operate at a considerably lower power usage than found in the literature. That involves not only the development of a bespoke local device, but also a redesigned IoT stack that can take advantage of the FPGA reconfigurability as well as the cloud resources available.

III. System Overview

To illustrate how our system can be used, consider a device that monitors a user named Steven's heart using an ECG sensor. Since he wants continuous monitoring, the device should be battery powered and mobile so it can be carried or worn. However, Steven has a problematic heart condition, and so it is critical his data be analysed as accurately as possible to make sure that no defects are missed or misdiagnosed.

Since Steven's internet connection is not 100% stable, he needs the system to be functional even when it is not available. His doctor needs a full overview of the monitoring data, and wants to keep an eye on all his patients' trends and histories.

Therefore, an IoT-based solution makes sense for him, where data is collected and uploaded to the cloud for databasing and further analysis. That does not require a constant uplink, and can be uploaded sporadically when convenient instead. Additionally, the doctor only needs the processed results that describe the heart's condition and not the raw data, which greatly reduces the amount of data that needs to be uploaded.

With human-centric applications such as this, it is important to keep updating the model as new information becomes available. With medical applications this is even more important when samples cannot be reliably recognised. These should be forwarded to a trained medical professional for manual inspection, and included in the training data once properly identified. This allows the system to continuously improve the model by expanding the training dataset. Once a newer and improved model is available, it gets sent to each device and the FPGA uses that instead. This offers a specific advantage over application specific integrated circuits (ASICS) that cannot change their configuration once deployed as easily.

Every new sample that is taken by the device is classified based on the current CNN model, which creates both a label for the current heart state and a measure of confidence in this label. If this confidence is high, we may assume that the sample is familiar enough to existing training data that the neural network can identify it. In the case of a detected heart condition, the device can directly provide feedback to the user without having to wait for any interaction with a cloud service and the communication delays that go with that. Routinely the device can also push summarised reports to the cloud for long term monitoring. Only the labels created during the classification are required for this step, making it considerably less data-intensive than uploading all raw data.

However, if the confidence value is low, we cannot trust the label assigned to it. We consider a sample to have low certainty when the softmax output of the CNN indicates low confidence in the assigned label (multiple labels having high likelihood instead of one singled out). That means that the sample should be forwarded to the cloud, so it can be explicitly labelled by a medical professional. At this point it can be incorporated into the training dataset to improve the next model to be trained. Over time, this should improve the resilience of the system by exposing it to a wider variety of data. It also ensures that all critical samples are reliably labelled, ensuring that no heart conditions are missed.

This IoT system can be used to create any application that uses an appropriate CNN for classification. Continuously updating the model on FPGA as new data is available allows it to adapt to changing environments. However, this relies on the local device to be able to process this model efficiently, which will be discussed in the following section.

IV. Optimised CNN Design

To deploy a CNN to an embedded FPGA that is in our power budget, its design must be altered to fit in the available resource constraints. Where a larger FPGA could simply instantiate the entire network simultaneously, our design requires a more minimalistic approach.

Inference-Only Implementation

Additionally, machine learning techniques such as CNNs consist of two phases: training and inference. Training is the process in which the network tries to learn from the input data and its labels. During this phase, the weights and biases that describe the CNN model are updated through backward propagation. During inference, the CNN infers/predicts the label of input data using the trained model. Since the training process is much more computationally expensive than inference, we choose to only place the inference on the device. While this would normally limit the system to offline training that cannot adapt to newly available data acquired after deployment, our IoT stack allows us to avoid this limitation.

Parameter and Connection Loading

When using CNNs on cloud servers that have copious RAM and CPU resources, all parameters (weights and biases) and input/output connections can be passed between layers directly. This means that the entire layer receives its input simultaneously, and every layer can simply be kept in RAM to be instantly available. Besides, modern CPUs can use SIMD (single-instruction-multiple-data) techniques to load multiple values into wide vector registers and simultaneously compute them in one instruction.

On an embedded FPGA the number of computational resources (DSP slices) and bandwidth of data buses (LUT slices) are inherently limited. To save computational resources, we must load input/output connections from memory as they are needed. To improve the speed of this step, all intermediate connections between layers are stored in Block-RAM on the FPGA. This is very fast local on-chip memory that offers reduced power and increased performance over externally connected memory.

Optimised Static Paremeters

On the FPGA the weights and biases are implemented into LUTs. While this means that they cannot be changed once the design is synthesized, it allows the compiler to further optimise resource consumption based on their values. Even considering bit-level optimisations, this can significantly improve the implemented size of the CNN.

Layer Combination

Lastly, CNNs commonly only integrate the activation operations into the convolution layer, while the pooling layer is implemented independently. On an FPGA this would require additional resources (usually block RAMs) for the intermediate data. Therefore, these have been combined in our design of the CNN.

Fixed Point Logic

According to a document from Xilinx [7], FPGA designs benefit from using fixed point data types over float point due to a reduction in logic resources, lower power, shorter latency, etc. Multiple researchers have proved that an accurate fixed point quantization of the CNNs would not cause any loss accuracy [8], [9].

Softmax Layer

As a final note, the traditional design includes a softmax layer which uses an exponential function to calculate likelihoods and certainty of the inference. On an FPGA this would involve either using a very large LUT or a Taylor expansion. Since this step is not required for local classification, this layer is deferred to the MCU or cloud. Even without it, we can choose the most likely classification and a measure of its certainty.

The structural optimisations of loading parameters and intermediate connections only when required – combined with the integration of the convolution and activation layers – allow us to fit a much larger CNN than is possible with a traditional design. This makes our system applicable to more applications, since the user merely has to convert an existing CNN model into this representation. This does not alter the behaviour of the model, and therefore performs exactly like "normal" models.

V. Evaluation

To evaluate our system – and in particular the intelligent local device – we should firstly revisit our requirements. As detailed in Section I, they are for the system to be applicable to a variety of applications, to achieve an accuracy greater than 95%, and for the local device to process new data faster than real-time without leaving the battery-limited power budget.

The system's flexibility to different applications has been discussed in the previous sections. In short, our custom CNN implementation optimises the available FPGA resources to support larger networks, allowing the system to address more complex problems. Using a "normal" CNN model greatly simplifies adapting the system to a new application. Additionally, the IoT stack used allows us to support changing and expanding problems.

Testing the other requirements cannot be done without a real implementation. Therefore, we created a real-world experiment using our Elastic Node embedded experimental platform [10], [11] and the popular MIT-BIH Arrhythmia database [12]. It contains over 40 thousand samples labelled with six classifications. These include normal (N), paced beat (/), and atrial premature beat (A) – the last two indicating specific irregularities in the beat in question.

Some basic preprocessing is used to denoise each sample. The Wavelet Transform is chosen since regular filters like FIR filters are not effective in removing noise in a non-stationary signal such as an ECG signal. Next, the dataset should be balanced for additional stability. When the number of samples for each class in the training is not equal, the data set is

imbalanced, which can cause reduced inference accuracy for minority class.

Once this data is split 2:1 between the training and testing data, it is used to train a traditional floating-point CNN using Tensorflow [13]. Testing this model results in an overall accuracy of 97% Overall, the classification of the CNN is on par with current solutions, and provides a good target for our embedded solution to attempt to achieve.

What remains to be evaluated is how well a fixed point based FPGA model can match these results. Note that we describe the fixed point representation with a notation of (x, y), where x is the number of fractional bits (representing numbers < 1) and y is the total width in bits.

A. Fixed Point Bitwidth

To evaluate the effect of the fractional bits x on the system's accuracy, it is varied from 0 to 16 and tested using the full testing set (13876 samples). This is done using the FPGA simulator built into Vivado, which executes the full CNN inference with all parameters and variables in the relevant fixed point.

The resulting accuracy, recall, and f1 score are shown in Figure 1 and all tend to their respective values for the floating point model. This shows that a larger fixed point representation can adequately represent the values required for inference.

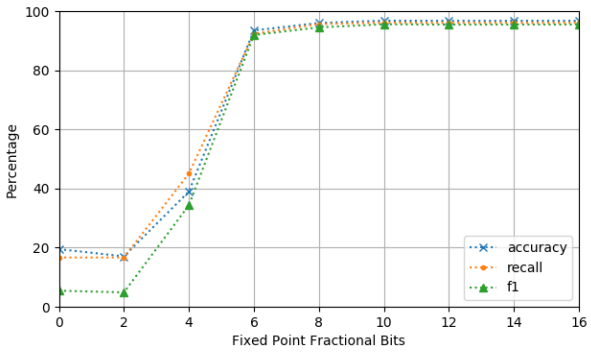

Fig. 1. Accuracy for different FP

As an example, the confusion matrix for $x = 12$ is shown in Figure 2. This shows the inference accuracy for each class, as well as the distribution of incorrect classifications. This exactly matches the original results obtained using the floating point Tensorflow model, proving that this fixed point approximation is accurate enough for this application.

Before finally choosing the correct fixed point representation for this application, we must first evaluate the effect it has on the FPGA resource consumption. An unnecessarily large representation may increase the power consumption for a diminishing return on the accuracy.

B. FPGA Utilisation

We compare the resource consumption of the different representations by synthesizing each of them into an FPGA

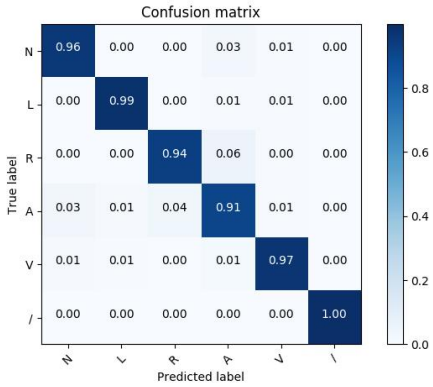

Fig. 2. Inference Confusion matrix for fixed point (12, 24)

configuration. Three different FPGAs from the Spartan 7 family from Xilinx are used, with varying available resources.

As shown in Figure 3, the number of DSP elements increase with larger FP representations. That is because each one has a fixed number of bits, and when this is surpassed by an operation additional DSPs are required.

The S25 and S50 devices have 80 and 120 DSP slices available respectively, which is much higher than the 20 available on the smaller S15 device. That means that the S15 cannot create some of the larger fixed point representations, and even for a (20,40) fixed point all of the available resources are in use.

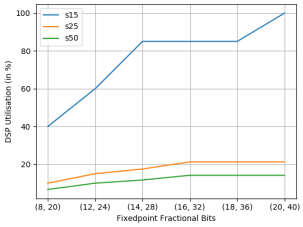

Fig. 3. DSP utilisation for different Fixed Point

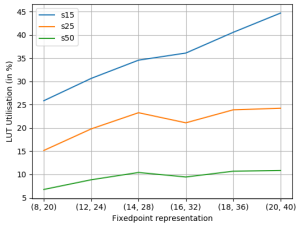

Fig. 4. LUT utilisation for different Fixed Point

Similarly, the percentage of LUTs for different devices is shown in Figure 4. As the number of LUTs available in these FPGAs is multiple orders of magnitude larger than the number of DSP elements, this is not a limiting factor as even the largest version fits comfortably in the LUTs available on the S15. However, using a larger percentage of the available resources on an FPGA increases the dynamic power consumption, which will be inspected next.

C. FPGA Power Consumption

The power estimate provided by Xilinx's Vivado software for each of the FPGAs is shown in Figure 5. This clearly shows the near-linear increase in power consumption as the dynamic power usage increases. Importantly, the static power of the different FPGAs considered is vastly different.

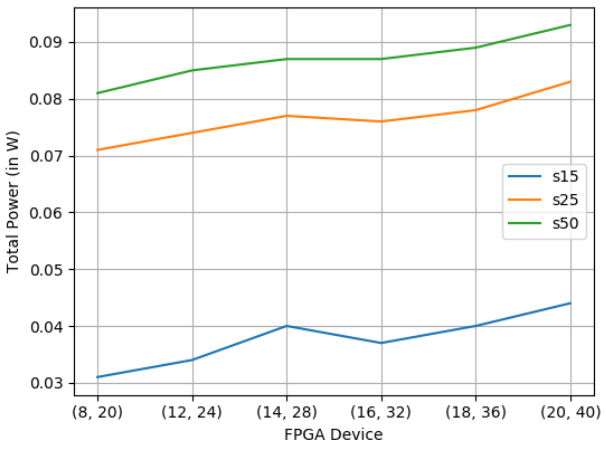

Fig. 5. Power consumption estimates

This makes the S15 considerably more efficient for smaller models, but it is rather limiting when larger CNNs are required. By choosing the correct FPGA, the user can optimise the device to their use case. This will also impact the speed at which the system can operate, since fully utilising an FPGA's resources can reduce its maximum operating frequency.

D. Local Device Power Breakdown

To validate the power estimates retrieved above, we deployed a specific CNN to two versions of our Elastic Node platform, one with an S15 and one with S25. Based on the fixed point accuracy experiments above, we chose a representation of (12,24). Based on an inspection of the intermediate values in the model, an integer bit size of 12 was chosen.

For the main components on the board, the active and the power usage is given in Table I. This idle usage shows that it can operate long term when the FPGA is not required, which during this time has zero usage as it is switched off.

TABLE I
ELASTIC NODE POWER BREAKDOWN

Component	Idle	Active S15	Active S25
MCU	2.3 mW	75.4 mW	78.8 mW
FPGA internal	0 mW	16.9 mW	26.6 mW
FPGA aux	0 mW	12.3 mW	18.8 mW
FPGA IO	0 mW	73.4 mW	74.6 mW
Total	2.3 mW	168 mW	198.8 mW

One can notice that these values are somewhat larger than the estimate provided in Section V-C. This can largely be attributed to assumptions made by the Vivado software, which has limited information about pins connected to other devices such as the MCU or flash memory. However, the device still operates under 200mW when fully active, which combined with a conservative duty cycle can easily be battery powered.

E. Processing Time

Our requirement for real time computation of incoming sensor data, and relying on a duty cycle to maximise FPGA sleep time, emphasises the local processing time for CNN inference.

1) Processing Speed Model: A timing model is presented here to estimate the throughput and performance of the system. This is needed as our design does not follow the usual CPU-based design of multiple instructions per neuron, or the large FPGA-based design of multiple neurons pipelined together. By calculating the number of operations required, an accurate estimation can be created for the processing time of the full CNN.

The simplified model for the CNN's processing time can be given by

$$t_{CNN} = t_{clock} \times n_{total} = t_{clock} \times (n_{c_1} + n_{c_2} + n_{ga} + n_{fc}) \quad (1)$$

where t_{clock} is the period of the clock, n_{c_1}, n_{c_2} are the number of operations required for the first and second convolution layers respectively. They can be calculated using

$$n_{c_i} = f \times K \times S \quad (2)$$

by using the number of features f, the kernel size K, and the number of convolution steps S. As we are using a stride of 1, the steps for layer c_x can simply be given by using H as the input height: $S_{c_x} = H_x - K + 1$. The height of every subsequent layer is then halved through a max pool operation.

Similarly the operations required for the global averaging layer n_{gc} can be written as

$$n_{gc} = f \times (H_{gc} + 1) \quad (3)$$

. Lastly, for the fully connected layer the formula is

$$n_{fc} = f \times C \quad (4)$$

for C being the number of classes being classified.

Using this model we can calculate the total number of operations required for our CNN. It classifies $C = 6$ classes with $f = 18$ features, yielding a total of $n_{total} = 143262$ operations. To calculate the processing rate and throughput, we also require the clock frequency of the device. Therefore, we deployed the same model used in Section V-D onto the Elastic Node to test its speed.

F. Real World Experiments

The CNN was able to be operated at 32MHz and 50MHz for both the S15 and S25 FPGA devices. The achieved processing speed is shown in Figure 6 with the estimates created using our model discussed above.

The real world results are only slightly increased from the estimate, since it includes overhead from the MCU such as retrieving results before the next sample can be started. The similarity here shows that by using Equation 1 a user can create an accurate estimate for the performance they can expect from using the Elastic Node when using their own model.

At a clock frequency at 32MHz, a throughput of 215.98 samples/s is achieved, and at 50MHz 335.24 samples/s. This

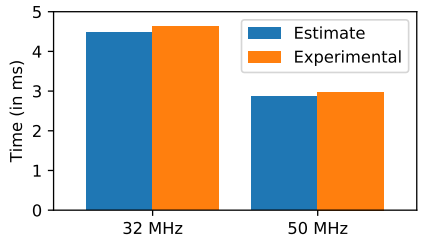

Fig. 6. Estimated and real world performance results

shows that the overall performance of the system is considerably faster than real-time, allowing the use of batching to collect data over a period of time before processing the full batch.

For example, using an FPGA frequency of 50MHz and using a simple peak detection to isolate heart beats, an average heart rate of 120 would require a processing time of 358ms every minute. Adding the FPGA's activation time of 100ms still allows the device to process all captured data within half a second, allowing it to sleep for more than 99% of the time.

1) Time Breakdown: Since the data is originally sampled on the MCU, it needs to be transferred via the external memory interface to the FPGA [11]. For improved performance, this is done as soon the first convolutional layer is processed – since that is the only layer that directly uses the inputs.

This pipelined procedure can be seen in the oscilloscope output in Figure 7, which shows the active time of the MCU and FPGA for one sample of processed data. At $t = 0$, the CNN is started and it calculates the first convolutional layer. At this point ($t = 0.36ms$) the next sample's data can be written while the computation continues with the second layer. Once this is complete ($t = 1.02ms$), we simply wait for the network to finish ($t = 2.93ms$) so we can retrieve the results.

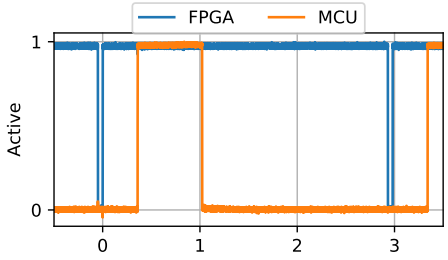

Fig. 7. Time breakdown for single sample

Doing this allows us to better utilise the FPGA while it is active, increasing the total throughput. Instead of waiting $0.66ms$ for the MCU to transfer the data to the FPGA, the device can continuously run samples one after the other.

VI. Conclusion & Outlook

Allowing embedded devices to locally process incoming data allows for very interesting pervasive IoT systems. It provides the ability to increase the complexity of the processing done on this data without resorting to constantly uploading all the raw data to the cloud. By updating the configuration of the FPGA during the deployed time of the system, improved knowledge and increased data collected can be leveraged.

We have presented our solution for such an IoT stack and an accompanying embedded device, which is capable of accurately computing Convolutional Neural Networks without relying on cloud computing. It achieves the same 97% accuracy as can be achieved using a full floating point Tensorflow-based model, but using under 200mW total during active computation. This provides a very energy efficient solution that can still process 335 heart beats per second, allowing the use of duty cycles and batching for longer battery life.

In the future we plan to automate the process of creating such a system by developing a tool that can optimise the variables mentioned automatically based on an energy or accuracy target, and select an appropriate FPGA. Additional neural networks such as Recurrent Neural Networks should also be included in this tool, allowing the user to address a greater variety of target applications.

References

[1] Ke, Hengjin, D. Chen, T. Shah, X. Liu, Xi. Zhang, L. Zhang, and X. Li, "Cloudaided online EEG classification system for brain healthcare: A case study of depression evaluation with a lightweight CNN", Software: Practice and Experience, 2018.

[2] Lee, Jangwon, J. Wang, D. Crandall, S. abanovi, and Ge. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles", In 2017 First IEEE International Conference on Robotic Computing (IRC), pp. 36-43. IEEE, 2017.

[3] Bettoni, Marco, G. Urgese, Y. Kobayashi, E. Macii, and A. Acquaviva, "A convolutional neural network fully implemented on FPGA for embedded platforms", In 2017 New Generation of CAS (NGCAS), pp. 49-52. IEEE, 2017.

[4] McDanel, Bradley, S. Teerapittayanon, and H. T. Kung, "Embedded binarized neural networks", 2017.

[5] Rastegari, Mohammad, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks", In European Conference on Computer Vision, pp. 525-542. Springer, Cham, 2016.

[6] Wang, Erwei, J. J. Davis, P. Y. Cheung, and G. A. Constantinides, "LUTNet: Rethinking Inference in FPGA Soft Logic", In 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 26-34. IEEE, 2019.

[7] Finnerty, Ambrose, and H. Ratigner, "Reduce Power and Cost by Converting from Floating Point to Fixed Point", 2017.

[8] Lin, Darryl, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks", International Conference on Machine Learning, 2016.

[9] Anwar, Sajid, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition", 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015.

[10] A. Burger and G. Schiele, "Demo Abstract: Deep Learning on an Elastic Node for the Internet of Things," International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), IEEE, 2018.

[11] G. Schiele, A. Burger and C. Cichiwskyj, "The Elastic Node: An Experimentation Platform for Hardware Accelerator Research in the Internet of Things," International Conference on Autonomic Computing (ICAC), IEEE, 2019.

[12] Moody, G. B., & Mark, R. G. (2001). "The impact of the MIT-BIH arrhythmia database. IEEE Engineering in Medicine and Biology Magazine", IEEE Engineering in Medicine and Biology Magazine, 2001

[13] Martín Abadi et al. Yuan Yu, and Xiaoqiang Zheng. (2015) "TensorFlow: Large-scale machine learning on heterogeneous systems", Software available from tensorflow.org.