3.1

AI-Powered Collision Avoidance Safety System for Industrial Woodworking Machinery

Francesco Conti[1], Fabrizio Indirli[2], Antonio Latella[3], Francesco Papariello[4], Giacomo Michele Puglia[5], Felice Tecce[5], Giulio Urlini[4] and Marcello Zanghieri[1]

[1]Alma Mater Studiorum – Università di Bologna, Italy
[2]Politecnico di Milano, Italy
[3]SCM Group, Italy
[4]STMicroelectronics, Italy
[5]DPControl, Italy

Abstract

Applying Artificial Intelligence technology to safety-critical industrial equipment requires preliminarily studies on the efficacy and limitations of such technology, to enable the definition of normative certification frameworks. In this chapter, we present the prototype of an ultrasound-based collision avoidance system for industrial woodworking machinery. Using a single ultrasound sensor, the prototype can identify the presence of an operator in less than 13 milliseconds, with high sensitivity (97.3%) and specificity (98.6%) also in the presence of noise. The solution presented is able to leverage increasing amount of data over time to increase accuracy, improving the model while always keeping the inference adequate for the memory, power and latency constraints of real-time execution on an embedded microcontroller unit.

Keywords: electro-sensitive protective equipment, ultrasound processing, time series analysis, machine learning, deep learning, deep neural networks, temporal convolutional networks, safety-critical, embedded systems, TinyML.

3.1.1 Introduction and Background

Every day, industry workers operate on complex pieces of machinery for tasks ranging from woodworking, car construction, circuit soldering, clothing fabrication, etc. Virtually all such machinery requires trained and skilful operation, and it is often hazardous if operated out of well-defined practices. Safeguarding the health of operators without disrupting operativity is therefore a paramount concern in the design of such pieces of machinery, prompting the specification of industrial standards for functional safety of equipment: safeguards that halt the operation of machinery and bring it to a safe state when certain conditions are met, e.g., a person is detected in the trajectory of a moving part by a non-contact Electro-Sensitive Protective Equipment (ESPE) sensor, such as a photodiode.

To ensure a shared set of rules that can be used as a normative framework, safety equipment in industrial machinery must typically be certified under international standards such as IEC 61508 and EN/IEC 61496 to achieve a given Safety Integrity Level (SIL). The former standard deals in general with all programmable electronic equipment, while the latter is more specific on requirements for designing, building, and verifying the operation of non-contact ESPE systems. A downside of international standards is that newly introduced technology that is not covered by the current version cannot be certified – and this includes innovative algorithms. Even technology improving the worker's safety or convenience must wait for inclusion in a new version of the standards, which only happens if the technology itself is mature enough to trigger the industry's interest in its inclusion.

In the fields of machine vision and data analytics, algorithms based on artificial intelligence – and particularly, Deep Learning (DL) – have recently become mainstream, both when executed in the cloud and directly *at the edge*, i.e., on the same computing devices that perform data collection. The most famous family of algorithms, Deep Neural Networks (DNNs), is now considered a mature technology, showing extremely good results for many data analytics tasks. DNNs would be a promising technology for improving the performance of safety equipment such as ESPE systems by extracting more relevant information out of sensor data, for example from the correlation

between multiple streams of information. Edge Deep Learning techniques enable an analysis such as this to be performed directly near the sensors themselves, ensuring high reliability and ultra-low-latency response to critical situations. Despite this promise, applications of AI to ESPE systems are not yet considered by any industrial standard, and industry interest would have to be gathered by means of advanced prototypes demonstrating the effectiveness of this idea in the field.

In this chapter, we showcase a prototype collision avoidance system for industrial woodworking machinery that is based on cheap ultrasound sensors – like the ones used for park assist in cars – coupled with an algorithm based on Temporal Convolutional Networks (TCNs), a sub-family of lightweight DNNs specifically dedicated to time series data analytics. The goal of the collision avoidance system is to detect the presence of a human body in front of the sensor array using only ultrasound information, possibly in presence of intrusive noise coming from other operations in a busy manufacturing environment.

3.1.2 Review of Industrial-level Methods for Edge DNNs

In recent years, DNNs became leading solutions in a broad variety of computational tasks, and they are being extensively integrated into digital industries. The rapid proliferation of pervasive Internet of Things (IoT) devices and of ubiquitous cognitive computing is pushing the industry towards performing Machine Learning (ML) inference on edge devices, which enables real-time processing of data and reduces strain on Cloud networks. These embedded platforms, however, pose stringent constraints on power consumption, latency, and memory footprint. Moreover, some devices are equipped with hardware accelerators that require specialized programming and mapping endeavours 1D be fully exploited These new challenges led to the development of Tiny Machine Learning (TinyML), a novel and rapidly growing field that aims to enable 1he porting of ML algorithms onto embedded computational platforms, characterised by strict requirements in terms of memory and power envelope.

3.1.2.1 Compression Techniques

In the modelling and training stages, several compression techniques can be used to trim the size of the network and reduce the number of computations.

Quantization approximates real (floating-point) values to integers with lower bit widths, enabling reduced-precision computation. In fact, normally most networks are trained using FP32 numbers, but smaller representations can greatly optimize the memory utilization and the inference performance with little loss of accuracy [1]. Model designers can experiment with different numerical precisions for the weights and/or the activations of each layer, building mixed-precision models to achieve the desired trade-off between prediction loss and model size. Although quantization is more effective when applied during training (*quantization-aware training*), also *post-training quantization* schemes are widely used.

Vector compression schemes focus on reducing the constants' size by clustering and sharing the weights and the biases, using algorithms such as k-means or hash functions.

Pruning removes redundant parameters or neurons that do not significantly contribute to the accuracy of results, for example, because they are 0. It can be performed at training time (*static pruning*) or at runtime (*dynamic pruning*) and it can either target the *neurons* or the *connections* between them.

3.1.2.2 Popular Frameworks and Tools

Most of the major deep learning frameworks support some compression schemes and other techniques tailored for Deep Learning at the Edge [2].

TensorFlow Lite is a lightweight framework for on-device inference, based on the popular TensorFlow (by Google). It supports post-training quantization targeting half-precision float (FP16) and INT8 datatypes. Moreover, *quantization-aware training* and pruning can be performed in TensorFlow and Keras. The models produced with these tools can be used also in **TensorFlow Lite Micro**, a runtime framework designed to perform inference on microcontrollers.

Apache TVM is an open compiler stack that provides end-to-end compilation of neural networks (modelled in TensorFlow, ONNX, Keras, or MXNet) to several backend frameworks and hardware targets. It supports quantization up to $1\sim4$ bits, as well as block-sparsity. Moreover, the **microTVM** extension allows targeting small bare-metal devices using a minimal C runtime.

ONNX provides an open format for AI models, aiming to simplify networks exchange between different frameworks, tools, and target hardware. It supports INT8 quantization, both at runtime and during training, for Convolutional, MatMul, and Activation layers.

STM32CubeAI is a software extension pack for the STM32CubeMX code-generation tool, which provides a user-friendly GUI to quickly configure STM32 microcontrollers to run Neural Networks inference. It supports ONNX and TFlite models and can perform post-training compression on them. The generated code provides APIs to use multiple models in the same codebase and to accelerate their execution using ARM CMSIS kernels.

In addition to the popular frameworks listed above, **novel specialized tools** implement more advanced compression techniques and finer-grained quantization settings: some examples are QKeras [3], Larq [4], and Brevitas [5].

3.1.3 Materials and Methods

3.1.3.1 System Architecture

The collision avoidance system that we propose exploits ultrasonic (US) sensing to detect the presence of a person or object within a certain distance from the industrial machine; in case of detection, a STOP signal is immediately conveyed to the machine control logic. The system works in a conceptually simple way: a set of transducers emit an ultrasonic pulse; if the pulse hits a person, it will produce an echo that can be sensed by the transducers themselves.

To work correctly, the system must operate with ultra-low latency, and, at the same time, it has to deal with many possible noise sources that pollute the signal and make it harder to achieve high sensitivity and specificity. To increase the system's resiliency against interfering waves in the US spectrum, we process the acquired data using a Neural Network to discern the US echo from other unwanted noises and detect the presence of a worker in the machine's trajectory.

As shown in the Figure 3.1.1, the main components of this system are:

- The ultrasonic sensors and their drivers
- A Lattice FPGA
- An STM32-H7 microcontroller board

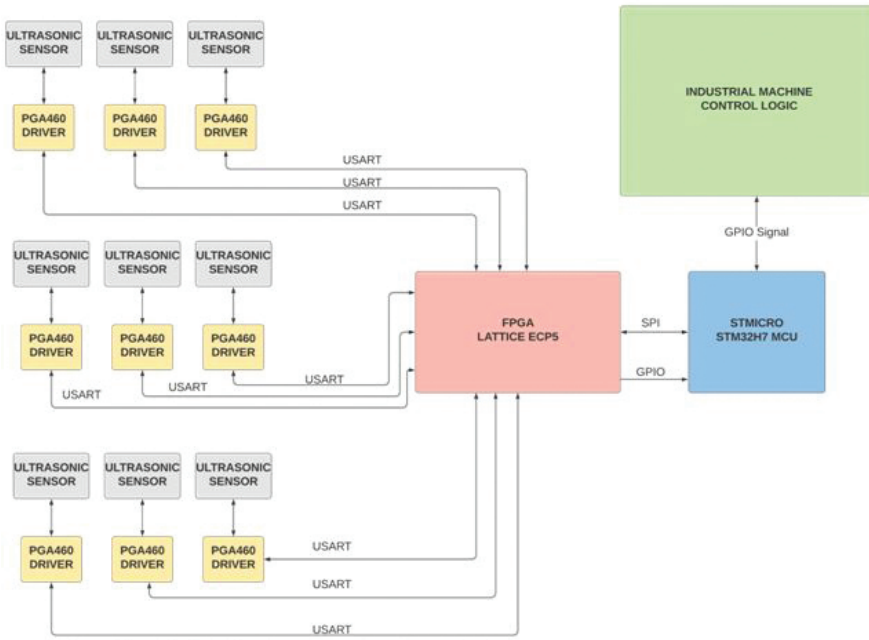

Figure 3.1.1 System architecture schema.

The data acquisition pipeline of the final system will start with a 3×3 matrix of 9 MULTICOMP MCUSD14A58S9RS-30C **ultrasonic ceramic transducers**, while the demo prototype uses only one sensor. Each of these devices acts both as an emitter and as a microphone for sound waves with a frequency around 50 kHz. When the emitted ultrasonic pulses encounter an obstacle, they get reflected towards the sensor, which translates the mechanical vibrations of the echo into a variable electrical tension. By analysing this signal, the distance between the sensor and an object, if any, can be easily derived.

Each sensor is driven by a dedicated Texas Instruments PGA460 **ultrasonic signal processor and driver**, that integrates a low-noise amplifier, a programmable time-varying gain stage, an (up to) 12-bit ADC and a DSP. In the proposed system, the ADC is configured with a resolution of 8 bits that matches the input bit width of the Neural Network. The sampling frequency is 1 MHz and the sampling period for each data chunk is 20.48 ms, yielding an output data rate of 8000 kbps for each driver.

A low-power Lattice ECP5 LFE5U-85F **FPGA** is used to aggregate the data streams (channels) coming from each of the PGA460 devices; moreover,

at start up the FPGA configures the drivers' resolution, sampling rate, and other parameters. The devices communicate using the USART protocol, in which the programmable logic acts as master, while the ultrasonic drivers are slaves. In particular, the communication happens through synchronous USART (clocked at 8 MHz) with a packet size of 8 bits and a baud rate of 8 Mbps.

After collecting all the data of a sampling window from the drivers, the FPGA performs subsampling, reducing the 20480 per-channel samples by a $10\times$ factor; then, all the channels' data is packed to be sent to the STM32 microcontroller. A data package produced by the FPGA contains 2048 8-bit samples for each channel, for a total size of 147.46 Kb. The communication between the two devices happens mainly via the SPI protocol (Mode 0), with a serial clock of 5 MHz, and is initiated by the MCU which acts as the master. An additional GPIO line is asserted by the FPGA to inform the microcontroller when it has finished its tasks.

The STMicroelectronics **STM32H743ZI2 board** (STM32-H7 for short), belonging to the Nucleo-144 family, features a high-performance ARM Cortex-M7 processor (with double-precision FPU) operating at 480 MHz, 2 MB of Flash memory, 1 MB of SRAM (including 192 KB of tightly-coupled scratchpad memory for real-time tasks), 4 DMA controllers, and several communication peripherals including UART/USART, SPI, USB-OTG, Ethernet, and GPIO lines.

Its main duties are to acquire the data from the Lattice device, perform the Neural Network's inference and send control signals to the industrial machine's PLC and to the FPGA (which, in turn, controls the ultrasonic drivers and transducers).

In particular, the MCU prototype firmware implements a state machine (depicted in the Figure 3.1.2) composed of five main states:

(1) *Waiting FPGA configuration*
(2) *Waiting Acquisition*
(3) *Data Transfer*
(4) *Inference*
(5) *Halted*

The system starts in state (1), in which the FPGA configures the transducers drivers and communicates with the MCU. When ready, the FPGA sends a signal to the MCU, which replies with a "*Start Data Acquisition*" command and waits in the state (2). When all the data needed for one inference have been acquired and pre-processed, the FPGA sends a "*Data_ready"* signal

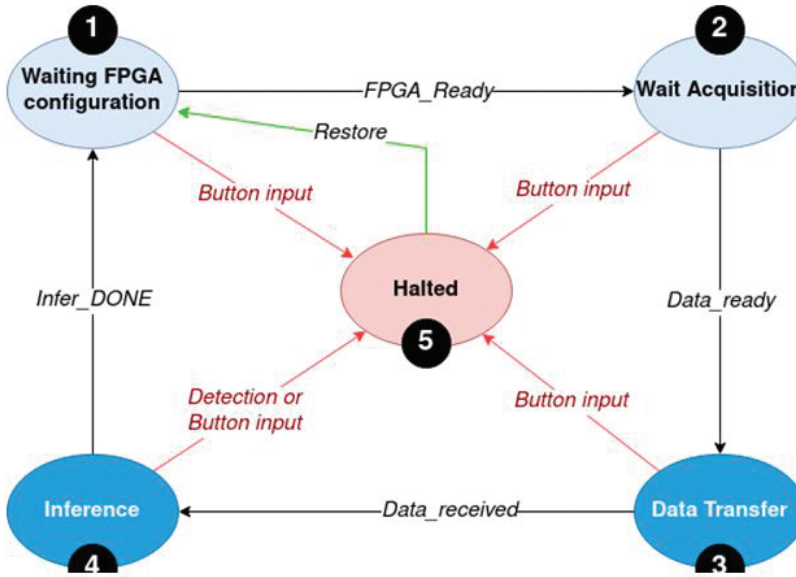

Figure 3.1.2 MCU firmware state machine.

which triggers the data transfer to the STM32 over SPI, happening in state (3). When the payload is received, the MCU informs the FPGA (which will prepare for the next acquisition) and then it runs the Neural Network inference in state (4). If an obstacle is detected, the MCU will assert a GPIO line connected to the industrial machine's controller, and the firmware will move to state (5); otherwise, the system will go back to state (1). The system can be halted also by pressing a specific button on the control panel, to allow the user to report false negatives: this information is stored and could be used to improve the model accuracy in the future. A dedicated command can be used to exit from the halting state and restore the normal execution from state (1).

3.1.3.2 Dataset Collection

To address the targeted use case, an ultrasound (US) dataset was collected with an acquisition setup reproducing the working conditions of the industrial machinery of interest.

Framing the problem as a binary classification task, where the goal is detecting the presence of a human inside the area of interest, US windows were recorded with and without a person originating a US echo.

Working in an anechoic chamber, bursts of US at 50kHz were sent toward a predefined region by utilizing the TI PGA460 illustrated in the previous section. Ultrasounds were sampled at 1MHz, producing UINT8 data; and a window of 28ms was recorded in each acquisition.

In addition to the presence or absence of a person, corresponding to the positive and negative class respectively, two conditions were varied to explore the variability of the real working conditions:

- The person's distance from the sensor, which was varied from 0.5m to 2.0m.
- The pressure level of a compressed-air jet used to reproduce the environmental noise of the machinery's room, which was varied from 0bar to 3bar (applied for the negative class as well).

A total of 227 US windows (85 negatives, 142 positives) were collected with different combinations of the described settings.

Four examples of the acquired US signals are shown in the Figure 3.1.3 (for clarity, only the first 20ms are displayed). All windows include the final part of the US burst, which contains no information but can be easily cut since it has the same timing in every recording. As it can be seen, the information needed for classification is strictly related to the reception of a US echo.

The shown recordings exemplify the motivation for addressing the task with Machine Learning (ML), and with Temporal Convolutional Networks (TCN) in particular. In the ML/DL paradigm, the modelling of the relationship between the signal pattern and the class is completely entrusted to the training of the algorithm, which learns a discriminant function that is entirely *data-driven*, i.e., fully independent from any feature extraction handcrafted ad hoc. Thus, there is no need to develop an analytical description of the positive echo, which would require a huge amount of trial and error. Moreover, TCNs (defined in detail in the next Subsection, *Detection Methods*) are deep models able to work directly on raw signals. In our use case, this prerogative is paramount, since it allows to automatize the discrimination between the signal of interest and the background noise due to the compressed-air jet pressure, making this aspect data-driven as well and avoiding the need for any manual calibration.

For the subsequent Deep Learning setup, data were kept UINT8, and under-sampled to 100kHz and randomly split into a training set and a test set with a 50%-50% proportion. The US burst was discarded from all recordings,

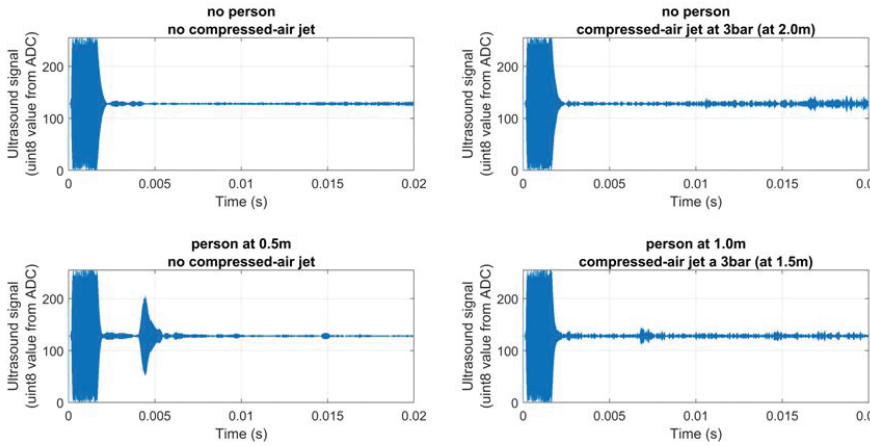

Figure 3.1.3 Visualizations of selected samples of the dataset.

and only 2048 samples (20.48ms at 100kHz) were considered for each window.

3.1.3.3 Detection Methods

Temporal Convolutional Networks (TCNs) are a recent category of deep models that have surged to State of The Art (SoA) in many tasks involving time sequences, surpassing Recurrent Neural Networks (RNNs) for trainability and accuracy [6–8].

The two features of TCNs are in the 1D convolutions, applied along the temporal direction:

- *Causality*: filters only cover a left-neighbourhood of each input sample, to exclude future samples.
- Dilation: a fixed step d is inserted between the kernel's input samples, to enlarge the receptive field while keeping the model size fixed.

The TCN presented in this work is inspired by TEMPONet, a topology that is SoA in real-time classification and regression of bio signals [9]. In this work, we impose a reduction of model size compared to the original TEMPONet structure: we strongly reduce the feature maps (as explained below), obtaining a net that is more compact as to both parameters and activations. Moreover, we use no activation (i.e., we set $d = 1$ for all convolutions), since we experimentally found that dilation provides no increase in accuracy.

The model architecture is made up of 3 Convolutional Blocks, each containing:

- 2 causal convolutions with filter size $k = 3$ and full padding;
- 1 convolution with filter size $k = 5$, variable stride s, followed by average pooling (filter 2, stride 2).

The 3 convolutional blocks have stride $s = 1, 2, 4$, respectively. Searching for a net as compact as possible, we lower the number of feature maps of the 3 convolutional blocks compared to the original model of [9]. We reduce Block I's maps from 32 to 2, reduce Block II's maps from 64 to 4, and reduce Block III's maps from 128 to 4.

After the convolutional blocks, 3 Fully Connected (FC) layers execute the classification. FC I has 8 units, FC II has 4 units, and FC III has 1 unit, representing the estimated probability of the input sequence belonging to the positive class.

All layers, except FC III, have ReLU non-linearity as activation function and are equipped with Batch-Normalization to counter internal covariate shift [10]. FC I and FC II are trained with dropout with $p_{drop} = 0.5$, to help regularization [11].

The optimized TEMPONet proposed in this work processes a 2048-sample input US window (20.48ms at 100kHz) with less than 1000 parameters. The model directly works on raw signals (UINT8 data), without the need for any pre-processing or feature extraction, which would cause overhead before inference.

The TCN model was implemented in Python 3.8 using PyTorch 1.6. Trainings were performed with Binary Cross-Entropy loss (computed in a class-balanced way, i.e., weighting equally the positive and negative class), and Stochastic Gradient Descent (SGD) with AdaM optimizer, initial learning rate $\lambda_0 = 0.001$, and minibatch size 128. SGD was applied for 8 epochs at FP32 precision, followed by Post-Training Quantization (PTQ) to 8bit (i.e., INT8). Then, 8 epochs of quantization-aware training were performed, applying PArameterized Clipping acTivation (PACT) [12] as implemented in the library NEMO (NEural Minimizer for tOrch [13], [14], an open-source package to quantize CNN for deployment on memory-constrained ultra-low power platforms, such as the STM32-H7 MCU targeted in this work. These further 8 epochs of quantization-aware training are a fine-tuning which mitigates the initial drop of accuracy due to PQT (as shown in the Section Results).

On the edge device, different variants of this TCN were tested (for example, with or without dilations), but all the presented experiments are based on the TCN without dilations.

Since an STM32 microcontroller is being used, the models are fed to the X-CUBE-AI extension (version 6.0) for STM32CubeMX, which configures the MCU and generates the code of the inference application. The generated API contains functions to manage multiple models and to run the inference on the data of an input buffer.

Three versions of the chosen TCN were produced, tested, and compared:

- The original model *(TCN FP32),* using FP32 inputs and no compression or quantization: it consists of 251224 MACC operations, 3.57 KiB of weights, and 24 KiB of activations.
- The compressed model *(TCN FP32 compressed)*, using FP32 inputs and X-CUBE-AI's weight-sharing compression based on *k*-means clustering (with compression level = 8): it consists of 251224 MACC operations, 1.84 KiB of weights, and 24 KiB of activations.
- The quantized model *(TCN UINT8 Quantized)*, which uses UINT8 inputs, outputs, activations, and weights: it consists of 226509 MACC operations, 1.02 KiB of weights and 6.05 KiB of activations.

3.1.3.4 Continual Learning Setup

To simulate a realistic scenario of Continual Learning, where the new data received by the TCN are also stored and used for periodic retraining, we conducted experiments applying an increasing data augmentation to the original US data. Data augmentation allows increasing the variability of the data seen by a model during training, thus improving its generalization capability. Exploring data augmentation provides insight into the model's ability to leverage an increasing training set, improving its learning over time.

In our experiments, data were augmented by a factor F_{augm} ranging from 50 to 1000. From each original US data window, F_{augm} synthetic windows were produced via a two-step transformation:

- Scaling by a random factor s uniformly drawn between 0.5 and 1.5, followed by clipping and rounding to recover UINT8 values.
- Time-shift by a random interval Δt uniformly drawn between -1.5ms and +1.5ms.

Since this augmentation injects randomness from the very beginning of our training pipeline, $N_{\mathrm{rep}} = 30$ repetitions of the experiment were performed for

each explored value of F_{augm}; each repetition involved augmenting the data from scratch and training a TCN from scratch.

3.1.4 Experimental Results

3.1.4.1 Evaluation Metrics

In the setup we propose, we target both *classification metrics*, regarding the goodness of the recognition provided by the algorithm, and *deployment metrics*, to assess the suitability of the model and of the chosen STM32-H7 MCU to the edge-inference workload.

The classification metrics we target are three, and are typical of binary classification in unbalanced settings:

- *Sensitivity* (also called *True Positive Rate* - TPR, or *recall*): the fraction of actual positives the net correctly classifies, formally TP / (TP + FN);
- *Specificity* (or *True Negative Rate* - TNR): the fraction of actual negatives the net correctly classifies: TN / (TN + FP);
- *Balanced accuracy* (or *macro-average accuracy*): the average between sensitivity and specificity.

In contrast with unbalanced accuracy, these three metrics are independent of class imbalance. Moreover, recourse to sensitivity and specificity allows characterizing the model's behaviour exploring different sensitivity-specificity tradeoffs, which is methodologically interesting as it allows to tune the model's detection threshold based on the application-specific relative importance of False Positives and False Negatives.

The deployment metrics we use are three, to assess the satisfiability of the main requirements of real-time on-edge computations:

Memory footprint: amount (and percentage of the available quantity) of RAM and FLASH memory used by the firmware (including the neural network); since dynamic memory is not used, this metric is available at compile-time.

- *Latency:* amount of time or clock cycles needed to complete one inference, once the input data is ready.
- *Power consumption:* average power draw of the MCU during a sequence of inferences.

The *memory footprint* metric is particularly important in a resource-constrained scenario and provides interesting insights on which alternative devices could be used for the computation. In terms of performance

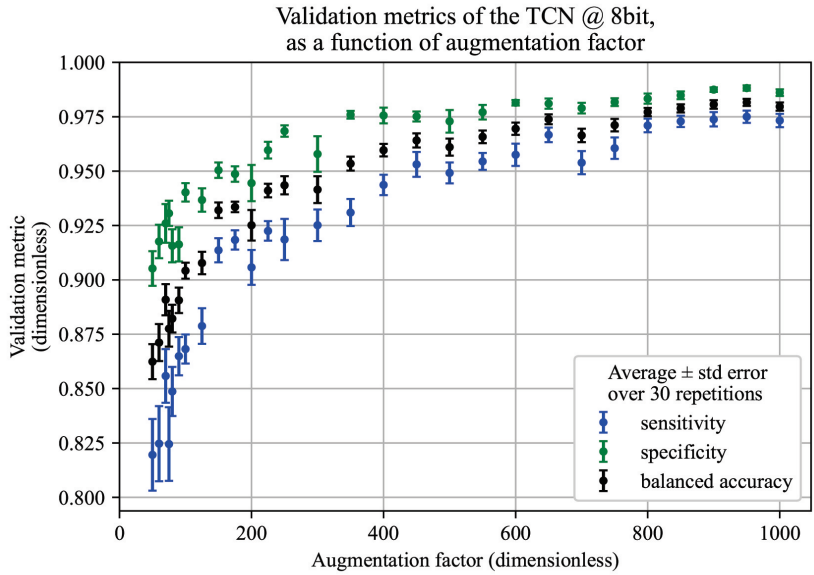

Figure 3.1.4 Validation metrics at different augmentation factors.

assessment, *throughput* was not considered since this system is mainly sequential and does not exploit parallelism or pipelining.

3.1.4.2 Continual Learning Scenario

Figure 3.1.4 shows the validation metrics of the proposed TCN quantized to 8bit, after training on data augmented by a factor F_{augm} ranging from 50 to 1000. The observed trend is a clear learning curve, which highlights that our setup is able to leverage an increasing amount of data to improve the recognition.

The same behaviour is shown by a further characterization, namely the Receiver Operating Characteristic (ROC) curve for the validation metrics, reported in the Figure 3.1.5. As it can be seen, the model is able to exploit the larger training set, produced by stronger augmentation, to improve its sensitivity-specificity Pareto frontier.

Our setup is thus well-suited for a Continual Learning scenario, where the classifier is required to improve its goodness by exploiting new data seen in periodic re-training.

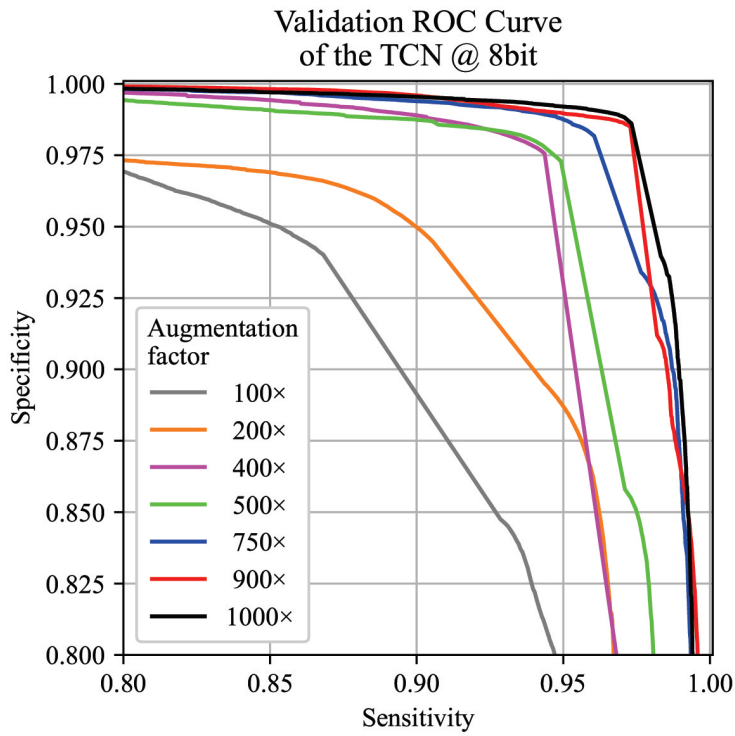

Figure 3.1.5 ROC curves at different augmentation factors.

3.1.4.3 Robustness Against Quantization

Table 3.1.1 reports the validation metrics at different stages of training, obtained for augmentation factor $F_{\text{augm}} = 1000$ and averaged over the $N_{\text{rep}} = 30$ repetitions run. Each validation is performed as an inference over the test set. The best recognition is achieved by the model in FP32 format. However, this numeric type is not hardware-friendly and is not suitable for deployment. After Post-Training Quantization to 8bit and 1 epoch of quantization-aware training, the model's goodness at validation drops by 1.75%, 0.74%, and 1.25% in sensitivity, specificity, and balanced accuracy, respectively. After 8 epochs of quantization-aware training, these degradations (at validation) are reduced to 1.02%, 0.14%, and 0.58% respectively.

This demonstrates that 8bit fine-tuning is capable to recover the accuracy drop following quantization. The final negligible deterioration proves that the proposed TCN is robust against quantization to 8bit.

Table 3.1.1 Validation metrics at different training stages.

Training level	Validation metric		
	Sensitivity	**Specificity**	**Balanced accuracy**
8 ep. FP32	0.9835 (baseline)	0.9875 (baseline)	0.9855 (baseline)
8 ep. FP32 + 1 epoch INT8	0.9660 (- 0.0175)	0.9801 (- 0.0074)	0.9730 (- 0.0125)
8 ep. FP32 + 8 epoch INT8	0.9733 (- 0.0102)	0.9861 (- 0.0014)	0.9797 (- 0.0058)

3.1.4.4 Latency, Energy and Memory Footprint on STM32H743ZI

The three TCN variants described in Section 3.4.1 were deployed and tested on the STM32 MCU using the X-CUBE-AI platform. To perform the measurements, the framework's "System Performance" application template was used, and the board was connected to a PC through its microUSB port, which served also as power source.

Table 3.1.2 reports the measured values of the chosen deployment metrics.

For each model, the execution latency is the average value over 16 inferences with random input data.

The memory footprint was computed statically by the compiler, based on the size of the program's segments and their location according to the linker script: in these experiments, the *.text* segment was allocated in the Flash memory, while *.data*, *.bss* and *.stack* were saved in the RAM. The values of RAM and Flash utilization reported in the Table 1.1.2 refer to the models' data structures, as reported by X-CUBE-AI.

The power consumption was measured using a USB power meter connected between the board and the computer. For each model, we report the average power draw over a 30-seconds interval, during which a continuous cycle of inferences was being executed on the board.

Despite the baseline model (TCN FP32) already meeting the selected requirements, the results show that compression and quantization techniques can successfully reduce the memory footprint and the power consumption on this workload. However, limitations in the current version of the framework prevented latency optimizations. Our future work will focus on optimizing

Table 3.1.2 Deployment metrics on three model variants.

Model version	**Latency**	**RAM Util.**	**Flash Util.**	**Power Draw**
TCN FP32	11.412 ms	32 KiB	3.57 KiB	1.639 W
TCN FP32 Compressed	11.418 ms	32 KiB	1.84 KiB	1.637 W
TCN UINT8 Quantized	12.768 ms	8.11 KiB	1.02 KiB	1.621 W

the inference time on top of the platform, to combine the quantized model's higher memory efficiency with performance gain.

3.1.5 Conclusion

In this work, an ultrasound-based and AI-powered collision avoidance system for industrial machinery was presented. Its development required engineering of the hardware acquisition and processing pipeline in terms of hardware components, system integration, and firmware development, as well as dataset collection, models optimization and deployment. The resulting 1-sensor prototype is able to accurately sense the presence of a person in the working area with low latency.

Acknowledgements

This work is conducted under the framework of the ECSEL AI4DI "Artificial Intelligence for Digitising Industry" project. The project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826060. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Germany, Austria, Czech Republic, Italy, Latvia, Belgium, Lithuania, France, Greece, Finland, Norway.

References

[1] T. Liang, J. Glossner, L. Wang, and S. Shi, "Pruning and Quantization for Deep Neural Network Acceleration: A Survey," Jan. 2021, Accessed: Jun. 07, 2021. [Online]. Available: http://arxiv.org/abs/2101.09671.

[2] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine Learning at the Network Edge: A Survey," Jul. 2019, Accessed: May 31, 2021. [Online]. Available: http://arxiv.org/abs/1908.00080.

[3] C. N. Coelho et al., "Automatic deep heterogeneous quantization of Deep Neural Networks for ultra low-area, low-latency inference on the edge at particle colliders," Jun. 2020, Accessed: Jun. 17, 2021. [Online]. Available: http://arxiv.org/abs/2006.10159.

[4] T. Bannink et al., "Larq Compute Engine: Design, Benchmark, and Deploy State-of-the-Art Binarized Neural Networks," Nov. 2020, Accessed: Jun. 17, 2021. [Online]. Available: http://arxiv.org/abs/2011.09398.

[5] Alessandro Pappalardo, "Xilinx Brevitas." Zenodo, doi: 10.5281/zenodo.3333552.

[6] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal Convolutional Networks for Action Segmentation and Detection," Jul. 2017, doi: 10.1109/CVPR.2017.113.

[7] S. Bai, J. Z. Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," Mar. 2018, Accessed: Jun. 17, 2021. [Online]. Available: http://arxiv.org/abs/1803.01271.

[8] J. L. Betthauser, J. T. Krall, R. R. Kaliki, M. S. Fifer, and N. V. Thakor, "Stable Electromyographic Sequence Prediction during Movement Transitions using Temporal Convolutional Networks," in International IEEE/EMBS Conference on Neural Engineering, NER, May 2019, vol. 2019-March, pp. 1046–1049, doi: 10.1109/NER.2019.8717169.

[9] M. Zanghieri, S. Benatti, A. Burrello, V. Kartsch, F. Conti, and L. Benini, "Robust Real-Time Embedded EMG Recognition Framework Using Temporal Convolutional Networks on a Multicore IoT Processor," IEEE Trans. Biomed. Circuits Syst., vol. 14, no. 2, pp. 244–256, Apr. 2020, doi: 10.1109/TBCAS.2019.2959160.

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in 32nd International Conference on Machine Learning, ICML 2015, Feb. 2015, vol. 1, pp. 448–456, Accessed: Jun. 17, 2021. [Online]. Available: https://arxiv.org/abs/1502.03167v3.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," 2014. Accessed: Jun. 17, 2021. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[12] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: PARAMETERIZED CLIPPING ACTIVATION FOR QUANTIZED NEURAL NETWORKS," 2018.

[13] "GitHub - pulp-platform/nemo: NEural Minimizer for pytOrch." https://github.com/pulp-platform/nemo (accessed Jun. 17, 2021).

[14] F. Conti, "Technical Report: NEMO DNN Quantization for Deployment Model," Apr. 2020, Accessed: Jun. 17, 2021. [Online]. Available: http://arxiv.org/abs/2004.05930.