A Novel Diagnosis Concept for Verification of Neural Networks on the Target Hardware

Ilkay Wunderlich
Technical University of Dresden
Institute of Computer Engineering
Dresden, Germany
email: ilkay.wunderlich@tu-dresden.de

Jens Braunes
PLS Programmierbare Logik & Systeme GmbH
Lauta, Germany
email: jens.braunes@pls-mc.com

Rainer G. Spallek
Technical University of Dresden
Institute of Computer Engineering
Dresden, Germany
email: rainer.spallek@tu-dresden.de

***Abstract*—Neural networks are increasingly being used in embedded applications. It is important to check whether the trained network fulfills its tasks not only in simulations, but also on real target hardware. This is of particular importance in safety-critical applications such as plant control or autonomous driving. For this reason, a diagnostic concept for Artificial Intelligence (AI)-based systems based on the Universal Debug Engine from PLS Programmierbare Logik & Systeme GmbH was developed at the Technical University of Dresden. This concept enables developers to verify the hardware implementation of the neural network. After the concept was developed, it was successfully tested on several example applications of which two are presented in the paper.**

***Index Terms*—neural networks, hardware, diagnosis, debugging.**

I. Introduction

Methods from the field of Artificial Intelligence (AI) are still advancing and have also produced impressive applications in connection with the Covid-19 pandemic. For example, one application of these methods, developed by researchers at the University of Central Florida, is able to use computed tomography images to identify whether COVID-19 pulmonary pneumonia is present [1]. Studies show that this AI based algorithm, with a detection rate of 90%, has similar accuracy as doctors who specialize in lung disease. However, in the area of health care, the AI methods typically run on systems with large computational power and are supervised by an expert, such as a doctor.

But in addition to this impressive use case, AI applications are also being used more and more frequently in the embedded area. Especially in today's automotive systems, techniques from the discipline of AI are increasingly being used. In this field, neural networks are a tool that should be highlighted, which are particularly well suited for detection and classification tasks in the field of computer vision. One common applications for networks is the fusion of sensor data as described by X. Zeng et al. in [2] with focus on sensor data in automotive systems. Another use case is the detection and classification of traffic participants for visual environment recognition. For this use case, plenty of public traffic data sets like the Berkeley Deep Drive Data Set [3] or City Scapes Data Set [4] exist. In such safety critical applications which mostly run without any supervision, it is of great importance to verify and test the implementation on the target hardware.

In [5] by J. Zhang et al., a large review of testing and verification methods for neural networks is given. However, the majority of the presented publications aim on software verification without the inclusion of the target hardware.

A more hardware focused method, which tests the hardware implementation of the neural network, is described by S. Huang et al. [7]. As described in the publication title, each layer is individually checked for correctness using an extra transition module for a specific Field Programmable Gate Array (FPGA)-based hardware accelerator.

In this research, the focus is set on a diagnosis concept, which allows developers to verify the neuronal network on a broad range of different embedded hardware solutions. Instead of the implementation of an extra module on the target hardware, the Universal Debug Engine® (UDE) from PLS Programmierbare Logik & Systeme GmbH (PLS) is used [8]. UDE is a debugger solution for various microcontroller families such as AURIX, TriCore, Power Architecture, Cortex, Arm, STM32 and more. The physical interface between the hardware and the host computer with UDE is realized with the PLS Universal Access Device (UAD).

The proposed method aims exclusively at the verification of the neural network hardware. False results of the neural network due to poor training, under-sizing of the neural network, or incomplete case coverage in the training data are not part of the diagnosis. Such issues must be clarified before porting the network on the hardware. For example, in [6] by A. Kirchknopf et al., methods to investigate detection results are given.

The rest of the paper is structured as follows. In Section 2, the "life" is described as an illustration of the different phases of a neural network. Section 3 deals with the possible sources of possible errors on hardware. In Section 4, the diagnosis concept to detect these errors is described theoretically whereas Section 5 demonstrates the diagnosis concept on two hardware examples. Section 6 concludes the work and alludes to additional features as well as the future work.

II. The "Life" of a Neural Network

The "life" of a neural network generally consists of the following three phases:

A. Training Phase

With large amounts of data, all trainable weight and bias parameters of the network are adjusted by gradient-based training algorithms. At the same time, the accuracy of the network is determined using validation data. The individual elements of the training and validation data, consisting of pairs of input values and the associated output values, correspond to the structures of the input and output layers. Due to the high computing intensity, high-performance Graphics Processing Units (GPUs) or special cloud services are usually used to train neural networks. Common frameworks for training neural networks are Tensorflow [9] and Pytorch [10].

The output of the Training Phase is called the Golden Model, which contains structure, trained parameters and other meta information of the neural network. A concrete example for the Golden Model could be a neural network which is trained with Tensorflow and exported as a Tensorflow Keras model file: `network.h5`.

B. Adaptation Phase

After the training phase, neural networks can be accelerated using optimization steps such as batchnorm fusion or pruning. With suitable quantization methods, the arithmetic operations are also transformed from floating point to integer formats. The adjustments reduce the arithmetic complexity and allow neural networks to run on embedded processors with acceptable power consumption and latency. A commonly known tool for optimization and quantization is Tensorflow Lite which is build in Tensorflow [13]. However, there are also many custom methods for adapting neural networks as described by I. Wunderlich, B. Koch and S. Schönfeld in "An Overview of Arithmetic Adaptations for Inference of Convolutional Neural Networks on Re-configurable Hardware" [11].

The result of the Adaptation Phase is called the Silver Model and represents the adapted and quantized version of the Golden Model. In general, there are quantization losses due to the optimization and adaptation steps, but these ideally lead to minor deviations between the outputs of the Golden and Silver Models. Nevertheless, it is advisable to carry out further tests with test data to ensure that the deviations are within an acceptable range.

After successfully generating the Silver Model, it can be ported to the target device. In the example from above, the Silver Model could be a Tensorflow Lite model file: `network.tfl`.

C. Inference Phase

The inference phase describes the use or application of the fully trained and adapted neural network. Typically, unknown data is interpreted and evaluated accordingly.

In the example, the network is running on a STMicroelectronics Micro Controller Unit (MCU). In Figure 1, the general interaction as well as the example between the three phases, the Golden and Silver Model are schematically visualized.

III. Possible Sources of Error

The use of neural networks offers several advantages, especially in safety-critical applications in which large amounts of data must be evaluated and processed within a short period of time. But only if it works as desired on the real hardware. Ultimately, this can only be completely clarified with a detailed hardware diagnosis, because, unfortunately, there is a whole range of potential sources of error. They can essentially be divided into the following categories:

- Conversion Errors:
  When converting in the adaptation phase, incorrect quantization can lead to arithmetic overflows and underflows, thus reduce the quality of the predictions immensely.
- Porting Errors:
  After the adaptation, errors such as exceeding memory limitations, incorrect programming of the interfaces or similar can occur when porting the quantized model.
- Implementation Errors:
  When implementing neural networks, there are many sources of error related to arithmetic, flow control, and data management. With frameworks such as STMicroelectronics' X-CUBE-AI, MCU manufacturers are already providing tested and functional code [12]. However, similar sources of error are conceivable again when adapting or expanding. It is, therefore, essential to ensure that neural networks are correct, especially in the case of safety-critical applications.

In order to enable the fastest, most efficient possible checking and verification of all those factors in the future, a new diagnostic concept for AI-based systems was developed at the Technical University Dresden in cooperation with PLS.

IV. The Diagnosis Concept

The diagnosis concept is realized as a diagnosis loop, which iteratively compares the results of the hardware with the results of the Golden or Silver Models. In Figure 2, the diagnosis loop is schematically shown. The diagnosis concept is implemented as a Python project on a host computer which is connected to the target hardware via the UAD. The individual processes (rectangular blocks in Figure 2) are described below:

A. Data Extraction

In the first step, the input matrix x_{HW} and corresponding output matrix y_{HW} are read via the Python automation interface of the software debugger UDE and hardware tool UAD from PLS.

The index $(.)_{\mathrm{HW}}$ indicates that the matrices come from the hardware. The dimensions of the matrices vary depending on the application. In computer vision tasks, the general input matrix x can have the following dimensions, as shown in (1):

$$\dim(x) = (W, H, C) \tag{1}$$

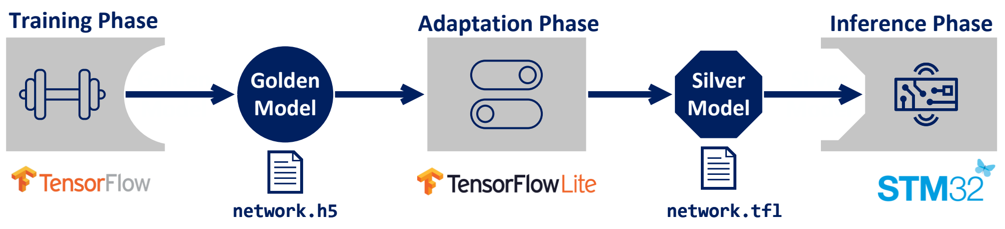


Figure 1. General and exemplary interaction between Training Phase, Adaptation Phase, Inference Phase, Golden Model and Silver Model.

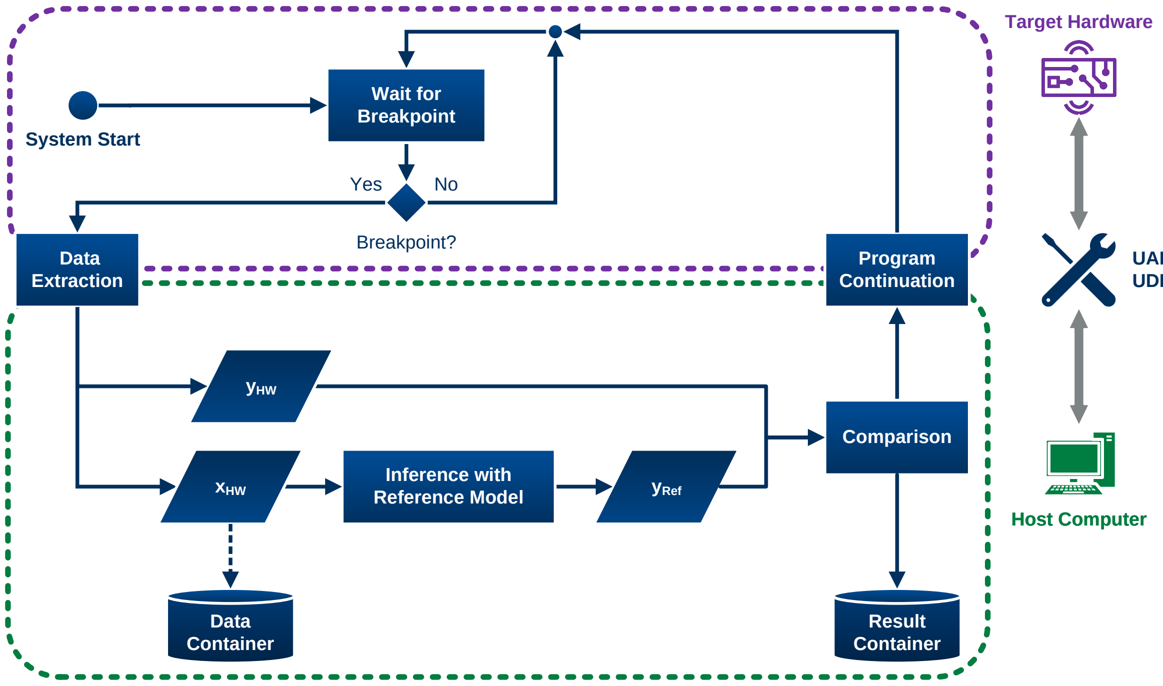


Figure 2. Schematic structure of the diagnostic loop for the continuous comparator-based analysis of the hardware and model outputs. Purple dashed frame: Processes belonging to the context of the target hardware. Green dashed frame: Processes belonging to the context of the host computer.

with W and H as the image width and height and C the color channel, which is generally $C = 3$ for Red Green Blue (RGB) images or $C = 1$ for grayscale images.

A common novice example of neural networks is a classification network which assigns to the input matrix x the classes ”cat” or ”dog”. In this case, the input matrix might have the dimensions $\dim(x) = (128, 128, 3)$ and the output is typically encoded by a two-element output matrix y with $\dim(y) = (2, 1)$ where the elements correspond to the class probability for ”cat” or ”dog”.

B. Inference with Reference Model and Comparison

Depending on availability, either the Golden or Silver Model or both can be used as the reference model. The inference with the chosen reference model is performed on the host computer and yields the reference output matrix y_{Ref}.

Afterwards, a comparison between y_{HW} and y_{Ref} is performed. If the hardware implementation of the neural network is correctly implemented, the following relationships between the output matrices must apply in every single iteration of the diagnosis loop:

$$y_{\text{HW}} = y_{\text{Ref, Silver}} \tag{2}$$

$$y_{\text{HW}} \approx y_{\text{Ref, Golden}} \tag{3}$$

The relationship from equation (2) can be easily checked using a binary equivalence test for y_{HW} and $y_{\text{Ref, Silver}}$, as shown in (4).

$$\text{Eq}\left(y_{\text{HW}}, y_{\text{Ref, Silver}}\right) = \begin{cases} \text{True}, & y_{\text{HW}} = y_{\text{Ref,Silver}} \\ \text{False}, & y_{\text{HW}} \neq y_{\text{Ref,Silver}} \end{cases} \tag{4}$$

If they match exactly, ”True” is returned, otherwise ”False”. For the second relationship from equation (3), the deviations to be expected are quantified by a difference metric, for example the Mean Squared Error (MSE), as shown in (5).

$$\text{MSE}\left(y_{\text{HW}}, y_{\text{Ref, Golden}}\right) = \frac{1}{N} \sum_{\substack{u \in y_{\text{HW}} \\ v \in y_{\text{Ref,Golden}}}} (u - v)^2 \tag{5}$$

Other difference metrics, like the squared Euclidean [14] distance are suitable as well.

The results of the comparison are stored for each iteration of the diagnosis loop for later visualization and evaluation purposes. Optionally, the input matrices x_{HW} of each diagnosis iteration can be stored as well. This data can later be used for training or validation purposes.

C. Program Continuation and Wait for Breakpoint

All necessary interactions with the target hardware can be effectively implemented with the Python automation interface of UDE. The debugger software UDE not only supports most high-end micro controllers but also multi-core SoCs (System on Chips), which are well suited for AI applications. With the access devices of the UAD family, also fast and secure communication with the respective target system is ensured.

Additionally to the data extraction process, the program flow needs to be controlled by starting the system and setting a breakpoint where the input and output matrices can be read. After the inference and comparison processes are done, the program is resumed until the next breakpoint is reached.

In this context, setting the breakpoints is not trivial. It is essential to ensure that the read input and output matrices belong together. As a rule, this must be done manually and for each application individually.

V. The Practice Test

In order to clarify whether the diagnostic strategy described can generally be implemented in practice, extensive tests were carried out with conventional development boards for different AI applications. Two of the applications are described in detail below.

A. Acoustic Scene Classification

STMicroelectronics' Sensor Tile Kit (STEVAL-STLCS01V1) is an all-round sensor system [15]. It is equipped with the following components, among others:

- STM32L476JGY Low Power MCU with Arm Cortex-M4 Floating Point Unit
- BlueNRG-MS-Bluetooth-Prozessor
- Various sensers: microphone, barometer, thermometer, accelerometer, etc.

In Figure 3, the hardware setup consisting of STM32 Sensor Tile and UAD2$^{\mathrm{pro}}$ is visualized.

AI applications for the sensor tile are already pre-implemented in the FP-AI-Sensing1 function package provided by ST from the STM32Cube software development system [16]. From these examples, the Acoustic Scene Classification (ASC) was selected for the practical test of the diagnostic concept. In this application, the ambient noise is analyzed with a neural network based on microphone data and assigned to the classes "indoor", "outdoor" and "vehicle". The input matrix x_{HW} is a transformed spectrogram with the dimensions $\dim(x_{\mathrm{HW}}) = (32, 30, 1)$ and the output matrix y_{HW} is a three-element matrix $\dim(y_{\mathrm{HW}}) = (3, 1)$ with the respective class probabilities. The golden model and the silver

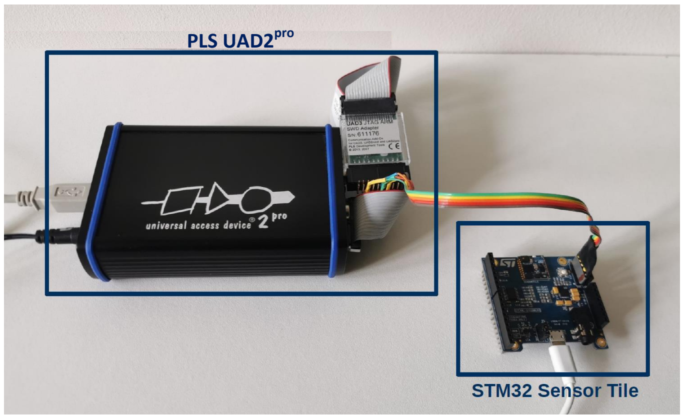


Figure 3. STM32 Sensor Tile connected to the PLS UAD2$^{\mathrm{pro}}$.

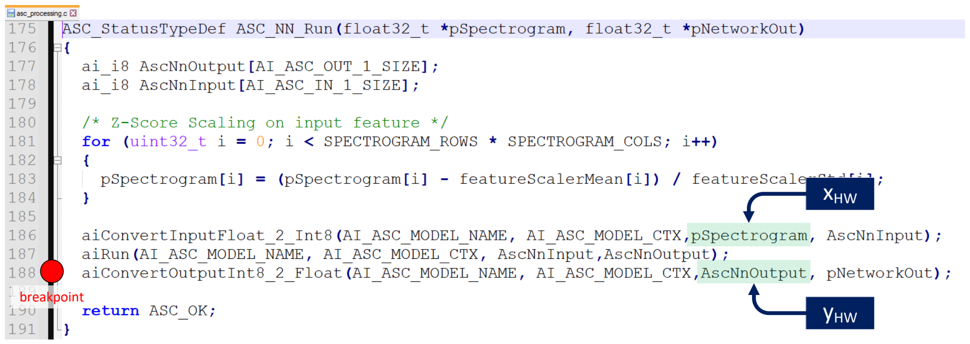
model of the neural network for the ASC are already available in the FP-AISensing1 function package, so the training and adaptation phases can be skipped. The Golden and Silver Models are available as a Tensorflow Keras model (`ASC.h5`) and as a Tensorflow Lite 8-Bit quantized Model (`ASC.tfl`).

In order to determine suitable interfaces for data transfer using the UDE debugger, the FP-AI-Sensing1 function package must first be compiled and build with the STM32Cube Integrated Development Environment (IDE). The resulting Executable and Linkable Format (ELF) file `SENSING1.elf` is then programmed into the flash memory of the sensor tile with the UDE Multi Program Loader and a breakpoint is set in line 188 of the `asc_processing.c` module for the ASC diagnosis, as shown in Figure 4. At this code line, both the input matrix x_{HW} and the associated output matrix y_{HW} can be read via the automation interface of the UDE.

The diagnostic loop can then be run using the analysis system and the reference models. In Figure 5 an example of diagnosis loop with 100 iterations is shown. It can be seen that the output matrices of the hardware implementation and those of the silver model always match. The averaged MSE is $\mu_{\mathrm{MSE}} = 3.3 \cdot 10^{-4}$ and is therefore within a range which is to be expected based on the tensorflow lite quantization. Further tests with larger iteration counts and different acoustic environments have shown the same behavior, so that in summary it can be said that the equations 2 and 3 apply and the hardware implementation is working correctly.

B. Recognizing People

A development board for image processing algorithms from STMicroelectronics based on the 32-bit low-power MCU STM32H743VI was selected as a further application example of the diagnostics environment. The OpenMV (Open Source Machine Vision) Cam H7 camera module can be used, among other things, to implement neural networks for computer vision applications [17]. The OpenMV Github repository offers already implemented AI entry-level examples [18]. The focus of the following test is on recognizing people. A classification network assigns a two-element probability matrix y_{HW} with the classes "person" and "no person" to the input image matrix



```
asc_processing.c
ASC_StatusTypeDef ASC_NN_Run(float32_t *pSpectrogram, float32_t *pNetworkOut)
{
  ai_i8 AscNnOutput[AI_ASC_OUT_1_SIZE];
  ai_i8 AscNnInput[AI_ASC_IN_1_SIZE];

  /* Z-Score Scaling on input feature */
  for (uint32_t i = 0; i < SPECTROGRAM_ROWS * SPECTROGRAM_COLS; i++)
  {
    pSpectrogram[i] = (pSpectrogram[i] - featureScalerMean[i]) / featureScalerStd[i];
  }

  aiConvertInputFloat_2_Int8(AI_ASC_MODEL_NAME, AI_ASC_MODEL_CTX,pSpectrogram, AscNnInput);
  aiRun(AI_ASC_MODEL_NAME, AI_ASC_MODEL_CTX, AscNnInput,AscNnOutput);
  aiConvertOutputInt8_2_Float(AI_ASC_MODEL_NAME, AI_ASC_MODEL_CTX,AscNnOutput, pNetworkOut);

  return ASC_OK;
}
```

Figure 4. View of the core function of ASC. At the position of the selected breakpoint, the input matrix x_{HW} and the associated output matrix y_{HW} for the diagnostics from the hardware can be read.

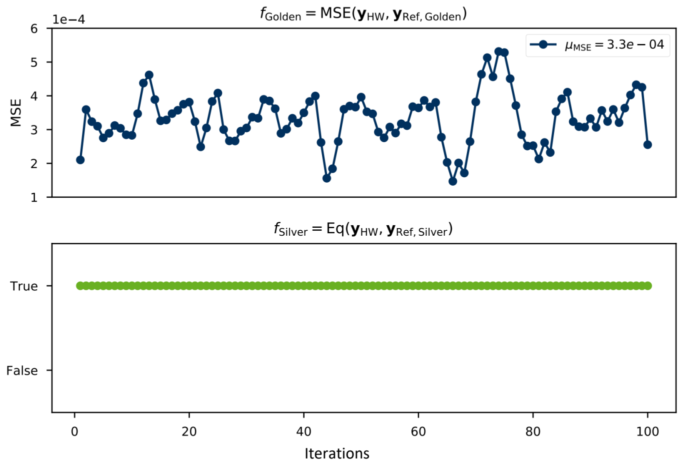
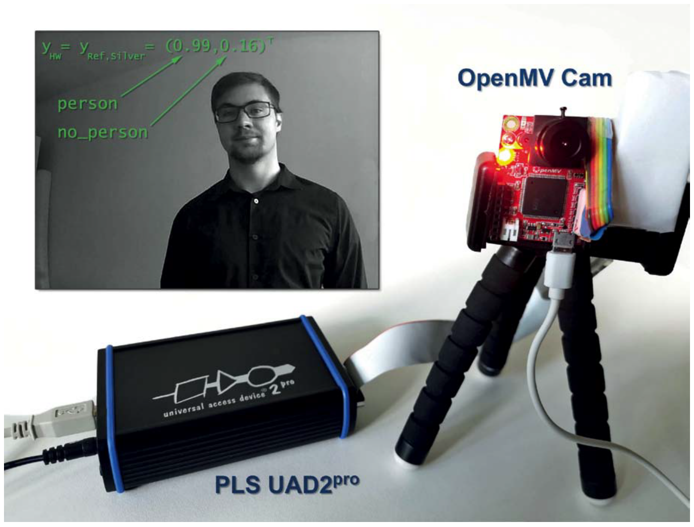


Figure 5. Evaluation diagram with 100 iterations of the diagnostic loop for the Sensor Tile diagnosis system.



$y_{HW} = y_{Ref,Silver} = (0.99, 0.16)^T$
person
no_person
OpenMV Cam
universal access device 2pro
PLS UAD2pro

Figure 6. Hardware setup with OpenMV Cam H7 connected with the PLS UAD$^{\mathrm{pro}}$ and an example of an input image for recognizing people. The reference models and hardware implementation certainly recognized the person (Ilkay Wunderlich) in the image.

x_{HW}. The neural network uses a grayscale image with the resolution $\dim(x_{\mathrm{HW}}) = (640, 480)$ as the input matrix for the person classification. Similar to the implementation of the diagnostic system for the ASC, a breakpoint is set at a suitable point in order to read out the input and output matrices via UDE. Here, too, the evaluation of the diagnostic loop confirms that the outputs of the OpenMV Cam and the predictions of the silver model always match. The deviations between y_{HW} and $y_{\mathrm{Ref,\ Golden}}$ are withing acceptable range as well.

Figure 6 shows the hardware setup, an example image for person detection and the corresponding output matrices y_{HW} and $y_{\mathrm{Ref,\ Silver}}$.

Using the developed diagnostic system based on the debugger UDE, not only the correctness of the AI implementation can be verified for the computer vision application but also the collected data x_{HW} from each iteration is saved and can later be used as new training or validation data. Especially for computer vision tasks, it is a great benefit to collect data of the target camera with its distinctive lens and exposure settings to increase the performance of the neural network.

VI. Conclusion

As the examined application examples show, efficient diagnostic systems can be implemented with suitable tools for AI-based embedded applications. With the comparisons between the predictions of the target hardware and the reference models, developers can ensure that the AI implementation is working correctly. The debugger UDE realizes the Python-based diagnostic system as a system in the loop. Additional features such as data collection, processing time measurement or continuous integration can be flexibly integrated into or around the diagnostic loop via the UDE Python interface.

The future work is on the one hand to generalize the Python-based concepts in order to add a direct neural network diagnosis feature to UDE and, on the other hand, to extend the approach to other emerging or established AI methods such as spiking neural networks, decision trees, hidden Markov models, etc.

ACKNOWLEDGMENT

The investigation of the described novel diagnosis concept was developed by PLS as well as the chair of VLSI-Design, Diagnostic and Architecture of the Institute of Computer Engineering at the Technical University Dresden.

REFERENCES

[1] University of Central Florida, ”AI can detect COVID-19 in the lungs like a virtual physician, new study shows: Algorithm can accurately identify COVID-19 cases, as well as distinguish them from influenza,” https://www.sciencedaily.com/releases/2020/09/200930144426.htm, ScienceDaily, 2020.

[2] X. Zeng, Z. Wang and Y. Hu, ”Enabling Efficient Deep Convolutional Neural Network-Based Sensor Fusion for Autonomous Driving,” https://doi.org/10.1145/3489517.3530444, Association for Computing Machinery, 2022.

[3] Y. Fisher et al., ”BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning,” https://arxiv.org/abs/1805.04687, arXiv, 2018.

[4] M. Cordts et al., ”The Cityscapes Dataset for Semantic Urban Scene Understanding,” https://arxiv.org/abs/1604.01685, arXiv, 2016.

[5] J. Zhang and J. Li, ”Testing and verification of neural-network-based safety-critical control software: A systematic literature review,” https://www.sciencedirect.com/science/article/pii/S0950584920300471, Information and Software Technology, 2020.

[6] A. Kirchknopf, D. Slijepcevic and I. Wunderlich ”Explaining YOLO: Leveraging Grad-CAM to Explain Object Detections,” https://diglib.tugraz.at/download.php?id=621f34738ca16&location=datacite, Proceedings of the Workshop of the Austrian Association for Pattern Recognition, 2021.

[7] S. Huang et al., ”Design and Implementation of Convolutional Neural Network Accelerator with Variable Layer-by-Layer Debugging,” https://doi.org/10.1145/3234804.3234806, Association for Computing Machinery, 2018.

[8] PLS Programmierbare Logik & Systeme GmbH, ”Universal Debug Engine (UDE),” https://www.pls-mc.com/products/universal-debug-engine, retrieved 14.12.2022.

[9] M. Abadi et al., ”TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,” https://www.tensorflow.org/, 2015.

[10] A. Paszke et al. ”PyTorch: An Imperative Style, High-Performance Deep Learning Library,” http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf, 2019.

[11] I. Wunderlich, B. Koch, and S. Schönfeld, ”An Overview of Arithmetic Adaptations for Inference of Convolutional Neural Networks on Re-configurable Hardware,” ALLDATA 2020, The Sixth International Conference on Big Data, Small Data, Linked Data and Open Data, pp. 34–40, 2020.

[12] STMicroelectronics, ”X-CUBE-AI: AI expansion pack for STM32CubeMX,” https://www.st.com/en/embedded-software/x-cube-ai.html, retrieved 14.12.2022.

[13] Tensorflow, ”Tensorflow Lite,” https://www.tensorflow.org/lite, retrieved 14.12.2022.

[14] N. H. Spencer, ”Squared Euclidean Distances,” Essentials of Multivariate Data Analysis, CRC Press, p. 95, ISBN 978-1-4665-8479-2, 2013.

[15] STMicroelectronics, ”STEVAL-STLCS01V1: SensorTile connectable sensor node: plug or solder,” https://www.st.com/en/evaluation-tools/steval-stlcs01v1.html, retrieved 14.12.2022.

[16] STMicroelectronics, ”FP-AI-SENSING1: function pack for ultra-low power IoT node with artificial intelligence (AI) application based on audio and motion sensing,” https://www.st.com/en/embedded-software/fp-ai-sensing1.html, retrieved 14.12.2022.

[17] K. W. Agyeman, I. Abdelkader, K. Wong and Y. Xu, ”OpenMV Cam,” https://openmv.io/, retrieved 14.12.2022.

[18] K. W. Agyeman, I. Abdelkader, K. Wong and Y. Xu, ”OpenMV: Open-Source Machine Vision,” https://github.com/openmv/openmv, retrieved 14.12.2020.