IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Analysis of Edge-Optimized Deep Learning Classifiers for Radar-based Gesture Recognition

MATEUSZ CHMURSKI[1,2], MARIUSZ ZUBERT[2], (MEMBER, IEEE), KAY BIERZYNSKI[1], AVIK SANTRA[1], (Senior Member, IEEE)

[1]Infineon Technologies AG, 85579 Neubiberg, Germany
[2]Lodz University of Technology, 90924 Lodz, Poland

Corresponding author: Mateusz Chmurski (e-mail: Mateusz.Chmurski@infineon.com).

This work has received funding from the Electronic Components and Systems for European Leadership Joint Undertaking under grant agreement No 826655 (Tempo). This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Belgium, France, Germany, Switzerland, Netherlands. The publication has been funded by the internal university grant of the Lodz University of Technology.

ABSTRACT The increasing significance of technology in daily lives led to the need for the development of convenient methods of human-computer interaction (HCI). Given that the existing HCI approaches exhibit various limitations, hand gesture recognition-based HCI may serve as a more intuitive mode of human-machine interaction in many situations. In addition, the system has to be deployable on low-power devices for applicability in broadly defined Internet of Things (IoT) and smart home solutions. Recent advances exhibit the potential of deep learning models for gesture classification, whereas they are still limited to high-performance hardware. Embedded neural network accelerators are constrained in terms of available memory, central processing unit (CPU) clock speed, graphics processing unit (GPU) performance, and a number of supported operations. The aforementioned problems are addressed in this paper by namely two approaches - simplifying the signal processing pipeline to avoid recurrent structures and efficient topological design. This paper employs an intuitive scheme allowing for the generation of the data in the compressed form from the sequence of range-Doppler images (RDI). Thus, it allows for the design of a neural classifier avoiding the usage of recurrent layers. The proposed framework has been optimized for Intel® Neural Compute Stick 2 (Intel® NCS 2), at the same time achieving promising classification accuracy of 97.57%. To confirm the robustness of the proposed algorithm, five independent persons have been involved in the algorithm testing process.

INDEX TERMS Accelerator, Data Augmentation, Edge Computing, FMCW, Gesture Recognition, Neural Networks, DNNs, Optimization, Radar, Intel NCS2

I. INTRODUCTION

Currently, artificial intelligence (AI) has led to rise in smart sensors and devices in the market [1]. It is leveraged in big data to Internet of Things (IoT) devices to smart homes to autonomous cars [2]. This growth is manifested in their increasing performance in pattern recognition tasks. DNNs and their varieties, namely, convolutional neural networks (CNNs), Recurrent Neural Networks (RNNs), Generative Adversarial Networks (GANs) [3], are becoming the most important methods in pattern recognition problems. Deep Learning methods have made distinct breakthroughs in a broad spectrum of fields, including computer vision, speech recognition, natural language processing, medical diagnosis, and board games [3]–[5].

In the last few years, researches carried out by numerous teams in R&D centers have led to the remarkable development of architectures such as AlexNet [6], VGGNet [7], and ResNet [8], which can learn a deep representation of the data and have become one of the most popular architectures in the field of computer vision. Recent advances in deep learning are the driving force for further research heading towards optimizing those solutions and deployment on the edge devices. According to Ericsson [4], 45.00% of the global internet congestion will be generated by IoT devices, which confirms

the need for thorough research in this direction.

Neural network optimization, including both architectural design and post training optimization, gives the developers the possibility to convert a very complex deep learning model into a streamlined implementation [9]. Architectural optimization in mobile deep learning models are based on the replacement of traditional convolutions with depthwise separable convolutions, squeezing the output channels of the network using 1x1 convolution [10], or splitting up the channels into groups and applying a depthwise convolution with a different kernel size to each group [10]. Such architectural optimizations are part of several renowned architectures such as MobileNetV1 [10], SqueezeNet [11], MixNet [12], and GoogleLeNet [13], [14]. Other approaches involve hyperparameter configuration and automatic architecture search [15], [16]. Post-training quantizations involve network pruning, quantizations, format optimization [17], [18]. Furthermore, another challenge is the choice of appropriate optimization strategy that highly depends on the data characteristics. In this regard, system optimization gives the developers the capacity to tune the system for the best system performance [19], [20].

The main idea behind the notion of edge computing is pushing the computational and communication resources from the cloud to the edge of networks to perform computations [3], [4], [21], [22], thereby avoiding unnecessary communication latencies, providing a privacy protection capability, and enabling faster response for the end-users [3], [4], [21], [22]. Thus, the necessity to optimize neural network models for such edge devices is of fundamental significance towards the improvement of an overall system performance.

The typical machine learning pipeline consists of the following steps: data collection, model training, and inference [23], [24]. For resource constrained embedded devices, an important aspect to be considered is an appropriate data processing, which allows for a low-dimensional representation of the data [25]. Low-dimensional data allows for the design of the less computationally complex algorithm. During the prototyping of the intelligent systems, one of the major concerns in designing a robust system is the amount of available data [26], [27]. It is unfeasible for the system designer to provide a huge source of data to train the system that does not overfit. For that reason, to avoid this problem, it is necessary to use data augmentation to increase the number of training examples [26], [27]. However, the data augmentation is very application-specific. For instance, the same data augmentation applied to images does not fit the model training for radar images or medical images.

The next steps in model generation are model training and optimization, where optimization involves appropriate hyperparameters tuning that lead to the best results [28]. Once a model is finalized, the model is deployed for inference on the edge device. Post-training optimization involves weight quantizations, node fusions, and network pruning for optimized performance for the specific embedded deployment.

Gesture sensing technology is one of the most intuitive and common approaches in the field of human-computer interaction (HCI) giving computers the possibility to understand human gestures and analyze human intentions [29]–[31].

Recent advances in this field are mainly vision-based products, (i.e., they use camera sensors, e.g., RGB and ToF) [29]–[31]. These systems are mainly based on analyzing the spatially-temporal relations between consecutive frames using CNN3D and long short-term memory (LSTM) [32], [33]. The analysis of consecutive frames is a computationally complex task which can not be implemented on resource-constrained devices such as contemporary neural network accelerators [34]–[38]. In the era of continuously increasing privacy needs, personal data protection, and the popularity of energy-efficient solutions, the development of algorithms that can be deployed on the edge of the network is a significant challenge [5], [22]. Therefore, the main focus of this paper is placed on the design of an intuitive signal processing scheme allowing for optimization and deployment of deep learning gesture classifier on the edge.

In this work, we propose an optimized deep learning model on the edge computing platform. First of all, to create the dataset, we collect data from the *BGT60TR13C* FMCW radar sensor [39]. The samples are assigned to a corresponding label, and a sophisticated preprocessing scheme is applied to raw radar signal, which allows for avoidance of computationally complex neural network operators [38]. Secondly, the dedicated deep learning topology is trained on the collected data and compared with state-of-the-art topologies [40]. In order to find the model with the best hyperparameters, the 5-fold cross-validation has been applied. Thirdly, the dedicated CNN layers pruning is applied [21], [41], the model is fine-tuned and tested. The pruned models are deployed on Rasperry Pi4 with Intel® NCS 2 to enable the decentralized and embedded application. The dataset has four gestures: up-down, down-up, left-right, rubbing. In order to prevent our system from overfitting effect, the sophisticated data augmentation scheme is applied during training. The model is trained using CPU optimization and hyperparameter tuning. Keras [42] with Tensorflow [18] backend is used as a framework for model design and training. The dedicated topology is proposed and benchmarked against topologies utilized in embedded application deployment. The Model Optimizer (MO) [43] is utilized to tune the model and deploy it on the edge device.

The main contributions of this paper are as follows:

1) The implementation of proof-of-concept edge-efficient gesture recognition radar system.[1]
2) Design of the data preprocessing, enabling the avoidance of application ineffective deep learning operators.
3) Rigorous comparison among topologies for edge gesture recognition solution.
4) Family of the deep neural classifiers optimized for the deployment on the NCS 2.

[1]Demonstration video: https://youtu.be/TrDcmcVKpiY.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

II. RELATED WORKS

Lien et al. [44] has taken the initial steps in exploring the radar as a new sensing modality in the field of gesture recognition, proposing the whole gesture recognition pipeline (i.e., data collection, digital signal pre-processing, signal transformations, feature extraction and training the classifier). They have chosen the low-dimensional features to implement possibly streamlined gesture sensing system. The extracted features have been used as an input for the Random Forest Classifier. The developed solution has been tested on two energy efficient platforms such as Raspberry Pi2 running at 900 MHz and the Qualcomm Snapdragon 400 (APQ8028) running at 1.6 GHz.

Molchanov et al. [45] has proposed the gesture sensing system based on the radar. They have developed the signal processing methodology, which allowed them to generate the range-Doppler maps (RDMs) and estimate the angle information. The angle information has been used for the calibration of the radar with the Time-of-Flight (ToF) camera. The result of this work has been part of the larger multi-sensor system, which included an RGB camera and a depth sensor [46]. They developed the multi-sensor gesture sensing system for automotive applications basing their solution on the CNN3D classifier. They achieved satisfactory classification results, however their solution is undeployable on the edge device, such as NCS 2, due to lack of support for CNN3D.

Hazra et al. [47], [48] proposed a radar gesture sensing system based on Long Recurrent All Convolutional Neural Network (LRACN) making use of time-distributed layer wrapper, which applies the same set of convolutional layers to each input time step. The extracted feature vector is passed to an LSTM layer for temporal feature modeling in the next phase. They show in their results, satisfying recognition accuracy 94.0%; however, the proposed algorithm is not deployable on resource-constrained devices (i.e., NCS 2, Edge TPU). The inference time is 1.0 s, which prevents it from providing a real-time interactive experience.

Zhang et al. [33] suggested an alternative solution. Their approach relies on the utilization of CNN3D with LSTM. They use CNN3D for short spatial-temporal modeling and LSTM for global temporal feature extraction.
Their results exhibit a satisfying average recognition capability of 94.00%. However, this classifier utilizes the operations, which significantly increase the memory footprint, and resource-constrained devices (i.e., NCS 2, Edge TPU) do not support them [38].

Hazra et al. [32], [48] suggested the classifier based on CNN3D feature embedding. This work essentially proposes the use of CNN3D with triplet loss for learning the embedded feature vectors, which are subsequently utilized by kNN (k-Nearest Neighbours) algorithm for classification. This approach exhibits similar limitations to the mentioned above.

Ahmed et al. [49] propose a hand gesture recognition system using an IR-UWB radar with an inception module based classifier. In this work, they implement a neural network classifier with nine 3D inception modules. The results of this research outperform state-of-the-art solutions. The complicated signal processing scheme results in high resource consumption, and it imposes the utilization of operations not supported by the OpenVINO framework [38].

III. BACKGROUND REVIEW

This section presents the used tools, optimizations for edge deployment, and model assessment methods. The next subsections discuss each of the components in more detail.

A. DEEP LEARNING FRAMEWORK

Keras [42] with TensorFlow [18] backend is used as a baseline framework for the design of our classifier. It is an open-source tool that supports fast prototyping with numerous auxiliary tools for efficient model analysis. In this work, we use the dedicated version of TensorFlow for Intel processors [50], which is built on top of the Intel® Math Kernel Library for Deep Neural Network (Intel® MKL-DNN) [51]. Furthermore, TensorFlow can be efficiently deployed on a wide range of devices such as CPUs, GPUs, and low-power devices with minimum effort.

B. TOPOLOGY DESIGN AND OPTIMIZATIONS

There are several aspects to be considered designing the topology, which must be deployable on a resource-constrained device, i.e., training time, model size, memory footprint during model execution, accuracy, the instruction set of the target device, and inference time. Those limitations generate the topology design problem, which must be taken into account during experimentations and system deployment. InceptionV1 and MobileNetV1 are the topologies supported by the edge device used in this work (i.e., Intel® NCS 2). This paper proposes a family of custom deep learning classifiers, which have been designed and optimized for the embedded deployment and benchmarked against baseline classifiers.

The number of trainable parameters is critical for resource-constrained devices and needs to be carefully optimized. While a shallow neural network leads to underfitting, a deeper neural network with a large number of trainable parameters can lead to overfitting and poor generalization if enough training data are not available. An optimized embedded system design needs to account for both accuracy and smaller memory footprint.

C. MODEL ANALYSIS

This section lists the classification metrics used in this work for the assessment of the model performance.

- $Precision$. This metrics is defined in the following way:

$$P = \frac{TP}{TP + FP} \tag{1}$$

where: P is a $Precision$, TP is the number of true positives, and FP is the number of false positives.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

$Precision$ is the measure of the ability to label the sample as truly positive. The worst value is 0, and the best is 1.

- $Recall$. This metrics is defined in the following way:

$$R = \frac{TP}{TP + FN} \tag{2}$$

where: R is a $Recall$, TP is the number of true positives, and FN is the number of false negatives. $Recall$ is the measure of what percentage of the positive samples is correctly classified. The worst value is 0, and the best is 1.

- $F1\ score$. This metrics is described with the following equation:

$$F1 = \frac{2(P \times R)}{P + R} \tag{3}$$

where: $F1$ is an $F1\ score$, P is the precision, and R is the recall. Intuitively, this is the combination of $precision$ and $recall$ in a single number. The $F1\ score$ is the number between 0 and 1, which is a harmonic mean of precision and recall.

IV. SYSTEM DESCRIPTION AND IMPLEMENTATION

This section considers system components and implementation details (i.e., hardware details, operating parameters, experimental setup, proposed signal processing, data augmentation and gesture vocabulary).

A. RADAR

The radar used in this work is the *BGT60TR13C* frequency modulated continuous wave (FMCW) radar sensor manufactured by *Infineon Technologies AG* with a center frequency of 60.0 GHz. The *BGT60TR13C* is a low-cost, low-power, and high-resolution solution. Fig. 1 depicts the radar board. Fig. 2 presents the block diagram of radar sensor. Radar chip is equipped with three receive channels RX, and one transmit channel TX. The core functionality of *BGT60TR13C* is to transmit the FMCW signal via the TX channel and receive the echo signals from the target object via one of the three RX channels. Both transmitted and received signals are mixed and passed to a baseband chain and an analog to digital converter (ADC) with 12 bits resolution and up to 4 MSps sampling rate. Each baseband chain consists of a high pass filter, a voltage gain amplifier (VGA), and antialiasing filters. The digitized signal is stored in a FIFO buffer. The data is transferred to an external host for further signal processing.

FIGURE 1. BGT60TR13C Radar Board [39]

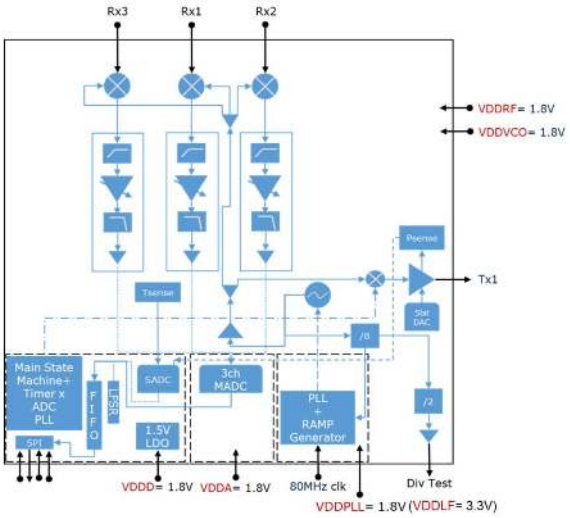

FIGURE 2. BGT60TR13C Radar Sensor Block Diagram [39]

B. RADAR SIGNAL MODEL

A transmitted FMCW waveform can be expressed in the following form [52]:

$$s_T(t) = A_T \cos\left(2\pi f_c t + 2\pi \int_0^t f_T(\tau)\, d\tau\right) \tag{4}$$

where $f_T = \frac{B}{T} \cdot \tau$ is the transmit frequency as a linear function of time, f_c is the carrier frequency, B is the bandwidth, A_T is the transmitted signal amplitude, and T is the time duration.

Reflected signal is received with the following time delay t_d:

$$t_d = 2 \cdot \frac{R_0 + vt}{c} \tag{5}$$

with the Doppler shift:

$$f_D = -2 \cdot \frac{f_c v}{c} \tag{6}$$

The receive frequency can be formulated as follows [52]:

$$f_R(t) = \frac{B}{T_c}(t - t_d) + f_D \tag{7}$$

where R_0 is the range at t = 0, v is the target velocity, and c is the speed of light.

The received signal can be described as follows [52]:

$$s_R(t) = A_R \cos\left(2\pi f_c(t - t_d) + 2\pi \int_0^t f_R(\tau)\, d\tau\right) \tag{8}$$

where A_R is the received signal amplitude. An intermediate frequency (IF) signal is generated as a result of mixing the received signal and the transmitted signal. The IF signal is forwarded to the low-pass filter and it is expressed with the following formula [52]:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

$$s_{IF}(t) = \frac{1}{2}\cos\left(2\pi\left(f_c \cdot \frac{2R_0}{c}\right) + 2\pi\left(\frac{2R_0}{c}\cdot\frac{B}{T} + \frac{2f_c v}{c}\right)t\right) \tag{9}$$

C. RADAR OPERATING PARAMETERS

The *BGT60TR13C* operates in the 60 GHz unlicensed band and it provides a resolution in centimeters. This feature makes this device suitable for the hand gesture recognition application. Since *BGT60TR13C* is operating in the V-band and it is transmitting signal of up to 6 GHz (57 GHz - 63 GHz) bandwidth, thereby it may provide a range resolution Δr of 2.5 cm and a Doppler resolution Δv of about 122 cm/s. Δr and Δv can be calculated using the following equations:

$$\Delta r = \frac{c}{2B} = 2.5\ cm \tag{10}$$

$$\Delta v = \frac{c}{2f_c}\cdot\frac{1}{n_c T_c} = 122\ cm/s \tag{11}$$

where c is the speed of light, approximately $3\times10^8\ m/s$, and f_c is the center frequency between 57 and 63 GHz, which is set to 60 GHz. Accordingly, B is the bandwidth of the chirp signal and it is calculated as 6 GHz. T_c is the chirp duration, n_c is the number of repeatedly transmitted chirp signals, and they are set to 37 μs and 64, respectively. In our application, the *BGT60TR13C* sends a periodic chirp signal through a transmitting antenna, and it receives a signal reflected from an object using one receiving antenna. The transmitted chirp signal is frequency modulated by a sawtooth wave function and the time delay τ and the Doppler shift f_D occur between the transmitted signal and the received signal, as shown in Fig. 3. The blue and red line in Fig. 3 represent the transmitted signal and reflected signal, respectively. In this work, the chirp signal consists of 32 samples. The frame frequency is configured to 10 Hz. The time delay τ is caused by the distance between the radar and the object reflecting the signal, the Doppler shift f_D is caused by the movement of the object relative to radar.

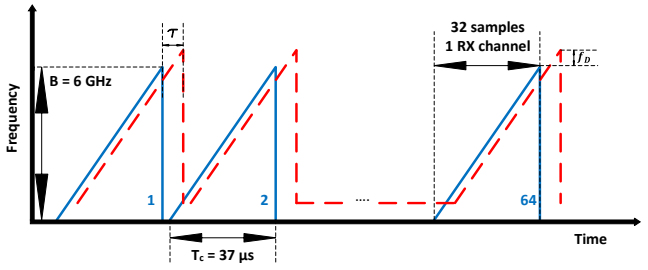

FIGURE 3. FMCW Waveform in the Frequency Domain

D. EXPERIMENTAL SETUP

The experimental setup consists of *BGT60TR13C* radar sensor, 3D-printed case which is fixed to the camera tripod, Raspberry Pi4 and NCS 2. Fig. 4 presents the experimental setup.

(a) Raspberry Pi4 with NCS 2 (b) Tripod with the case and Radar Sensor (c) Experimental Setup

FIGURE 4. Experimental Setup

E. RADAR SIGNAL PROCESSING

The radar signal is processed in multiple steps. First, the intermediate frequency signal $s_{IF}(t)$ corresponding to time within a chirp, also referred to as fast time, is brought to zero by subtracting the mean value of the chirp from each of the samples. Then, the raw intermediate frequency signal is multiplied with the Hann window function. The range r_t of the target in front of the radar is computed with the first order fast Fourier transform (FFT) along the fast time with an FFT size of 128, from which the positive part is taken. Next, in order to resolve the signal in velocity v_t, the result of first order FFT is multiplied with the Hann function along the slow time direction, and the second order FFT is calculated with an FFT size of 64 along the slow time direction, forming the range-Doppler image (RDI) with 64 x 64 dimensions. In order to increase the signal to noise ratio, the absolute value of RDI is smoothened with wiener and median filter. Then to remove ghost targets, RDI is thresholded with OS-CFAR in both directions. Fig. 5 and 6 illustrate detailed and general signal processing workflow, respectively.

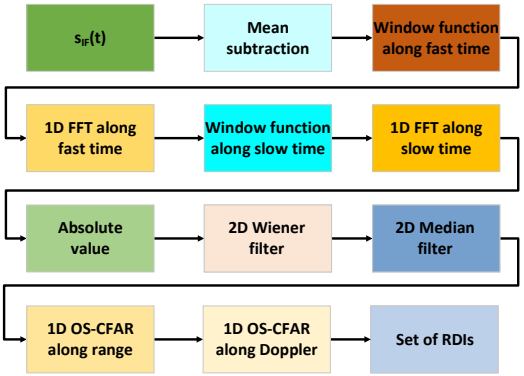

FIGURE 5. Detailed Signal Processing Workflow

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

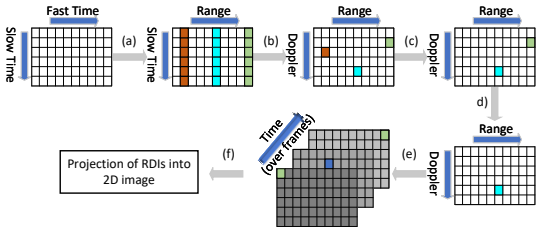

FIGURE 6. Signal Processing Workflow

In this work, we introduce an approach allowing for the transformation of the data from a high-dimensional space into a low-dimensional space and the formation of the range-time dependency map. Derived RDIs make up the volume $\mathcal{S}_R \in \mathbb{R}^{t \times x \times y \times f}$ where $t \geq 1$. Each timestep stores an RDI denoted by $\Phi \in \mathbb{R}^{x \times y \times f}$, where $x \times y$ are range and Doppler dimensions and f is the number of feature channels, which is in our case 1, as only one RX channel is used. A single RDI is a matrix of dimensions $m \times n$, where $x \in \{1, .., m\}$ and $y \in \{1, .., n\}$.

$$\Phi_{m,n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

We have to find an index (i, j) of the largest element $a_{i,j}^{max}$ in the matrix, denoting $I = \{1, ..., m\}$ and $J = \{1, ..., n\}$ as sets of row and column indices. There is an index $i, j, \exists i \in I$ and $\exists j \in J$ such that a_{ij} is the maximum element of the matrix. The next phase is slicing the $C_{1\times n}^{t}$ vector representing the distance of the object in the given time step of the gesture. Subseqently, the $C_{1\times n}^{t}$ vector is transposed ${C_{1\times n}^{t}}^{T}$ and concatenated with the subsequent time slices forming range-time plot. The steps of the projection algorithm are depicted in Fig. 7. As it is illustrated in Fig. 7, the projection algorithm produces the unique 2D gesture signature, which can be used for training ML classifier.

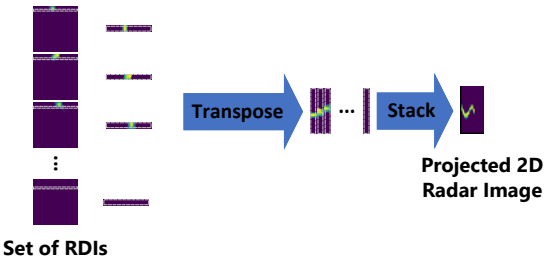

FIGURE 7. Projection of the extracted RDIs into 2D Radar Image

F. THE GESTURE VOCABULARY

The system defines four gestures: Up-down, Down-up, Left-right, Rubbing

- Up-down
- Down-up
- Left-right
- Rubbing

The training samples were collected in different environments by five different individuals to ensure the maximum variance of the dataset. Each sample represents unique gesture signature, which is used for training. The examples of the gesture signatures are depicted in the Fig. 8.

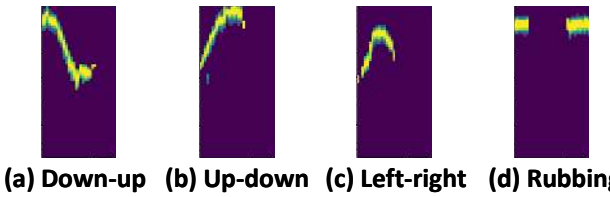

FIGURE 8. Exemplary Data Samples

G. DATA AUGMENTATION

To increase the number of training samples and handle the imbalance of the dataset, we have applied data augmentation techniques allowing to avoid the overfitting and increase the overall performance of the model. To generate one augmented sample, we iterate over the whole dataset and randomly choose one of the techniques mentioned below, i.e., one data augmentation procedure generates one augmented sample. We apply randomly the following procedures:

- **Random shifting of the 2D Radar Images:**
  2D Radar image is shifted randomly in the time and range dimension ($\pm 0.05x$, $\pm 0.1y$). The empty space is filled with zeros.
- **Zeroing out regions:**
  Random selection of square area (patch) in the 2D Radar Image, which is filled with zeros. Patch sizes are from one pixel up to a square of 5 x 5 pixels.
- **Adding constants to the sequence:**
  To reduce the impact of numerical values, a random integer with a value up to 10% of the maximum pixel value is added to the 2D Radar Image.

The examples of augmented samples are illustrated in Fig. 9.

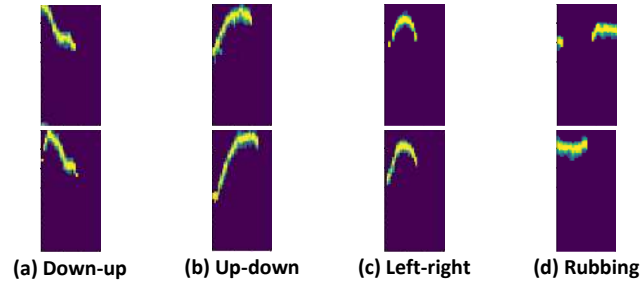

FIGURE 9. Augmented examples

V. TOPOLOGY AND EMBEDDED OPTIMIZATION

This section discusses the proposed topology and the embedded optimization. First of all, we present the details of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

the training environment, which has been used for the model optimization and training. Secondly, we show the details of the training procedure. Thirdly, we present the details of the proposed topology and the design challenges, which have to be met with regards to the model optimizer. Finally, we present the optimization workflow, which has been done on the x86 platform.

A. TRAINING ENVIRONMENT

The training environment of the system is shown in Fig. 10. The computational farm is based on Red Hat Enterprise Linux Server 7.7, twenty seven Intel® Xeon processors with 503 GB RAM memory. The system utilizes Python programming language with the TensorFlow version [53] dedicated for Intel processors based on Intel® MKL-DNN library, which replaces the default Eigen kernels by MKL kernels, optimizes default TensorFlow operations, and offers graph fusion capabilities for faster graph computation.

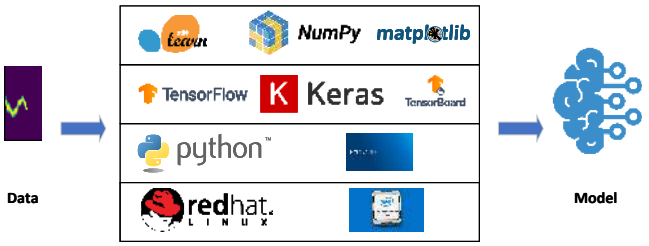

FIGURE 10. Training Environment

B. TRAINING PROCEDURE

Fig. 11 shows the steps to train the model. The first step is the data collection. The second step is a choice of one of the supported frameworks to build the model according to MO requirements. In the next step, model is trained and optimized. Then the system retrains the model and generates intermediate representation (IR) in the form of .bin and .xml files.

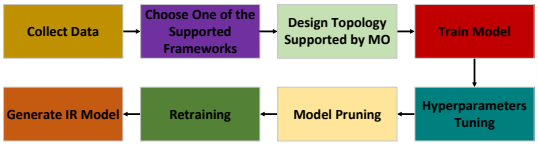

FIGURE 11. Training steps

C. PROPOSED TOPOLOGY

To meet the requirements related to the set of operations supported by the NCS 2. We had to design possibly streamlined topology, avoiding the utilization of such operations as LSTM, TimeDistributed, CNN3D and RNN. Table 1 outlines the set of operations supported by the neural network accelerator used in this project. We can distinguish between three groups of support. Full support, not supported and partially supported. In this case, Batch Normalization is partially supported due to lack of the capability of the MO to fuse the Batch Normalization and CNN2D into one operation, what is manifested in the bigger model size.

TABLE 1. Model Requirements for NCS 2. 3D: CNN3D, BN: Batch Normalization, 2D: CNN2D, TS: Time Distributed. Symbols: FS - fully supported, NO - not supported, PS - partially supported

	RNN	LSTM	3D	BN	DropOut	2D	TS
NCS2	NO	NO	NO	PS	FS	FS	NO

The proposed topology is depicted in the Fig. 12. The model is comprised of six convolutional layers extracting the visual features. All convolutional layers are followed by ReLu activation and MaxPooling2D to decrease the dimensionality of the data. To make our classifier more robust and prevent it from the overfitting effect, we have applied the dropout after the first two convolutions and before the classification layer. The first convolution utilizes kernel size 7x7 with stride 1 to increase field of view of the kernel and extract the most significant features, the next four convolutional layers make use of the kernel size 3x3 and stride 1. The successive convolutional layers increase the number of filters, however the last convolution uses 1x1 kernel size to decrease the dimensionality. The activation function of the classification layer is the softmax. The overall number of parameters is 386614.

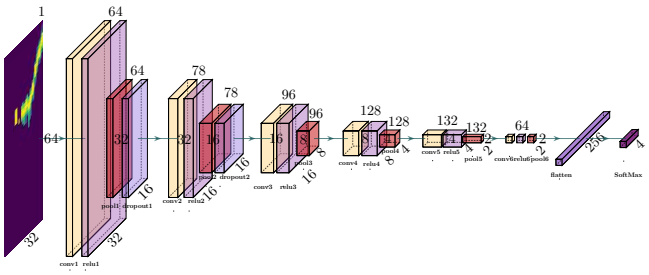

FIGURE 12. Proposed topology

D. SYSTEM WORKFLOW

Fig. 13 presents the system workflow diagram consisting of two stages. The first stage is localized in the cloud, it includes data acquisition and labelling process, followed by the data preprocessing. The preprocessed data have been augmented to increase the number of the training samples. In the next step, the model is trained and optimized for the NCS 2. During the training process, we use the ADAM optimizer with the Crossentropy loss function. To optimize and convert the model, first we prune the selected filters of the classifier, then we perform the fine tuning and finally we perform the final conversion with the dedicated optimizations, i.e., weights quantization. In the second stage, the converted model is deployed on the edge device and tested.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

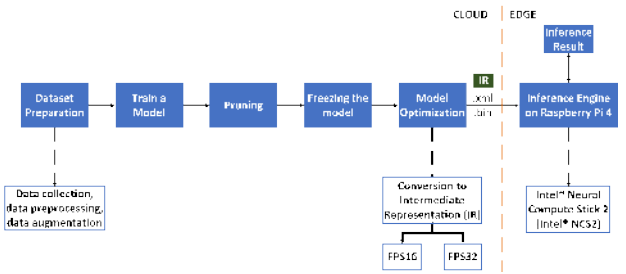

FIGURE 13. System Workflow

E. OPTIMIZATIONS

Fig. 14 describes the model optimization environment used in the system development. The system implements the data augmentation to increase the number of training samples and to improve the model performance. The system applies dedicated MKL-DNN kernels, to speed-up the calculations. To decrease the number of training parameters, the model is pruned and its training variables are converted into constants. The freezed models are converted into IR using Intel® OpenVino Toolkit for inference on the target device. The detailed optimization workflow is shown in Fig. 14. To prune the model, we calculate L1 norm of the convolutional filters. Then, the results are sorted in the ascending order and the least significant filters are removed.

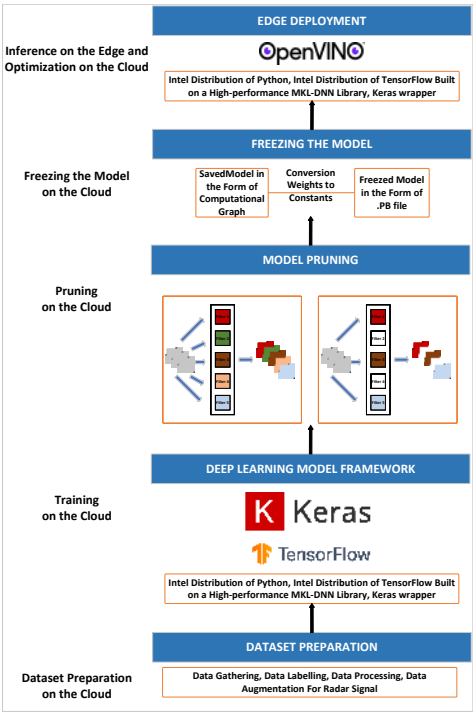

FIGURE 14. Optimization Workflow

F. PRUNING DETAILS (NO OPTIMIZATION FOR INTEL® NCS 2)

According to [54], the filters with smaller kernel weights produce feature maps with weak activations as compared to the filters with high activations. To decrease the number of training parameters, we calculated the L1 norm of each convolutional filter's weights. Subsequently, the result of an L1 norm has been plotted. Basing on those plots, we have been able to choose the filters with the least value of an L1 norm, which do not contribute a lot to the network. Fig. 15 shows an L1 norm of each of filter's weight. After experimentations, we have decided to prune the second, third, fourth and fifth convolutional layer filter. Table 2 shows the number of convolutional filters in different variants of the proposed classifier. It can be seen that the number of convolutional filters is decreased gradually. From the plot, it can be noticed that the first ten channels do not store any important information. For that reason, we cut ten filters from the second, third, fourth and fifth layer. Subsequently, we pruned 12 filters in the second layer, 15 filters in the third and fourth layer and 10 filters in the fifth layer. In the next model, we increased the number of pruned filters to 18 in the third and fourth layer and to 20 in the fifth layer. In the last model, we removed 32 filters in fourth and fifth layer.

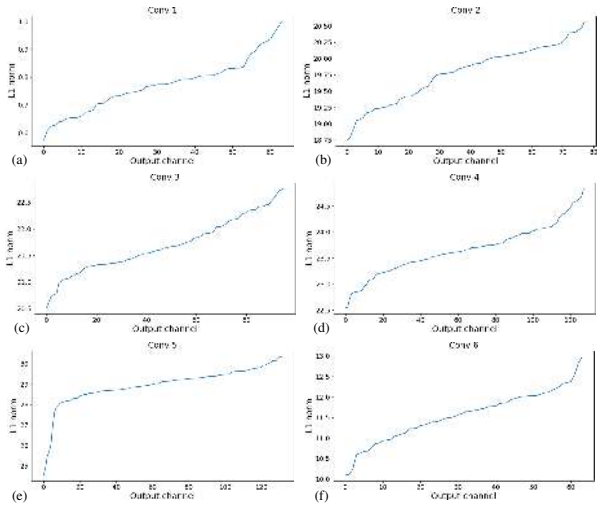

FIGURE 15. L1 norms: a) Conv1 - L1 norm, b) Conv2 - L1 norm, c) Conv3 - L1 norm, d) Conv4 - L1 norm, e) Conv5 - L1 norm, f) Conv6 - L1 norm

TABLE 2. Pruning summary (no optimization)

	No pruning	Pruning 1	Pruning 2	Pruning 3	Pruning 4
Conv1	64	64	64	64	64
Conv2	78	68	66	66	66
Conv3	96	86	81	78	78
Conv4	128	118	113	108	96
Conv5	132	122	122	112	100
Conv6	64	64	64	64	64
Model size	4.79 MB	4.42 MB	4.13 MB	3.85 MB	3.56 MB

G. EDGE ENVIRONMENT

On the edge side, the RaspberryPi 4 and NCS 2 are used. RaspbianOS has been chosen as an operating system. On

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

the software side, the TensorFlow with Keras backend is used as a base tool for inference. In order to run our model on the NCS 2, OpenVino package is used as a library for optimizing the inference process. The *BGT60TR13C* radar sensor is applied as device performing the measurements. The edge environment is illustrated in Fig. 16.

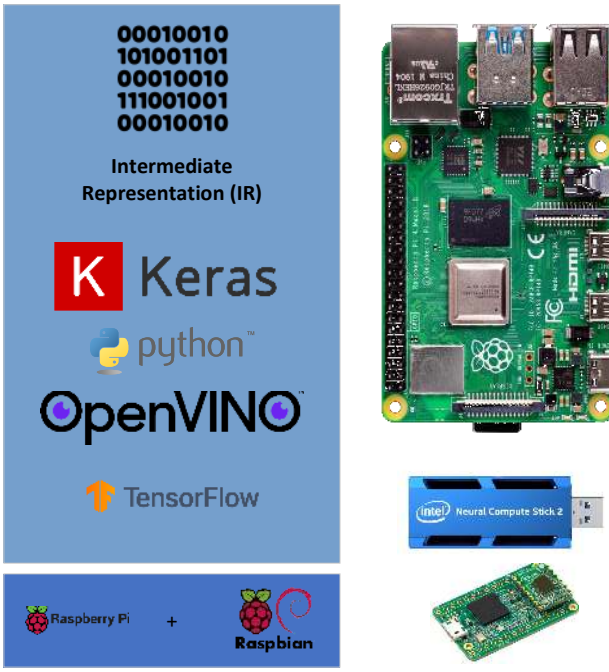

FIGURE 16. Edge Environment

VI. OPTIMIZATION AND INFERENCE ON THE EDGE

This section introduces the utilities used for model optimization. It presents the model types offered by the MO. We also discuss the inference steps.

A. MODEL OPTIMIZATION FOR THE EDGE

The system utilizes MO to convert the model to IR. MO removes from the NN graph unnecessary operations, fuses some of the mathematical operations into one node. In this project, there are two types of IR model: FPS32 and FPS16.

B. INFERENCE ON THE EDGE

Inference on the Edge creates the topology design problem, due to lack of support of all layers by the MO and limitations of hardware resources. For that reason, the data representation must meet the specific requirements imposed by the basic NN operations. The inference workflow proceeds as follows:

1) The core object is created, initialized and, the required device plugins are loaded, depending on the required device (e.g., VPU, CPU, GPU, NCS2).
2) The core object instance reads an IR file into the CNNNetwork object.
3) The input and output data format is set to be compliant with the NN topology.
4) The model is compiled, configured and loaded into the host device memory.
5) The inference request is created to allocate the buffers for the input data.
6) The input data is copied into the input data buffers.
7) The inference mode is selected (i.e., synchronous, asynchronous) and the inference request is executed.
8) The output of the inference request is read and processed.

The list of NN operations is given in the table 1 and the steps included in the inference process are described in the Fig. 17.

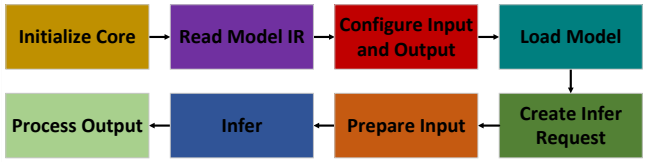

FIGURE 17. Inference steps

VII. EXPERIMENTAL RESULTS

This section discusses the experimental results. First, we analyse the 5-fold cross-validation which gives us the general information about model performance. Second, we analyse the accuracies before and after optimization. Third, we investigate the relationship between the model size, the accuracy, inference time (i.e., inference time vs model size and accuracy vs inference time) and load time. Fourth, we analyse classification report giving the detailed information about the classification metrics, i.e., accuracy, precision, F1 score, recall (model test classification metrics before and after optimization). Lastly, we study the end-to-end system latency for each kind of gesture.

A. 5-FOLD CROSS-VALIDATION

To prove the stability of our classifier, we performed the 5-fold cross-validation. Basing on 5-fold cross-validation, we are able to determine an average performance of each model. It can be seen that proposed classifier achieves significantly better performance than the Inception family models with an average accuracy of 97.57%. Proposed algorithm achieves comparable performance with MobileNetV1 models. Whereas accuracies of MobileNetV1 for decreasing values of α parameter are 98.12%, 98.08%, 96.33%, and 90.24%, proposed classifier offers comparable performance maintaining low number of training parameters, what has a direct impact on the model size. Classifiers exhibit similar behaviour in each fold. The performance of Inception family networks deteriorates with the decreasing number of Inception modules. The best performance exhibits Inception network with three Inception modules achieving an average accuracy of 88.67%. The average accuracy of Inception network with one and two inception modules is 74.65% and 78.39%, respectively. Classifier accuracies and losses for each k-fold are presented in Fig. 18 and 19.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

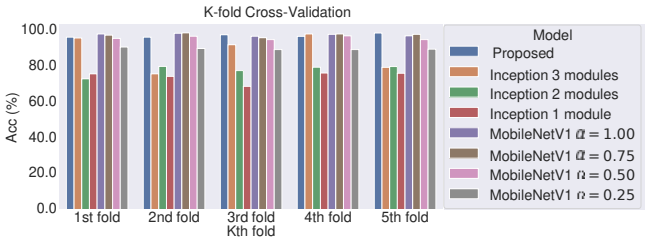

FIGURE 18. 5-fold Cross-Validation - accuracy

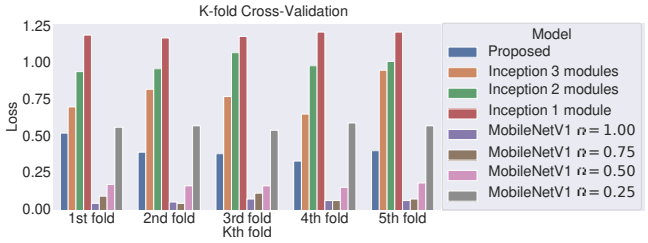

FIGURE 19. 5-fold Cross-Validation - loss

B. ACCURACY

Fig. 20-22 present accuracies for the optimized and non-optimized variants of our classifier. From the plots for non-optimized as well as optimized versions, it can be seen that classifiers exhibit the promising tendency together with decreasing number of training parameters, i.e., the test accuracy improves (Proposed topology (x86) - 97.4%, Pruned 1 (x86) - 98.1%, Pruned 2 (x86) - 98.0%, Pruned 3 (x86) - 97.7%, Pruned 4 (x86) - 97.6%).

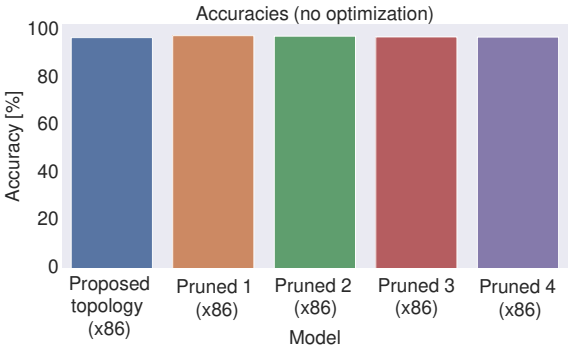

FIGURE 20. Accuracies (no optimization - x86 architecture)

From Fig. 21-22, it can be seen that the test accuracy in the case of optimized versions attains acceptable trade-off between classifier performance, model size and inference time with respect to the non-optimized versions, what has been depicted in Fig. 23-28. In case of optimized versions, the tendency is similar, namely the test accuracy slightly increases in comparison to variant which is not optimized (i.e., Proposed topology - 98.0%, Pruned 1 - 98.1%, Pruned 2 - 98.0%, Pruned 3 - 97.6%, Pruned 4 - 97.5%).

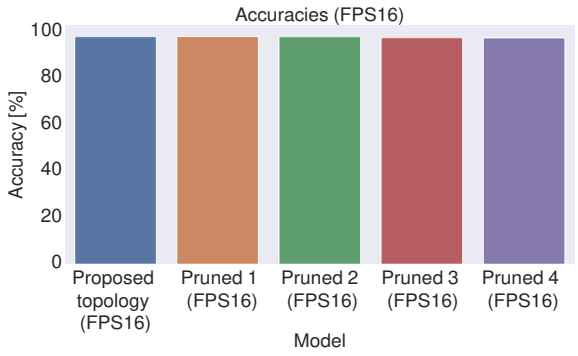

FIGURE 21. Accuracies (optimized - FPS16)

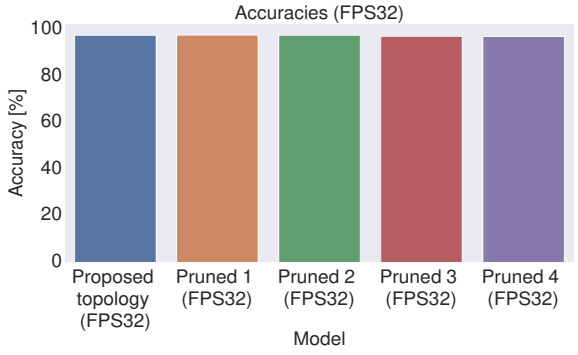

FIGURE 22. Accuracies (optimized - FPS32)

C. DETAILED MODEL PERFORMANCE

Fig. 23-25 presents the plots representing the relation between inference time and the model size. Green, orange and pink rectangles group the classifier families. Green corresponds to MobileNetV1, pink to Inception and orange to the proposed classifier.

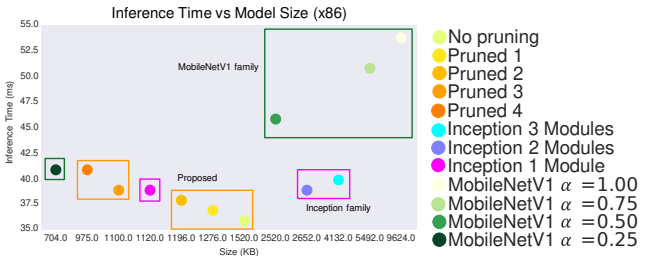

FIGURE 23. Time vs Model Size (x86)

From the plots, we can observe that optimized versions of the classifiers offer significantly shorter inference times in comparison with non-optimized versions. FPS32 version of each model is slightly bigger than the standard size versions, what is caused by MO, whereas FPS16 versions are much smaller and they preserve a good performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

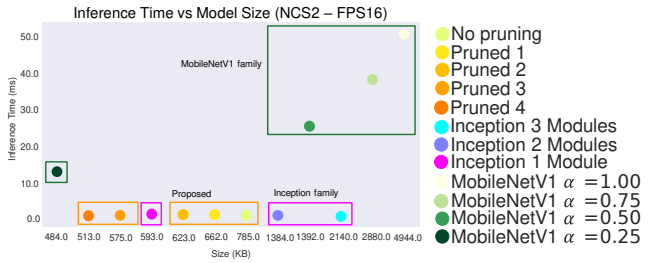

FIGURE 24. Time vs Model Size (FPS16)

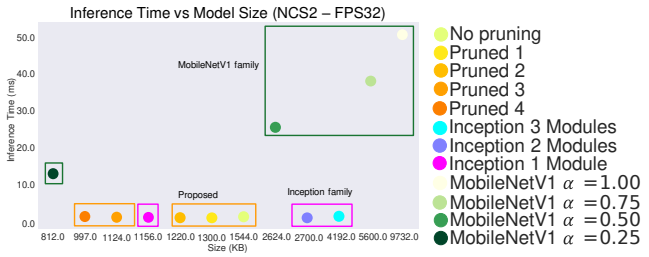

FIGURE 25. Time vs Model Size (FPS32)

Fig. 26-28 represent the dependency between the accuracy and the inference time. It can be noticed that the proposed classifier and its derivatives achieve the best trade-off between accuracy and inference time.

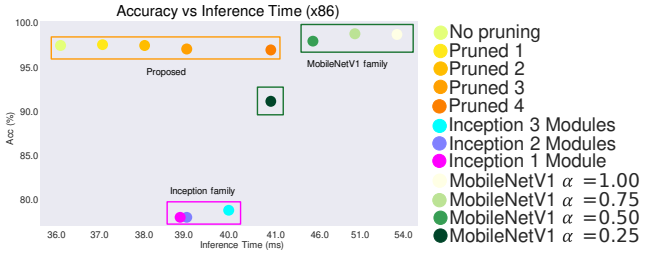

FIGURE 26. Accuracy vs Inference Time (x86)

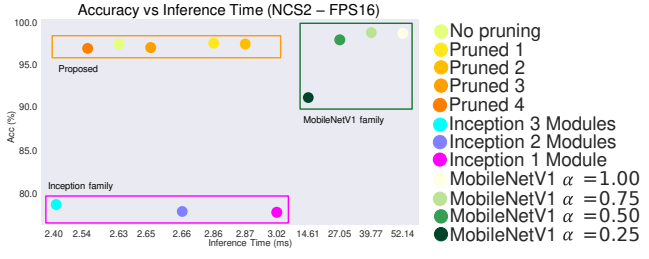

FIGURE 27. Accuracy vs Inference Time (FPS16)

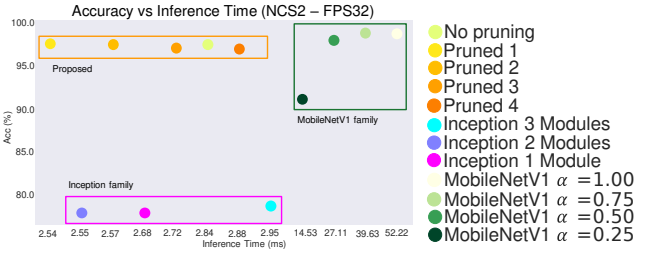

FIGURE 28. Accuracy vs Inference Time (FPS32)

Fig. 29-31 represent model loading times. As it can be seen, loading times in the case of optimized versions are much shorter than in the case of non-optimized versions. For the proposed classifier (non-optimized version), loading times vary from 7.21 s to 8.86 s, what makes them comparable with loading times of MobileNetV1 family models and Inception family models.

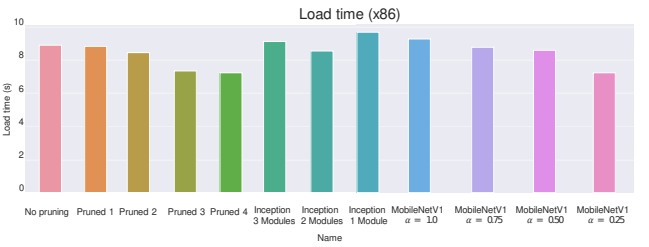

FIGURE 29. Loading time (x86)

In the case of the optimized versions, loading times exhibit similar behaviour, however they are much shorter than in the non-optimized versions.

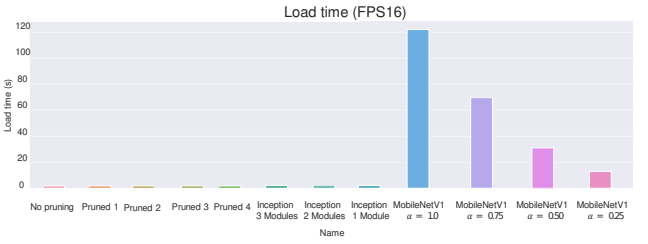

FIGURE 30. Loading time (FPS16)

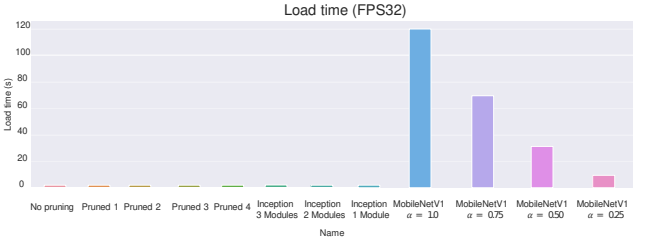

FIGURE 31. Loading time (FPS32)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

D. CLASSIFICATION REPORT

Tables 3-6 present the classification report representing the classification metrics (i.e., Accuracy, Precision, Recall and F1 Score) for each classifier.

TABLE 3. Accuracy - all topologies. N: no pruning, F16: NCS2 FPS16, F32: NCS2 FPS32, B: NCS2 FPS16/FPS32

	Accuracy
Proposed topology (x86)	0.974
Pruned 1 (x86)	0.982
Pruned 2 (x86)	0.980
Pruned 3 (x86)	0.977
Pruned 4 (x86)	0.976
Inception Net with 1 module (x86)	0.780
Inception Net with 2 modules (x86)	0.786
Inception Net with 3 modules (x86)	0.792
MobNetV1 $\alpha = 0.25$ (x86)	0.916
MobNetV1 $\alpha = 0.50$ (x86)	0.985
MobNetV1 $\alpha = 0.75$ (x86)	0.994
MobNetV1 $\alpha = 1.00$ (x86)	0.993
Proposed topology (NB)	**0.980**
Pruned 1 (B)	**0.981**
Pruned 2 (B)	**0.980**
Pruned 3 (B)	**0.976**
Pruned 4 (B)	**0.975**
Inception Net with 1 module (F16)	**0.785**
Inception Net with 1 module (F32)	**0.784**
Inception Net with 2 modules (B)	**0.785**
Inception Net with 3 modules (B)	**0.793**
MobNetV1 $\alpha = 0.25$ (F16)	**0.918**
MobNetV1 $\alpha = 0.25$ (F32)	**0.917**
MobNetV1 $\alpha = 0.50$ (F16)	**0.986**
MobNetV1 $\alpha = 0.50$ (F32)	**0.985**
MobNetV1 $\alpha = 0.75$ (B)	**0.993**
MobNetV1 $\alpha = 1.00$ (B)	**0.993**

TABLE 4. Precision - all topologies. N: no pruning, F16: NCS2 FPS16, F32: NCS2 FPS32, B: NCS2 FPS16/FPS32

	down_up	left_right	rubbing	up_down
Proposed topology (x86)	0.973	0.968	0.975	0.980
Pruned 1 (x86)	0.983	0.989	0.976	0.981
Pruned 2 (x86)	0.983	0.989	0.969	0.981
Pruned 3 (x86)	0.983	0.979	0.962	0.985
Pruned 4 (x86)	0.986	0.962	0.969	0.990
Inception Net with 1 module (x86)	0.927	0.575	0.940	0.000
Inception Net with 2 modules (x86)	0.939	0.754	0.696	1.000
Inception Net with 3 modules (x86)	0.890	0.730	0.751	1.000
MobNetV1 $\alpha = 0.25$ (x86)	0.935	0.982	0.911	0.837
MobNetV1 $\alpha = 0.50$ (x86)	0.980	0.996	0.986	0.976
MobNetV1 $\alpha = 0.75$ (x86)	0.997	0.996	0.990	0.995
MobNetV1 $\alpha = 1.00$ (x86)	0.997	0.996	0.986	0.990
Proposed topology (NB)	**0.979**	**0.986**	**0.975**	**0.980**
Pruned 1 (B)	**0.982**	**0.989**	**0.972**	**0.980**
Pruned 2 (B)	**0.982**	**0.989**	**0.968**	**0.980**
Pruned 3 (B)	**0.982**	**0.978**	**0.962**	**0.985**
Pruned 4 (B)	**0.986**	**0.962**	**0.968**	**0.989**
Inception with 1 module (F16)	**0.986**	**0.635**	**0.798**	**0.000**
Inception with 1 module (F32)	**0.983**	**0.633**	**0.800**	**0.000**
Inception with 2 modules (B)	**0.938**	**0.754**	**0.695**	**1.000**
Inception with 3 modules (B)	**0.892**	**0.731**	**0.750**	**1.000**
MobNetV1 $\alpha = 0.25$ (F16)	**0.939**	**0.982**	**0.911**	**0.839**
MobNetV1 $\alpha = 0.25$ (F32)	**0.939**	**0.982**	**0.911**	**0.835**
MobNetV1 $\alpha = 0.50$ (F16)	**0.980**	**1.000**	**0.986**	**0.976**
MobNetV1 $\alpha = 0.50$ (F32)	**0.980**	**1.000**	**0.986**	**0.972**
MobNetV1 $\alpha = 0.75$ (B)	**0.997**	**0.996**	**0.986**	**0.995**
MobNetV1 $\alpha = 1.00$ (B)	**1.000**	**0.996**	**0.983**	**0.990**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

TABLE 5. Recall - all topologies. N: no pruning, F16: NCS2 FPS16, F32: NCS2 FPS32, B: NCS2 FPS16/FPS32

	down_up	left_right	rubbing	up_down
Proposed topology (x86)	0.993	0.955	0.972	0.976
Pruned 1 (x86)	0.990	0.955	0.989	1.000
Pruned 2 (x86)	0.990	0.955	0.989	0.990
Pruned 3 (x86)	0.983	0.958	0.989	0.976
Pruned 4 (x86)	0.986	0.969	0.986	0.956
Inception Net with 1 module (x86)	0.997	0.913	0.986	0.000
Inception Net with 2 modules (x86)	0.997	0.927	0.989	0.005
Inception Net with 3 modules (x86)	0.997	0.875	0.986	0.117
MobNetV1 $\alpha = 0.25$ (x86)	0.993	0.743	0.968	0.976
MobNetV1 $\alpha = 0.50$ (x86)	0.997	0.955	0.993	1.000
MobNetV1 $\alpha = 0.75$ (x86)	0.997	0.986	0.996	1.000
MobNetV1 $\alpha = 1.00$ (x86)	0.997	0.983	0.996	0.995
Proposed topology (NB)	**0.989**	**0.951**	**0.989**	**0.995**
Pruned 1 (B)	**0.989**	**0.954**	**0.989**	**0.995**
Pruned 2 (B)	**0.989**	**0.954**	**0.989**	**0.990**
Pruned 3 (B)	**0.982**	**0.958**	**0.989**	**0.975**
Pruned 4 (B)	**0.986**	**0.968**	**0.985**	**0.956**
Inception Net with 1 module (F16)	**0.996**	**0.930**	**0.989**	**0.000**
Inception Net with 1 module (F32)	**0.993**	**0.930**	**0.989**	**0.000**
Inception Net with 2 modules (B)	**0.996**	**0.927**	**0.989**	**0.004**
Inception Net with 3 modules (B)	**0.996**	**0.878**	**0.985**	**0.117**
MobNetV1 $\alpha = 0.25$ (F16)	**0.993**	**0.747**	**0.979**	**0.966**
MobNetV1 $\alpha = 0.25$ (F32)	**0.993**	**0.743**	**0.979**	**0.966**
MobNetV1 $\alpha = 0.50$ (F16)	**0.997**	**0.955**	**0.996**	**1.000**
MobNetV1 $\alpha = 0.50$ (F32)	**0.997**	**0.951**	**0.996**	**1.000**
MobNetV1 $\alpha = 0.75$ (B)	**0.997**	**0.983**	**0.996**	**1.000**
MobNetV1 $\alpha = 1.00$ (B)	**0.997**	**0.983**	**0.996**	**0.995**

TABLE 6. F1 score - all topologies. N: no pruning, F16: NCS2 FPS16, F32: NCS2 FPS32, B: NCS2 FPS16/FPS32

	down_up	left_right	rubbing	up_down
Proposed topology (x86)	0.983	0.962	0.974	0.978
Pruned 1 (x86)	0.986	0.972	0.983	0.990
Pruned 2 (x86)	0.986	0.972	0.979	0.985
Pruned 4 (x86)	0.986	0.965	0.977	0.973
Pruned 4 (x86)	0.986	0.965	0.977	0.973
Inception Net with 1 module (x86)	0.960	0.706	0.962	0.000
Inception Net with 2 modules (x86)	0.967	0.832	0.817	0.010
Inception Net with 3 modules (x86)	0.940	0.796	0.852	0.210
MobNetV1 $\alpha = 0.25$ (x86)	0.963	0.846	0.939	0.901
MobNetV1 $\alpha = 0.50$ (x86)	0.988	0.975	0.989	0.988
MobNetV1 $\alpha = 0.75$ (x86)	0.997	0.991	0.993	0.998
MobNetV1 $\alpha = 1.00$ (x86)	0.997	0.990	0.991	0.993
Proposed topology (NB)	**0.985**	**0.968**	**0.982**	**0.987**
Pruned 1 (B)	**0.986**	**0.971**	**0.980**	**0.987**
Pruned 2 (B)	**0.986**	**0.971**	**0.979**	**0.985**
Pruned 3 (B)	**0.982**	**0.968**	**0.975**	**0.980**
Pruned 4 (B)	**0.986**	**0.965**	**0.977**	**0.972**
Inception Net with 1 module (F16)	**0.991**	**0.754**	**0.883**	**0.000**
Inception Net with 1 module (F32)	**0.988**	**0.753**	**0.885**	**0.000**
Inception Net with 2 modules (B)	**0.966**	**0.831**	**0.816**	**0.009**
Inception Net with 3 modules (B)	**0.941**	**0.798**	**0.852**	**0.209**
MobNetV1 $\alpha = 0.25$ (F16)	**0.965**	**0.848**	**0.944**	**0.898**
MobNetV1 $\alpha = 0.25$ (F32)	**0.965**	**0.846**	**0.944**	**0.896**
MobNetV1 $\alpha = 0.50$ (F16)	**0.988**	**0.977**	**0.991**	**0.988**
MobNetV1 $\alpha = 0.50$ (F32)	**0.988**	**0.975**	**0.991**	**0.986**
MobNetV1 $\alpha = 0.75$ (B)	**0.997**	**0.990**	**0.991**	**0.998**
MobNetV1 $\alpha = 1.00$ (B)	**0.998**	**0.990**	**0.990**	**0.993**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

E. STUDY OF THE END-TO-END SYSTEM LATENCY

This section investigates the end-to-end system latency. For each kind of gesture, we measured the duration of the gesture sample - t_s, data processing time - t_p, inference time - t_i and we summed those times, obtaining the total time - T_t. The measurements have been repeated 10 times for each gesture type then the average times have been calculated. Depending on the gesture type, the duration of the gesture sample is different. We assume that the gesture can maximally last 3.2 s, i.e., in case of shorter sample duration, it is zero padded. The measurements have been performed for the proposed topology (non-pruned version - FPS16) deployed on the NCS 2. The tables 7-10 present the end-to-end latency for each kind of gesture.

TABLE 7. Down-up gesture - end-to-end latency

	t_s [s]	t_p [s]	t_i [s]	T_t [s]
	0.752	0.757	0.004	1.514
	0.951	0.925	0.003	1.879
	0.952	0.913	0.003	1.867
	0.852	0.809	0.002	1.663
	0.952	0.909	0.003	1.863
	0.852	0.834	0.003	1.688
	0.751	0.732	0.003	1.487
	0.701	0.679	0.002	1.382
	0.852	0.820	0.003	1.675
	0.801	0.774	0.003	1.578
Average time	**0.842**	**0.815**	**0.003**	**1.660**

TABLE 8. Left-right gesture - end-to-end latency

	t_s [s]	t_p [s]	t_i [s]	T_t [s]
	0.301	0.311	0.004	0.616
	0.301	0.321	0.004	0.626
	0.300	0.314	0.003	0.616
	0.400	0.391	0.002	0.793
	0.451	0.456	0.003	0.910
	0.401	0.395	0.003	0.799
	0.451	0.430	0.002	0.883
	0.050	0.07465	0.003	0.126
	0.318	0.341	0.003	0.662
	0.350	0.353	0.003	0.706
Average time	**0.332**	**0.339**	**0.003**	**0.674**

TABLE 9. Rubbing gesture - end-to-end latency

	t_s [s]	t_p [s]	t_i [s]	T_t [s]
	1.153	1.221	0.004	2.378
	1.203	1.158	0.003	2.364
	1.102	1.047	0.002	2.152
	1.252	1.182	0.002	2.437
	1.302	1.270	0.003	2.575
	1.203	1.144	0.003	2.350
	1.252	1.196	0.002	2.451
	1.252	1.201	0.003	2.456
	1.252	1.206	0.003	2.461
	1.553	1.538	0.003	3.094
Average time	**1.252**	**1.216**	**0.003**	**2.472**

TABLE 10. Up-down gesture - end-to-end latency

	t_s [s]	t_p [s]	t_i [s]	T_t [s]
	0.552	0.620	0.005	1.177
	0.701	0.687	0.003	1.391
	0.551	0.557	0.003	1.110
	0.501	0.503	0.003	1.007
	0.551	0.522	0.002	1.076
	0.050	0.071	0.003	0.124
	0.471	0.492	0.003	0.966
	0.551	0.531	0.003	1.085
	0.601	0.600	0.003	1.204
	0.500	0.491	0.003	0.994
Average time	**0.503**	**0.507**	**0.003**	**1.013**

VIII. SUMMARY

This paper demonstrates an optimized radar gesture classification model with the dedicated signal processing scheme. The conducted experiments covered training process on the cloud, pruning, optimization for the edge and inference. The results of the experiments exhibit the significant improvements in the widely understood model performance. We examined the model performance before and after optimization (i.e., 5-fold cross-validation, test accuracy, classification report, relations between accuracy, model size, inference time and loading time). From the accuracy analysis and classification report, it can be noticed that the pruned versions (x86) of our classifier exhibit a better performance than the non-pruned version. With regards to classification performance of all classifiers, classification report shows that the inference results are comparable with FPS16 and FPS32 versions. Whereas the classification performance of all classifiers (both optimized as well as non-optimized alternatives) remain on the similar level, the more detailed

analysis of model sizes, inference times and loading times allows to observe the benefits of pruning and optimization for the Intel® NCS 2. In terms of accuracy and model size, we can observe that each alternative (x86, FPS16 and FPS32) of our classifier preserves the most beneficial relation between accuracy and the model size. The optimized version of the proposed classifier achieves in the worst case 97.50% accuracy, it provides the most beneficial relation between the model performance and the model size. Especially in the case of FPS16 version. Another aspect worth of consideration is the relationship between the inference time and the model size. This relationship has been depicted in Fig. 23-25. Comparing the relationship between the inference time and the model size, we can notice the significant improvement of each variant of the proposed classifier. Another significant issue from the deployment on the edge perspective is the dependency between the accuracy and the inference time. This relationship has been presented in the Fig. 26-28. As we can see, the optimized versions of the classifiers exhibit the tendency in the direction of decreasing inference time and preserving the good accuracy, what is particularly important in the domain of model execution on the edge and providing the real time inference time. The next examined issue are the model loading times, which have been depicted in Fig. 29-31. In the case of non-optimized classifiers, model loading times vary from 7.21 s to 9.65 s. In the case of optimized versions of the classifiers, the optimization results are particularly visible regarding the proposed classifier and Inception family models. The loading times of MobileNet family classifiers are significantly worse in comparison to the proposed and Inception family models. The last considered aspect is the end-to-end latency of the system. The tables 7-10 list end-to-end latencies measured for each kind of gesture. It can be noticed that the average times for down-up, left-right, rubbing and up-down gestures are 1.660 s, 0.674 s, 2.472 s, 1.013 s, respectively. Those results prove the real-time operation of our system.

To sum up, we proposed the optimized radar gesture classifier with the dedicated radar signal processing scheme allowing for deployment on the edge device. To the best of our knowledge, we have proposed the first radar gesture classification model which has been deployed on the edge device such as NCS 2. The proposed solution has been compared with the InceptionV1 and MobileNetV1 family models in various variants. All topologies have been assessed in terms of various aspects, i.e., classification performance, inference times, model sizes. The proposed models (i.e., No Pruning, Pruned 1, Pruned 2, Pruned 3, Pruned 4) achieve the best results in terms of model sizes and inference times. In terms of accuracy, our classifier achieves better results than in case of the classifiers with 3 Inception modules, 2 Inception modules, 1 Inception module, and MobileNetV1 with $\alpha = 0.25$. MobileNets with $\alpha = 0.50$, $\alpha = 0.75$ and $\alpha = 1.00$ offer slightly better classification results, at the same time offering bigger model size and longer inference time, what is of great significance in the case of the deployment on the edge. In the future, we are going to increase the number of recognized gestures and implement our solution on the other accelerators (i.e., NVIDIA Jetson Nano, RockPi, Edge TPU). Moreover, we will also extend our optimization ideas and computing approaches to process full 3D data [55]–[57].

REFERENCES

[1] A. Shehab and S. Al-Janabi, "Edge Computing: Review and Future Directions (Computación de Borde: Revisión y Direcciones Futuras)", REVISTA AUS Journal, no. 26-2, pp. 368-380, 2019. [Accessed 15 December 2020].

[2] F. Alemuda and F. J. Lin, "Gesture-Based Control in a Smart Home Environment," 2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, 2017, pp. 784-791, DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.120.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1738-1762, Aug. 2019, DOI: 10.1109/JPROC.2019.2918951.

[4] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar and A. Y. Zomaya, "Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence," in IEEE Internet of Things Journal, vol. 7, no. 8, pp. 7457-7469, Aug. 2020, DOI: 10.1109/JIOT.2020.2984887.

[5] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, DOI: 10.1109/JPROC.2017.2761740.

[6] "ImageNet classification with deep convolutional neural networks | Communications of the ACM", Dl.acm.org, 2020. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386. [Accessed: 15- Dec- 2020].

[7] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: 15- Dec- 2020].

[8] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[9] "Improving TensorFlow* Inference Performance on Intel® Xeon® Processors", Intel. [Online]. Available: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/improving-tensorflow-inference-performance-on-intel-xeon-processors.html. [Accessed: 18- Dec- 2020].

[10] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/1704.04861. [Accessed: 15- Dec- 2020].

[11] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/1602.07360. [Accessed: 15- Dec- 2020].

[12] M. Tan and Q. Le, "MixConv: Mixed Depthwise Convolutional Kernels", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1907.09595. [Accessed: 16- Dec- 2020].

[13] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1512.00567. [Accessed: 15- Dec- 2020].

[15] T. Elsken, J. Metzen and F. Hutter, "Neural Architecture Search: A Survey", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1808.05377. [Accessed: 18- Dec- 2020].

[16] P. Ren et al., "A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2006.02903. [Accessed: 18- Dec- 2020].

[17] "OpenVINO™ Toolkit Overview - OpenVINO™ Toolkit", Docs.openvinotoolkit.org, 2020. [Online]. Available: https://docs.openvinotoolkit.org/latest/index.html. [Accessed: 14- Dec- 2020].

[18] M. Abadi et al., "TensorFlow: A system for large-scale machine learning", arXiv.org, 2016. [Online]. Available: https://arxiv.org/abs/1605.08695. [Accessed: 14- Dec- 2020].

[19] S. Sun, Z. Cao, H. Zhu and J. Zhao, "A Survey of Optimization Methods From a Machine Learning Perspective", 2019. .

[20] R. Sun, "Optimization for deep learning: theory and algorithms", NASA/ADS, 2019. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2019arXiv191208957S/abstract. [Accessed: 27- Dec- 2020].

[21] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1655-1674, Aug. 2019, DOI: 10.1109/JPROC.2019.2921977.

[22] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan and X. Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," in IEEE Communications Surveys and Tutorials, vol. 22, no. 2, pp. 869-904, Secondquarter 2020, DOI: 10.1109/COMST.2020.2970550.

[23] "Overview of ML Pipelines | Testing and Debugging in Machine Learning", Google Developers. [Online]. Available: https://developers.google.com/machine-learning/testing-debugging/pipeline/overview. [Accessed: 16- Dec- 2020].

[24] "What is an ML pipeline and why is it important? | Algorithmia Blog", Algorithmia Blog, 2020. [Online]. Available: https://algorithmia.com/blog/ml-pipeline\#:~:text=One\%20definition\%20of\%20a\%20machine,the\%20ML\%20model\%20fully\%20automated. [Accessed: 16- Dec- 2020].

[25] A. Bernstein and A. Kuleshov, "Low-Dimensional Data Representation in Data Analysis",fv 2014.

[26] M. Alom et al., "A State-of-the-Art Survey on Deep Learning Theory and Architectures", 2019. .

[27] Y. Roh, G. Heo and S. E. Whang, "A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective," in IEEE Transactions on Knowledge and Data Engineering, DOI: 10.1109/TKDE.2019.2946162.

[28] T. Yu and H. Zhu, "Hyper-Parameter Optimization: A Review of Algorithms and Applications", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2003.05689. [Accessed: 27- Dec- 2020].

[29] J. S. Sonkusare, N. B. Chopade, R. Sor and S. L. Tade, "A Review on Hand Gesture Recognition System," 2015 International Conference on Computing Communication Control and Automation, Pune, 2015, pp. 790-794, DOI: 10.1109/ICCUBEA.2015.158.

[30] M. Oudah, A. Al-Naji, and J. Chahl, "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques," Journal of Imaging, vol. 6, no. 8, p. 73, Jul. 2020.

[31] M. Yasen and S. Jusoh, "A systematic review on hand gesture recognition techniques, challenges and applications", 2019.

[32] S. Hazra and A. Santra, "Short-Range Radar-Based Gesture Recognition System Using 3D CNN With Triplet Loss," in IEEE Access, vol. 7, pp. 125623-125633, 2019, DOI: 10.1109/ACCESS.2019.2938725.

[33] Z. Zhang, Z. Tian and M. Zhou, "Latern: Dynamic Continuous Hand Gesture Recognition Using FMCW Radar Sensor," in IEEE Sensors Journal, vol. 18, no. 8, pp. 3278-3289, 15 April15, 2018, doi: 10.1109/JSEN.2018.2808688.

[34] "Exploration of task-based scheduling for convolutional neural networks accelerators under memory constraints | Proceedings of the 16th ACM International Conference on Computing Frontiers", Dl.acm.org, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3310273.3323162. [Accessed: 16- Dec- 2020].

[35] Z. Li, Y. Wang, T. Zhi and T. Chen, "A survey of neural network accelerators", 2017.

[36] D. Xu et al., "Resilient Neural Network Training for Accelerators with Computing Errors," 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 2019, pp. 99-102, DOI: 10.1109/ASAP.2019.00-23.

[37] Y. Chen, Y. Xie, L. Song, F. Chen and T. Tang, "A Survey of Accelerator Architectures for Deep Neural Networks", 2020.

[38] "Supported Framework Layers - OpenVINO™ Toolkit", Docs.openvinotoolkit.org. [Online]. Available: https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_prepare_model_Supported_Frameworks_Layers.html#tensorflow_supported_operations. [Accessed: 16- Dec- 2020].

[39] Infineon Technologies AG Internal Technical Documentation, Munich, 2019.

[40] "Overview of OpenVINO™ Toolkit Public Models - OpenVINO™ Toolkit", Docs.openvinotoolkit.org. [Online]. Available: https://docs.openvinotoolkit.org/latest/omz_models_public_index.html. [Accessed: 27- Dec- 2020].

[41] H. Li, A. Kadav, I. Durdanovic, H. Samet and H. Graf, "Pruning Filters for Efficient ConvNets", arXiv.org, 2017. [Online]. Available: https://arxiv.org/abs/1608.08710. [Accessed: 16- Dec- 2020].

[42] F. Chollet and others, "Keras", Keras.io, 2015. [Online]. Available: https://keras.io. [Accessed: 15- Dec- 2020].

[43] "Model Optimizer Developer Guide - OpenVINO™ Toolkit", Docs.openvinotoolkit.org. [Online]. Available: https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html. [Accessed: 27- Dec- 2020].

[44] Lien, J., Gillian, N., Karagozler, M., Amihood, P., Schwesig, C., Olsen, E., Raja, H. and Poupyrev, I., 2016. Soli: Ubiquitous Gesture Sensing With Millimeter Wave Radar: ACM Transactions On Graphics: Vol 35, No 4. [online] Dl.acm.org. Available at: <https://dl.acm.org/doi/10.1145/2897824.2925953> [Accessed 17 January 2021].

[45] P. Molchanov, S. Gupta, K. Kim and K. Pulli, "Short-range FMCW monopulse radar for hand-gesture sensing," 2015 IEEE Radar Conference (RadarCon), Arlington, VA, 2015, pp. 1491-1496, doi: 10.1109/RADAR.2015.7131232.

[46] P. Molchanov, S. Gupta, K. Kim and K. Pulli, "Multi-sensor system for driver's hand-gesture recognition," 2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG), Ljubljana, 2015, pp. 1-8, doi: 10.1109/FG.2015.7163132.

[47] S. Hazra and A. Santra, "Robust Gesture Recognition Using Millimetric-Wave Radar System," in IEEE Sensors Letters, vol. 2, no. 4, pp. 1-4, Dec. 2018, Art no. 7001804, DOI: 10.1109/LSENS.2018.2882642.

[48] A. Santra and S. Hazra, Deep learning applications of short-range radars. ARTECH HOUSE Incorporated, 2020.

[49] S. Ahmed and S. H. Cho, "Hand Gesture Recognition Using an IR-UWB Radar with an Inception Module-Based Classifier," Sensors, vol. 20, no. 2, p. 564, Jan. 2020.

[50] "Intel® Optimization for TensorFlow* Installation Guide", Intel. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-optimization-for-tensorflow-installation-guide.html. [Accessed: 18- Dec- 2020].

[51] "Intel® Math Kernel Library for Deep Learning Networks: Part...", Intel. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-mkl-dnn-part-1-library-overview-and-installation.html. [Accessed: 18- Dec- 2020].

[52] J. Lin, Y. Li, W. Hsu and T. Lee, "Design of an FMCW radar baseband signal processing system for automotive application", 2016. [Online]. Available: https://link.springer.com/content/pdf/10.1186/s40064-015-1583-5.pdf. [Accessed: 30- Mar- 2021]

[53] Introduction to TensorFlow with Intel Optimizations. 2018 [Online]. Available: https://indico.cern.ch/event/762142/sessions/290684/attachments/1752969/2841010/Tensorflow.pdf. [Accessed: 02- Mar- 2021]

[54] H. Li, A. Kadav, I. Durdanovic, H. Samet and H. Graf, "Pruning Filters for Efficient ConvNets", arXiv.org, 2017. [Online]. Available: https://arxiv.org/abs/1608.08710v3. [Accessed: 02- Mar- 2021]

[55] Y. Liang, F. He, and X. Zeng, "3D mesh simplification with feature preservation based on Whale Optimization Algorithm and Differential Evolution," Integrated Computer-Aided Engineering, vol. 27, no. 4, pp. 417–435, 2020

[56] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "MeshCNN," ACM Transactions on Graphics, vol. 38, no. 4, pp. 1–12, 2019

[57] Y. Wu, F. He, D. Zhang and X. Li, "Service-Oriented Feature-Based Data Exchange for Cloud-Based Design and Manufacturing," in IEEE Transactions on Services Computing, vol. 11, no. 2, pp. 341-353, 1 March-April 2018, doi: 10.1109/TSC.2015.2501981

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3081353, IEEE Access

MATEUSZ CHMURSKI received the B.Sc. and M.Sc. degrees in computer science from Technical University of Lodz (TUL), Lodz, Poland, in 2018. From 2015 to 2018, he was a Research Assistant in the laboratory of Infineon Technologies AG, pursuing his bachelor and master degrees. Since 2018, he has been employed in the headquarter of Infineon Technologies AG, Munich, Germany, pursing his PhD degree in Artificial Intelligence on the Edge.

MARIUSZ ZUBERT (M'12) received the Ph.D. degree in electronic from the Technical University of Lodz (TUL), Lodz, Poland, in 1999. Then, he received the D.Sc. degree in computer science from the Silesian University of Technology (SUT), Gliwice, Poland, in 2011. Since 1999, he has been employed at TUL, where he is a university professor. He is an author or co-author of over 100 publications. His interests include heat transfer problems; VLSI, MEMS/MOEMS and nano technologies; the multi-domain modeling and simulation of ASIC and SiC PiN Schottky Diodes; the design and modeling of ASICs for mobile industry; the real-time monitoring system of high voltage power lines for Ontario Hydro, Kinectrics Inc. for New York City and Ontario (grant NATO); Modeling of Electromagnetic Interactions in Modern (More-Than-Moore) 3-D Integrated Semiconductor Structures; The image processing and diagnosis of neurodegenerative diseases (e.g., BSE – Mad cow disease, Alzheimer, etc); the 3D ultrastructural amyloid plaque reconstruction and proliferation model using Gaussian Hidden Markov Random Fields; the biometric identification of people using the iris pattern, the automatic translation of multi-physical problems described by PDE/DAEs to Hardware Description Languages (VHDL-AMS, HDL-A, etc) as-well-as the complex interdisciplinary research including informatics, electronic, higher mathematics, physics as-well-as health informatics and biometrics.

KAY BIERZYNSKI received M.Sc. degree in computer science from the Technical University of Dresden. Then, in December 2015, he joined Infineon Technologies AG as a PhD candidate and worked on artificial intelligence at the network edge. Since December 2018 he works as a technical project lead in Infineon Technologies AG and is responsible for the technical management of funding projects in the area of machine learning.

AVIK SANTRA (S'09-M'10-SM'18) received his M.S. degree in Signal Processing with first class distinction from Indian Institute of Science, Bangalore in 2010. He is currently leading signal processing and deep learning algorithm/solutions research & development for radar and depth sensors processing for human-machine interface applications at Infineon, Neubiberg. Earlier in his career, he has worked as system engineer for LTE/4G modem at Broadcom Communications, and also has worked as research engineer developing cognitive radars at Airbus. He is a reviewer at various IEEE and Elsevier journals, and is recipient of several outstanding reviewer awards. He is author of the book titled 'Deep Learning Applications of Short-Range radars', published by ArTech House. He has filed over 50 patents and published more than 35 research papers related to various topics of radar waveform design, radar signal processing and radar machine/deep learning topics.