LOCAL PROCESSING SYSTEMS FOR MACHINE LEARNING ALGORITHMS

Ph.D. Thesis – Abstract

for obtaining the scientific title of doctor at

Politehnica University Timișoara

in the doctoral field Electronic Engineering, Telecommunications and Information Technology

Author Eng. Ioan LUCAN-ORĂȘAN

scientific coordinator Prof. Univ. Dr. Habil. Eng. Cătălin Daniel CĂLEANU

September 2024

Artificial intelligence is an extensive field of research developed in recent years, and today it is increasingly present for different real-world applications. This means that the field has reached a sufficiently advanced stage that it has moved from the level of research to real use cases, in a safe manner. As artificial intelligence algorithms have reached an acceptable level of maturity, the focus has also begun to exist on hardware devices with their execution capabilities. The PhD thesis presents the results of using basic technical solutions for deep neural network compression in order to obtain a neural network model compatible with embedded local systems using ARM Cortex-M CPU based microcontrollers.

The general objective of the thesis is represented by the compression of convolutional neural networks for automotive applications, such as for example the estimation of the driver's eye gaze. This is of particular importance due to the large size of state-of-the-art networks, which are usually run on high-performance PC workstations or remote servers. Such a solution is not suitable for applications that need to run in real-time, with low power consumption and where remote data transmission must be avoided for privacy and security reasons. Therefore, neural network compression plays an important role, with the aim of reducing the size of a model so that it is possible to execute inference on local real-time processing devices with low power consumption.

The fundamental contribution of this thesis is the study, description and implementation of the main methods of convolutional neural networks compression together with their validation using different reporting metrics on 32-bit STM32 series microcontrollers with ARM Cortex-M CPUs. In particular, the knowledge distillation method is extensively explored using layerwise or widthwise compression, as well as combining them with a detailed presentation of the results obtained.

The thesis is organized into six chapters and several subchapters, as shown in Figure 1. The first three chapters refer to the introductory part and general theoretical notions, and the last three chapters present the experiments with the essential contributions and results of this work.

Chapter 1 presents the motivation for the elaboration of this paper, the objectives initially set and a brief high-level presentation of the paper structure.

Chapter 2 aims to introduce in the form of general notions some theoretical information referring to the most common deep neural network architectures. Subsequently, emphasis is placed on the complexity of the architectures described in the context of running them on hardware devices with limited resources, which leads to the need to define deep neural network architectures for such devices. Consequently, a brief description of these specific architectures

follows along with a series of conclusions.

Chapter 3 presents the possibilities of executing the inference process for deep neural network models, with the aim of highlighting the advantages and necessity of running inference on local devices compared to remote computing systems (in the cloud).

Chapter 4 is the first chapter that makes original contributions to this work. It presents a number of embedded devices suitable for running the deep neural network model inference process along with the most common software and libraries that help convert models into a format compatible with embedded devices. Subsequently, a description of the embedded devices with ARM Cortex-M CPUs used in this paper follows along with their main advantages. The last part of this chapter comprises a review of 13 papers published in the last 6 years that use deep neural networks for different applications with ARM Cortex-M CPU-based microcontrollers. Finally, important results and conclusions are presented on different aspects, such as: the architectures used, the accuracy obtained, useful hardware characteristics, challenges and research opportunities.

Chapter 5 presents in the first part the theoretical notions regarding the most common methods of deep neural networks compression: quantization of parameters after or aware of training process, weights pruning or the elimination of other structural elements and knowledge distillation. In addition, the theoretical concepts for optimizing models according to a certain hardware platform and the automatic search for an optimal architecture are also briefly presented. The chapter ends with a detailed presentation of the important results and conclusions obtained from the publication of three scientific papers on post-training quantization, magnitude-based weights pruning and knowledge distillation.

The last chapter concisely presents the main results and personal contributions of the author with reference to their dissemination through scientific publications. It concludes with the presentation of future research directions.

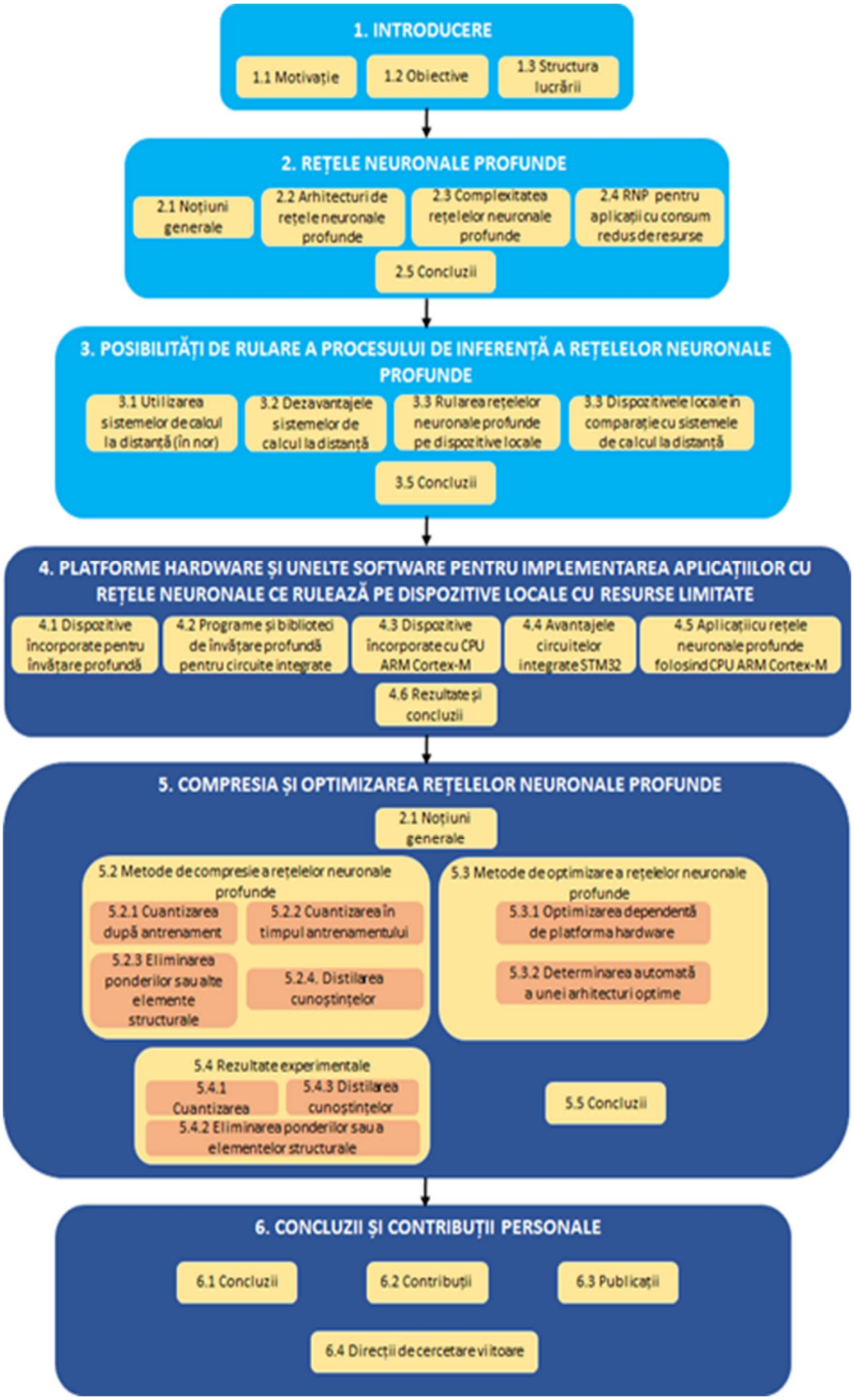

Figure 1. Structure of the paper by chapters and subchapters.

1. DEEP NEURAL NETWORKS

Convolutional Neural Networks *(„Convolutional Neural Networks", CNNs)* have been used since 1980, when a deep neural network called Neocognitron was designed [1]. Using this hierarchical multilayer architecture, it was possible to recognize visual patterns.

CNNs are a specialized type of neural networks for processing data that have a known, grid-like topology. Examples of such data can be those in time series, which can be represented as a 1D grid that reads samples at regular time intervals, or data in an image, which can be represented as a 2D grid of pixels. Convolutional networks have been tremendously successful for a lot of practical applications. The name "convolutional neural network" indicates that the network uses a mathematical operation called convolution. Convolution is a specialized type of linear operation. Convolutional networks are simply neural networks that use convolution instead of general matrix multiplication in at least one of their layers [2].

The CNN networks are the most commonly used for the image classification problem, with a similar operation to how our human brain learns by observing images of different classes. CNNs learn the contents of an image by applying filters to the image and processing methods of different filter sizes, quantity, and nonlinear operations. These filters and operations are applied to multiple layers so that the spatial dimensions of each subsequent layer decrease and their depths increase during the image transformation process. For each filter applied, the depth of the learned content increases. This starts with edge detection, followed by pattern recognition, then a collection of shapes called entities, and so on. This is analogous to the human brain when it comes to how we understand information [3].

Figure 2 shows the diagram for a typical CNN model and its components. In the first step, the convolution operation using a number of 64 filters with the size of 5 x 5 is applied to a grayscale image with the size of 48 x 48 to produce a feature map of number and size 44 x 44 x 64. In the next step, the pooling operation ("Max pooling") with a filter size of 5 x 5 and the stride set to 2 is applied to the input form 44 x 44 x 64. The result of this operation is an output shape reduced to the size of 20 x 20 x 64. After the first 2D convolution operation, the ReLu nonlinear activation function is applied and continues after each operation until the last fully connected layer.

Figure 2. CNN Architecture [3].

Recurrent Neural Networks *(„Recurrent Neural Networks", RNNs)* presented in the paper of Rumelhart et al. [4] since 1986 are a family of neural networks used mainly to detect patterns in a sequence of data. Just as a convolutional network is a neural network specialized for processing a grid of *X-values*, such as an image, a recurrent neural network is a neural network specialized for processing a sequence of values $x^{(1)}, \ldots, x^{n}$. Similarly, just as convolutional networks can be easily adapted to large images, and some convolutional networks can process images of varying sizes, recurrent networks can be adapted to much longer sequences than would be practically possible with other non-specialized networks for sequential data. In addition, most recurring networks can also process variable-length sequences [2].

A main feature of these networks is the ability to support temporal correlations between

sequential input patterns. This is achievable due to the backward propagation of the output signal *("feedback"),* therefore the output signals depend on their previous values. Figure 3 shows the basic structure of a recurrent neural network.

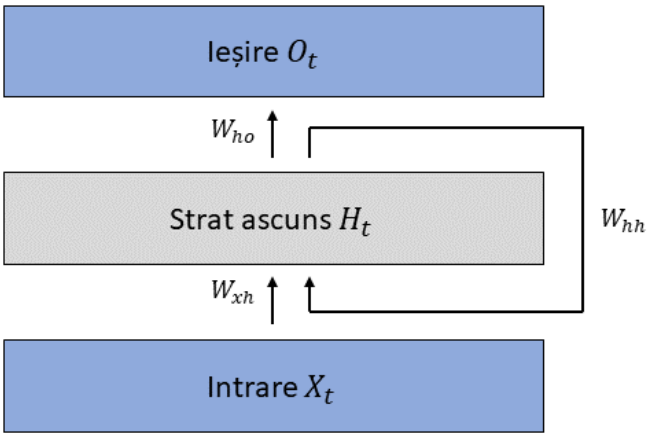

Figure 3. Recurrent Neural Network.

The most commonly used types of recurrent neural networks are: *(„Long short-term memory", LSTM)* and *(„Gated recurrent unit", GRU)*. The LSTM model was introduced in 1997 by S. Hochreiter and J. Schmidhuber [5] and is characterized by better memory capacity than previous networks, four times more memory compared to conventional RNNs. GRU networks are models that allow more information from the past to be saved in order to obtain a better prediction.

ResNet architecture was introduced by Kaiminh He *et al.* in the paper [6] with the main goal of solving the problem of accuracy degradation for deep neural networks as their depth increases. This degradation is not a consequence of overfitting, the reason is that after a certain critical depth, the output loses the input information, so the correlation between input and output begins to deviate, resulting in a decrease in accuracy.

An ensemble of six models with different depths achieved a validation error in the top five of 3.57% and won first place at ILSVRC-2015 *(„ImageNet Large-Scale Visual Recognition Competition", ILSVRC)*. ILSVRC is a competition for the evaluation of object detection algorithms and image classification from 2010 to 2017. An example of a ResNet model is shown in Figura 4.

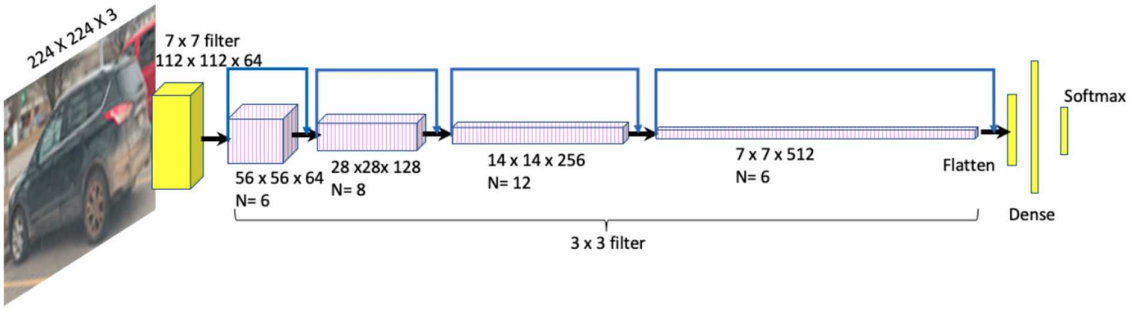

Figura 4. Arhitectura ResNet *[3]*.

2. DEMONSTRATION EXPERIMENT

A. Presentation

In the paper [7] we implemented and evaluated a pre-trained CNN model on the STM32F779I-EVAL evaluation board without the help of an operating system, using a low-cost, energy-efficient ARM Cortex-M7 CPU based microcontroller. This is a small model (Figure 5) consisting of only 3 convolutional layers and was trained using the CIFAR-10 dataset for the image classification problem into 10 output classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck).

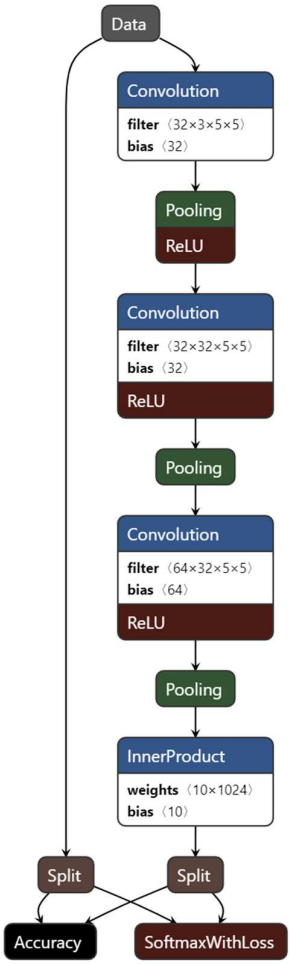

Figure 5. CNN model block diagram [7].

For the implementation of the application I used a series of peripheral devices:

- Peripheral devices available on the board: camera module and LCD display module *(„Liquid-Crystal Display", LCD);*
- Internal peripheral devices: DCMI *(„Digital Camera Module Interface", DCMI)* and the accelerator "Chrom-ART" *(„Direct Memory Access 2D", DMA2D).*

The camera module is used for image acquisition and the LCD display module for displaying the image and the result after the inference process. The internal DCMI pheripheral device is used to ensure by hardware the interface with the camera module, and the DMA2D "Chrom-ART" accelerator for efficient data transfer.

I have implemented two ANSI C program variants: (1) a program to acquire the input

image from the camera module for processing using the CNN model, and (2) a program to test the CNN model using the CIFAR-10 dataset for testing.

Using the first variant, I have obtained positive results as shown in Figure 6.

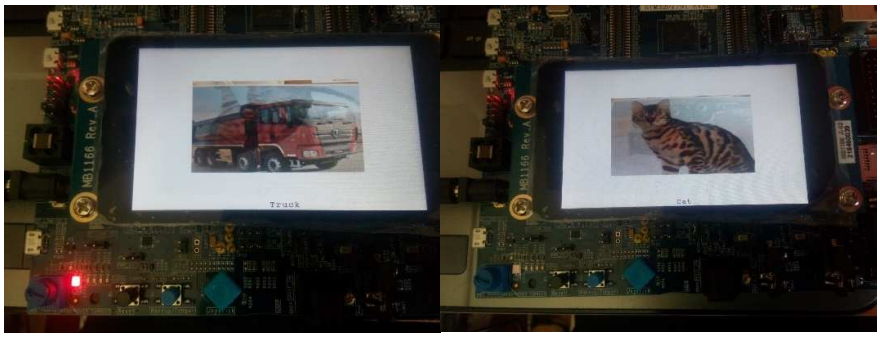

Figure 6. Results obtained for test images aquired using the camera module [7].

For the second variant of the program, I have used the total number of images for test (10000), 1000 of images for each predicted class. I have stored them on a microSD card, organized separately on files with a naming specific for each class. The images are read by the microcontroller via the corresponding peripheral device for the microSD card interface ("SD/SDIO/MMC"). Subsequently, one image from each class is decoded and tested in turn. After each test, the result obtained is verified, and the positive results are stored in the internal memory. Once all images have been tested, the stored results are saved on the microSD card in ".csv" format. Table 1 shows the accuracy achieved for each class.

Table 1. Accuracy results obtained with the CNN model.

Image class	Number of positive results (%)
Airplane	657 (65,7)
Automobile	755 (75,5)
Bird	483 (48,3)
Cat	494 (49,4)
Deer	552 (55,2)
Dog	470 (47)
Frog	489 (48,9)
Horse	591 (59,1)
Ship	690 (69)
Truck	761 (76,1)

B. Conclusions

Thanks to advanced technology that integrates digital signal processing (DSP) functions with hardware support for a wide range of microcontrollers and GPUs, general calculations and various mathematical functions can be implemented with a small number of CPU instructions and executed in a significantly shorter time [8], [9].

The ARM Cortex-M CPU microcontroller series provide to the users a set of general

DSP-based functions, with the intention of covering a wide range of requirements for the implementation of neural networks. The library that contains this set of functions is called CMSIS-NN [10]. One of the examples used to demonstrate this set of functions is a convolutional neural network called CIFAR 10, the name being directly related to the dataset used for training. In the published paper [7], we used the mentioned example in order to run a convolutional neural network on ST's hardware platform, STM32F779I-EVAL.

The overall accuracy achieved is an average of 59.42% with a runtime of 77ms (about 13 frames per second). Although the accuracy is still far from other high-end models optimized for embedded devices, for example an accuracy of 92,4% as reported in the paper [11], the size of the model I used is much smaller (140 KB in the case of 4.3 MB of [11]).

3. POST-TRAINING QUANTIZATION

A. Presentation

Quantization is one of the widely used methods to reduce the size of a model. This is done by changing the numeric format used to represent the parameters. For example, the representation format can be changed from a 32-bit floating-point to an integer of only 8 bits or less. The quantization method is frequently used for the compression of DNNs, in particular to facilitate the running of inference on low-cost devices such as 32-bit microcontrollers. Even if there are microcontrollers that have a precision FPU unit, its use is usually avoided to reduce memory and power consumption [12]. At the same time, however, there are many microcontrollers without an FPU unit.

In the paper [13] I implemented and evaluated the available post-training quantization methods using the TensorFlow Lite library: (1) dynamic range quantization (PTDRQ), (2) integer quantization (PTIQ), (3) float16 quantization (PTFQ), and (4) integer quantization with int16 activations (PTIQA).

I used three models of convolutional neural networks with different sizes: DenseNet121 (7.0 MB) [14], ResNet50 (23.5 MB) [6] and SE-PreAct-ResNet101 (42.5 MB) [15]. I chose these models to track the impact on accuracy and compression ratio depending on the size. For training, the CIFAR-10 dataset is used for the image classification task.

All the above mentioned quantization methods are implemented using a similar execution flow, as described in the block diagram shown in Figure 7. After the training stage, the models were saved in the SavedModel format, which is a compatible format for TensorFlow Lite conversion. After training the model (Step 1), in Step 2 the model with the SavedModel format is converted to the TensorFlow Lite format to use its size as a reference point for determining the compression ratio. In Step 3, the model is converted to TensorFlow Lite format with the quantization option enabled. In Step 4, the compression ratio is calculated as the ratio of the initial floating-point model size (calculated in Step 2) to the model size after quantization. In the last step (Step 5) the quantized model is tested using the CIFAR-10 test images, then the accuracy is calculated.

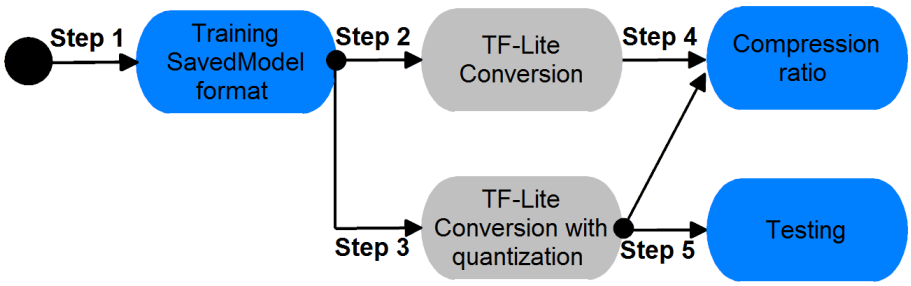

Figure 7. TensorFlow Lite post-training quantization block diagram [13].

B. Conclusions

Figure 8 shows the compression ratio obtained for each DNN model after applying the quantization methods. Since the original floating-point models use a 32-bit representation and quantization is performed on an 8-bit representation, a compression ratio of 4× is expected. Based on the obtained result, the compression ratio is slightly less than 4×, and using a DNN model with a small size (e.g. DenseNet121) the ratio tends to be smaller than the others, especially for dynamic range quantization and integer quantization with 16-bit activations. Using the 16-bit floating-point quantization method, the compression ratio is 2×. For this method, the difference between the DNN models is insignificant. In conclusion, the compression ratio using TensorFlow Lite quantization methods varies slightly depending on the size of the model.

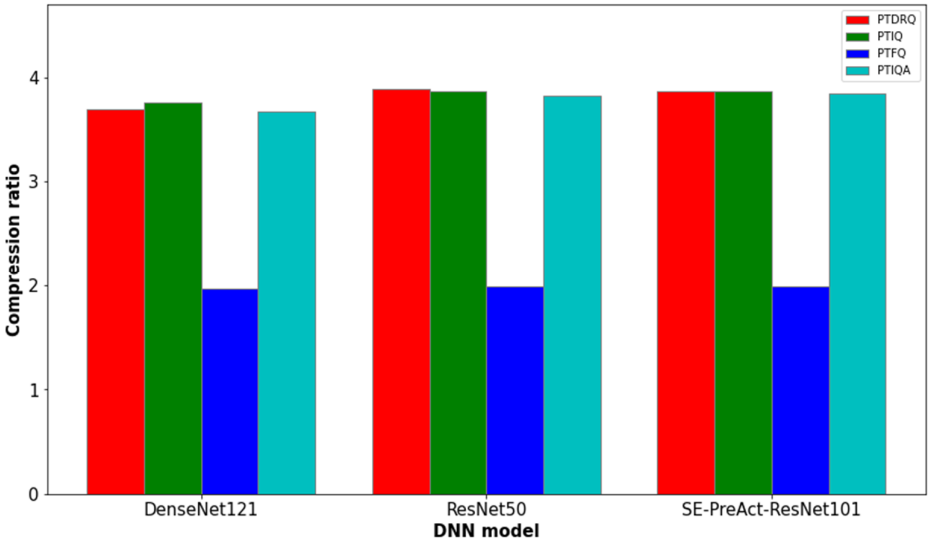

Figure 8. Compression ratio [13].

Table 2 shows the accuracy achieved after each quantization method, together with the accuracy before quantization and the total number of parameters. Generally speaking, the accuracy loss is less than 1%. Therefore, TensorFlow Lite's post-training quantization methods do not significantly affect the accuracy.

More precisely, using 16-bit floating-point quantization the accuracy loss of is insignificant, while the model size is reduced by 50%. In some cases, for example DenseNet121 and ResNet50, the accuracy is even improved. Using the PTIQA technique, an increase in accuracy of 0.01% is achieved for DenseNet121, and with the PTDRQ technique, an increase in accuracy of 0.02% for ResNet50 is obtained. For SE-PreAct-ResNet101, the lowest accuracy loss is achieved using the PTFQ technique, followed by PTIQA. Using full integer quantization, the largest decrease in accuracy is obtained. At the same time, however, the decrease is small. In the worst case, this is only 0.43% for the DenseNet121 model. A different result is obtained for SE-PreAct-ResNet101, where the lowest precision was obtained using dynamic range quantization.

Table 2. Accuracy results before and after quantization.

Features/Model	**DNN model**		
	DenseNet121	***ResNet50***	***SE-PreAct-ResNet101***
Model size	7.0 MB	23.5 MB	42.5 MB

Initial accuracy	95.51%	95.13%	94.76%
PTDRQ	95.47%	95.15%	**94.63%**
PTIQ	**95.08%**	**95.03%**	94.69%
PTFQ	95.51%	95.13%	94.75%
PTIQA	95.52%	95.12%	94.72%

In the paper [13], I compared TensorFlow Lite post-training quantization methods using three different convolutional neural network models. They are classified according to their size as small, medium, and large. The compression ratio using 8-bit quantization is just under 4×, and using 16-bit floating-point quantization is 2×. I noticed that for the small size category, the compression ratio tends to be lower. The degradation of accuracy is almost insignificant; in the worst case, the impact is only 0.43% for the small size category when using the PTIQ method.

4. MAGNITUDE-BASED WEIGHTS PRUNING

A. Presentation

It has been observed that certain structural elements in a network are redundant and their contribution to the final prediction is small. Based on this observation, it was concluded that removing these elements is possible to reduce the size of a model, without a significant impact on accuracy. Such examples can be: removing connections, convolutional layers, or neurons from fully connected layers. A challenge that comes with this technique is to find a suitable strategy for identifying the elements that can be pruned with minimal degradation on network performance.

Unlike quantization, weights pruning is a technique that addresses the problem of over-parameterization of neural networks. It is known that the size of a neural network is not in complete correspondence with its performance, therefore, pruning these parameters or structural elements will result in a smaller size of the network, without affecting the performance. In addition to other compression methods, it comes as an essential strategy for reducing the size of the model in the face of the growing need to deploy deep learning models on resource-constrained devices.

In this direction, my proposal was to use the teacher model from the paper [16] for implementing magnitude-based weights pruning techniques using the TensorFlow optimization toolkit. A weight-matrix model that contains multiple values of 0 may be more efficient in terms of memory footprint and inference time. However, the hardware device used may impose certain limitations and operations that involve 0 weights not being optimized. With this in mind, dedicated hardware accelerators have made progress in recent years. The recent study by V Isaac–Chassande *et al.* [17] present an overview and design details of the state-of-the-art dedicated hardware accelerators for random 0-value array computing. The teacher model is designed for estimating eye gaze in automotive applications. In the context of human-computer interaction, the field of gaze estimation is very important. Today, driver inattention is still a major contributor to car collisions. The features of Advanced Driver Assistance Systems *("ADAS")* have been specifically designed to mitigate the incidence of car accidents caused by driver inattention. Estimating the driver's gaze is considered a primary input for algorithms that can recognize inattention while driving.

TensorFlow Model Optimization Toolkit [18] provides solutions for magnitude-based weights pruning. Using this pruning method, the weights that have low values are set to zero with the goal of eliminating unnecessary connections. This is applied during the training process to allow the neural network to adapt to the involved changes. The training process can only be

fine-tuned and does not have to be performed from scratch. Using a pre-trained model and applying a fine-tuning step is usually a better option.

The pruning schedule can be configured as constant sparsity schedule or by using a polynomial decay function. With the first configuration, a target value is defined as the percentage of weights that will be zero. Using the polynomial decay function, the pruning schedule is performed according to the polynomial decay function by specifying the initial and final sparisty.

The purpose of this paper is to evaluate and compare different pruning schedules, such as constant sparsity schedule and polynomial decay function for magnitude-based weights pruning using the TensorFlow model optimization toolkit and a custom CNN architecture for the eye gaze estimation.

B. Conclusions

The constant sparsity schedule is applied using a final sparsity value in the range [0.10 ... 0.89] with the increment step of 0.01. This means that the model is subjected to 80 different configurations and training loops. A training loop is performed for 35 epochs. The schedule using polynomial decay function has a configuration similar to constant sparsity schedule, with a final sparsity value in the same range [0.10 ... 0.89] and the step of 0.01, with the same number of loops and training epochs. The initial sparsity is set to 0.00 for each training loop.

The most positive result was obtained for the second experiment. In this case, the accuracy degradation is only 1.08%, the final elimination value is 0.75, while the compression ratio is 8.06.

Table *3* shows the results obtained for the constant sparsity schedule. It comprises the maximum validation accuracy, the maximum test accuracy, the target sparsity for test accuracy, and the compression ratio at which we have achieved the maximum test accuracy. The experiments were conducted three times to calculate an average of the results and highlight the overall behavior. The average result for validation accuracy is 75.06%, while for test accuracy it is 73%. Considering the test accuracy of the teacher model of 74.48% [16], we achieved an average target sparsity value of 0.73 and a compression ratio of 7.74 with a small decrease in accuracy of only 1.48%.

The most positive result was obtained for the second experiment. In this case, the accuracy degradation is only 1.08%, the final elimination value is 0.75, while the compression ratio is 8.06.

Table 3. Results using constant sparsity schedule.

Result	**Constant Sparsity**		
	1st	***2nd***	***3rd***
Validation accuracy	75.06%	75.23%	74.89%
Test accuracy	72.47%	73.40%	73.15%
Target sparsity for test accuracy	0.76	0.75	0.68
Compression ratio for test accuracy	8.38	8.06	6.79

Table 4 shows the results obtained for the pruning schedule using the polynomial decay function. The results are presented using the aforementioned performance indicators. With this configuration, the average result for validation accuracy is 73.8% and the average for test

accuracy is 72.9%. These results were obtained with an average target sparsity value of 0.57 and a compression ratio of 5.52.

In this case, the most positive result was obtained for the first experiment, with an accuracy degradation of 0.74%, a final elimination value of 0.59, and a compression ratio of 5.69.

Using constant sparsity schedule, the model weights are significantly reduced down to 0.76 target sparsity, achieving a maximum compression ratio of 8.38 with a degradation in test accuracy of only 1-2%. With the pruning schedule using the polynomial decay function, the target sparsity value obtained is maximum of 0.62 with a compression ratio of 6, lower than in the case of constant sparsity schedule, while the accuracy degradation is similar to constant sparsity. Based on the above results, it can be concluded that in order to achieve high target sparsity, constant sparsity schedule is a more suitable choice compared to the polynomial decay function.

Table 4. Results using polynomial decay function pruning schedule.

Result	**Polynomial Decay Function**		
	1st	***2nd***	***3rd***
Validation accuracy	73.79%	73.53%	74.21%
Test accuracy	73.74%	72.47%	72.47%
Target sparsity for test accuracy	0.59	0.5	0.62
Compression ratio for test accuracy	5.69	4.88	6

In this paper we have implemented and evaluated the following magnitude-based weights pruning schedules: constant sparsity schedule and polynomial decay function pruning schedule with the implementation performed using TensorFlow model optimization toolset. Using constant sparsity schedule, a compression ratio of up to 8.06 with a target sparsity value of 0.75 can be achieved, while the accuracy degradation is only 1.08%. Using polynomial decay function pruning schedule, the result is less satisfactory in terms of compression ratio, which is up to 5.69 with a final value of 0.59, while the accuracy degradation is 0.74%. Therefore, constant sparsity schedule is a better choice compared to the polynomial decay function.

5. KNOWLEDGE DISTILLATION

A. Presentation

In the paper [16] I set out to address the problem of eye gaze estimatimation with applicability in the automotive field. My proposal uses a concept of knowledge distillation applied to a custom CNN model architecture, with the role of a teacher model. The field of eye gaze estimation has significant importance in the context of human-computer interaction and for multiple applications in various fields, such as the automotive industry, market research and medical. Currently, drivers' inattention continues to be an essential factor that in most situations leads to car collisions. Implementation of advanced driver assistance systems *(„Advanced Driver-Assistance Systems", ADAS)* was proposed as a potential solution to mitigate the incidence of car accidents caused by driver inattention. The driver's gaze can be an indicator for detecting fatigue or inattention while driving. In such situations, it is possible to give

warnings to the driver and, if necessary, decide on appropriate measures to avoid a collision [19].

Compressing a DNN model is a common practice for achieving a small network for low-cost and resource-limited hardware devices. Knowledge distillation is a particular method that involves training to transfer knowledge from a larger-sized network to another network that is significantly smaller in size. Bucilua *et al.* [20] successfully demonstrated for the first time that the knowledge gained from a large set of models can be transferred to a single and smaller model. The purpose of using this method is to train the student network so that it replicates the performance of the teacher network, but with a smaller size, respectively fewer memory usage and computational resources.

The process of implementing the knowledge distillation algorithm involves the following main steps: (1) defining the teacher and student networks, (2) training the teacher network and (3) training the student network with knowledge transfer from the teacher network. These three steps are briefly outlined below.

1) Defining teacher and student networks

The teacher network can be a standard high-performance model with a large number of parameters (e.g. ResNet, DenseNet or EfficientNet), while the student network can also be a standard model but with a smaller number of parameters. Depending on the task, a custom architecture of teacher or student networks can also be used.

2) Teacher Network Training

The teacher network is trained using a standard training procedure, with the aim of achieving the best accuracy.

3) Training the student network with knowledge transfer from the teacher network

At this stage, the knowledge distillation algorithm is used. Forward propagation is performed for the teacher and student networks, while backward propagation applies only to the student network. The primary loss function is defined using two distinct loss functions: the student loss function and the distillation loss function.

The model's knowledge are classified into three different types: response-based knowledge, feature-based knowledge, and relationship-based knowledge [21]. Response-based knowledge focuses on the final output layer, where the student model will learn the predictions of the teacher model. Feature-based knowledge uses the knowledge from the intermediate layers of the teacher model to train the student model. Relationship-based knowledge focuses on the correlation between feature maps, graphs, similarity matrix, feature embeddings, or probabilistic distributions. In this thesis, we used response-based knowledge because it showed the best results for different tasks and applications.

The block diagram for the knowledge distillation algorithm is shown in Figure 9.

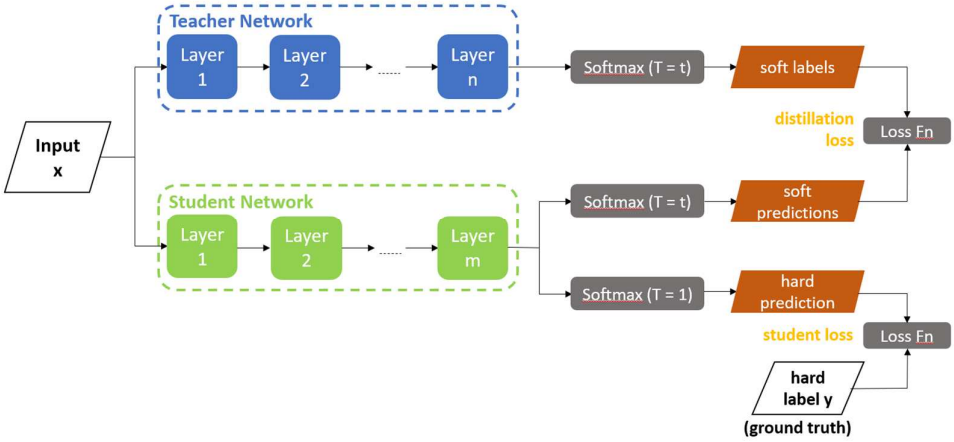

Figure 9. Block diagram for knowledge distillation [16].

The transfer of knowledge from the teacher model to the student one is achieved by minimizing the main loss function in order to produce the same probabilities as those produced by the teacher model. More specifically, this refers to the output of the *"Softmax"* function applied to predictions before they are normalized. For these predictions of the teacher model, the name "*logits"* is usually used. Most commonly, the correct class of the probability distribution has a higher level compared to the other class probabilities that are close to zero. Therefore, the result provided by this probability distribution is very similar to the grounds truth labels of the dataset. Regarding this behavior, Hinton *et al*. [22] introduced the softmax parameter *"temperature"*. By denoting this parameter with *T*, the probability of class p_i in *"logit"* z_i is calculated with the equation (5.1).

$$p_i = \frac{exp\ (z_i/T)}{\sum_j exp\ (z_j/T)} \tag{5.1}$$

When the parameter *T* is set to the value 1, the equation becomes a standard *"Softmax"* function. When *T* has a value greater than 1, the probability distribution will provide more information about the classes for which the teacher model reported a prediction close to the correct class. This is the knowledge of the teacher model that is transferred to the student model using the distillation algorithm. When the distillation loss function is calculated using the *"soft"* labels, the same value of *T* is used to calculate the *"softmax"* function on the *"logits"* of the student model.

Model training with knowledge distillation has been observed to be a benefit using the ground truth labels of the dataset as well. Therefore, the second loss function is calculated with the parameter *T* set to the value 1. This is a standard loss function called the student model loss function. By noting the distillation loss function with H_d (5.2) and the loss function of the student model with H_s (5.3), the main loss function is calculated using the equation (5.4):

$$H_d = H\big(\sigma(z_t; T = \tau), \sigma(z_s; T = \tau)\big) \tag{5.2}$$

$$H_s = H(y, \sigma(z_s; T = 1)) \tag{5.3}$$

$$L(x; W) = \alpha * H_s + \beta * H_d \tag{5.4}$$

where *H* is the cross-entropy loss function, σ is the *"Softmax"* function, and z_t, z_s are the *"logits"* of the teacher and student models, *y* is the ground truth label, *x* is the input, *W* are the parameters of the student model, and α, β are coefficients.

The teacher model is a small custom CNN model with only 5 convolutional layers. It is used as a base model for student models. To get a smaller student model, we applied two different methods: layerwise compression and widthwise compression. Layerwise compression involves reducing the number of convolutional layers, while widthwise compression stands for reducing the number of filters. The optimal student model is considered to be the one that has the smallest size, but at the same time has not suffered a significant loss of accuracy. To increase the search space in finding the most optimal student model, various combinations of layerwise and widthwise compression are also considered.

For layerwise compression, I have defined the following student models:

- *Student_cut5*, where I gave up the last convolutional layer;

- *Student_cut4*, where I gave up the last two convolutional layers;

For widthwise compression, I have defined the following student models:

- *Student_width10*, a model with 10% fewer filters;
- *Student_width30*, a model with 30% fewer filters;
- *Student_width50*, a model with 50% fewer filters.

Layerwise and widthwise compression combinations are defined by applying widthwise compression to the student models resulting from layerwise compression. I used the name *Student_cutX&widthY*, where *X* is the layerwise compression model and *Y* is the widthwise compression model. In total, 11 student models were obtained with the aim of finding the optimal configuration.

B. Conclusions

- **Results using the knowledge distillation algorithm on the PC station**

Depending on the user's requirements, the compression of a model can be for several types of outcomes: (1) reducing the size of the model without an impact on performance or even improving accuracy if possible, (2) reducing the model size and inference time, while supporting a small degradation in accuracy, and (3) significantly reducing the model size and inference time, but keeping accuracy within acceptable limits.

Based on the results obtained, the following models can be used for the first case: *Student_cut5*, *Student_width10* and *Student_cut5&width10*. In the case of these models, the accuracy is higher than that of the teacher model. The highest compression ratio and reduction of inference time are achieved with the *Student_cut5&width10* model. However, if the user's requirement is not to achieve maximum accuracy, then the *Student_cut5&width10* model is the most suitable solution.

For the second case, with the following models we achieved a decrease in accuracy of less than 1% compared to the teacher model: *Student_cut4*, *Student_width30* and *Student_cut5&width30*. The highest compression ratio and inference time reduction are achieved for the *Student_cut5&width30* model, which is best suited for this category.

When model size and inference time reduction are considered the most important aspects, the following models are best suited: *Student_width50*, *Student_cut5&width50*, *Student_cut4&width10*, *Student_cut4&width30,* and *Student_cut4&width50*. The highest compression ratio and reduction of inference time was achieved with the *Student_cut4&width50* model, but in this case the accuracy is only 68.29%. The model in this category that has the highest accuracy is *Student_cut4&width10*. Its accuracy is close to that of the teacher model, and the compression ratio is 3.24.

Based on the results presented above, the following conclusions can be summarized:

- Using the knowledge distillation algorithm, accuracy can be improved by up to 9.5% compared to using a conventional training procedure. This can be achieved using the model obtained from the layer and width compression method: *Student_cut4&width30*;
- The knowledge distillation algorithm is more efficient when the coefficient α is set to 0 or 0.5 and T is set to a higher value, such as 14. Therefore, the efficiency is higher when the main loss function (5.4) is based only on the distillation loss function H_d or when the student loss function H_s and distillation loss function H_d have the same weight;
- The combined use of layerwise and widthwise compression is more effective than using them independently;

- The inference time is reduced more using widthwise compression compared to layerwise compression. The combination of these leads to an improvement, but not significantly.

- **Results on the STM32H747I-DISCO hardware platform**

The purpose of validation on the STM32H747I-DISCO platform is to demonstrate that the resulting accuracy is similar to that obtained on the PC station, given that no compression is applied that could introduce a reduction in accuracy. The small difference that is noticeable may be a consequence of the fact that the accuracy is calculated using different instruments.

Regarding the use of RAM and ROM, the following considerations can be briefly presented depending on the layerwise or widthwise compression: (1) RAM usage is lower using widthwise compression (in the case of layerwise compression, the difference is not significant), (2) ROM usage is gradually lower as the size of the model decreases (in this case, there is no visible difference between layerwise and widthwise compression, and (3) the combination of layerwise compression and widthwise compression is more efficient because the RAM required is also reduced.

MACC complexity is higher when using layerwise compression. This is significantly reduced when using widthwise compression. The most effective choice is also to use a combination of layerwise and widthwise compression.

The inference time follows a distribution similar to that presented in the results obtained on the PC station. In conclusion, inference time is reduced more using widthwise compression, and the combination with layerwise compression leads to improvements. In the case of the *Student_cut4&width30* model, the inference time is 870.5 ms, and the accuracy is 73.66%. This denotes that one frame per second can be achieved with an accuracy close to the maximum value. This inference time can be supported for practical applications, which allows the implementation of a real-time gaze detection system on a low-cost and energy-efficient hardware platform.

Nowadays, deep neural networks and their associated paradigm of deep learning are ubiquitous in the automotive field. Published paper [16] refers to an optimization procedure that aims to implement a neural network model for the eye gaze estimation problem on low-cost hardware. My proposal uses a concept of knowledge distillation applied to a custom CNN architecture, called the teacher model. Based on this, several CNN student models are derived using layerwise and widthwise compression techniques. Subsequently, they are evaluated in terms of different performance metrics such as, neural network size and inference time. We have proposed the analyzed compression methods that are more suitable and can meet specific user requirements, such as model size, accuracy, and inference time. Finally, we evaluated the student models on the STM32H747IDISCO embedded device in terms of accuracy, memory usage, MACC complexity, and inference time. Using the knowledge distillation algorithm, accuracy can be improved by up to 9.5% compared to the conventional training procedure. A compression ratio of up to 8.86 is achieved with a decrease of less than 10% in accuracy. The inference time using width compression is reduced more. Combining layer and width compression is more efficient and a good compromise between model size, accuracy, and inference time. The results of validation on STM32 hardware showed that in order to reduce the use of RAM and ROM, the combination of layer and width compression is the optimal solution. This solution is also the right choice for optimizing MACC complexity and inference time simultaneously. However, I was able to identify some limitations of the methodology used, as it only transfers knowledge related to the results of the initial model and does not capture the internal representations learned by the teacher model. Also, the student model can learn less

from the teacher model variants where there is an important architectural gap compared to the student model.

6. CONCLUSIONS AND PERSONAL CONTRIBUTIONS

Deep neural networks have reached a very advanced stage in recent years, being used more and more frequently for different applications and devices in people's lives. Obviously, artificial intelligence will be a field that profoundly marks the technological progress of the current century. The use of neural network models from very isolated applications such as smartwatches to applications that solve as many requirements as possible with a single model has led to the need for a very large diversity of models. At the same time, it is known that the high-end models with which the best performance has been achieved are very expensive in terms of occupied memory, computing requirements and power consumption. For this reason, in the relatively recent period and today the level of research has reached a point where the focus is on designing smaller models with satisfactory performance and that are suitable for the class of isolated applications (mentioned above). Meanwhile, to build on the already existing progress with large neural networks, the compression of pre-trained neural networks is another parallel research point. These directions are justified by the large number of billions of local devices that require artificial intelligence algorithms [23]. Therefore, I have noted the relevance of a research at the level of doctoral studies in this direction, having as its main purpose the compression of deep neural networks.

The main contribution of this thesis is the study, description and implementation of deep neural network compression algorithms such as quantization, knowledge distillation and magnitude-based weight pruning using convolutional neural networks of different sizes and a specific model for the driver's eye gaze detection application. The implementation was carried out using different configurations in order to present in a detailed way the results regarding the accuracy, the compression ratio obtained and the inference time with a focus on the right configuration according to the system requirements.

During this doctoral thesis, I studied more than 160 bibliographic titles, I elaborated two articles published in ISI journals Q2 („IEEE Access”), Q3 („MDPI Electronics”) and indexed „Web of Science”; an indexed conference paper „ISI Proceedings” and „Web of Science”; an indexed conference paper BDI and two conference papers that are currently being drafted. For all these works I contributed as first author, except for the last conference paper that is being written and where I will be as second author. I would like to point out that currently the papers have a total of 31 citations according to the Google Scholar report. In the next section I have presented the personal contributions at the level of detail for each scientific paper.

The personal contributions present in the journal article ISI Q3 („MDPI Electronics”) [24] are:

- I have presented some low-cost (less than $10 for a microcontroller) and representative hardware devices along with the libraries or supporting tools that help implement machine learning algorithms on them;
- I have summarized 13 papers in recent years where microcontrollers with ARM Cortex-M CPUs have been used to run machine learning algorithms. I aimed to point out the following main aspects: the architecture of the model, the hardware characteristics (such as: available memory footprint, CPU architecture and operating frequency), the tools and support libraries used together with the results obtained in terms of: accuracy, inference time and power consumption;
- I have described in detail the results obtained with a focus on the cases where the best

results have been obtained. I have also discussed the main reasons that are in many cases an impediment to achieving the desired results;
- I have highlighted research challenges and opportunities that highlight important aspects and pose a challenge for future progress in this field.

The personal contributions present in the journal article ISI Q2 („IEEE Access") [16] are:

- I have defined and trained a custom CNN architecture to classify the eye gaze estimation, with a size of 6.14 MB and an accuracy of 77.48%;
- I applied the knowledge distillation methodology with different configurations to train multiple student models defined using layerwise and widthwise compression methods. For the teacher model I used the custom CNN architecture;
- I presented an evaluation of the experimental results in order to identify the most appropriate compression method according to the user's requirements, such as: model size, desired accuracy and expected time for an inference;
- I validated the student models using the AI validation interface of the X-CUBE-AI extension package on the STM32H747I-DISCO hardware device. To this end, I have presented details on the accuracy achieved, memory usage, MACC complexity, and inference time.

Personal contributions present in the indexed conference paper „ISI Proceedings" and „Web of Science" [7] are:

- I have converted a CNN network model from the generic Caffe format to a format compatible with the STM32 microcontroller;
- I have implemented the interfaces between the peripheral modules (such as LCD display module and camera module) and the CNN model, so that together they form a functional system;
- I validated the CNN model directly on the microcontroller in real time using the camera module and tested the overall behavior of the model using the test images of the CIFAR 10 dataset.

Personal contributions present in the indexed conference paper BDI [13] are:

- I have compared the compression ratio that is obtained using 8-bit integer quantization and 16-bit floating-point quantization using four different PTQ techniques;
- I analyzed the compression ratio according to the size of the DNN model, using three different models;
- I have highlighted the influence of PTQ techniques on the accuracy of the models.

The personal contributions present in the paper aimed on magnitude-based weights pruning (work in progress) are:

- I have applied the technique magnitude-based weight pruning to a custom CNN model trained in order to classify the eye gaze estimation;
- I have compared the constant sparsity schedule and the polynomial decay function with respect to the compression ratio, the target sparsity, the test and validation accuracy;
- I have presented an evaluation of the experimental results in order to identify the

appropriate schedule for magnitude-based weights pruning in order to obtain a high degree of elimination and a high compression ratio, without a significant loss of accuracy.

Bibliography

[1] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural Networks,* vol. 1, p. 119–130, 1988.

[2] I. Goodfellow, Y. Bengio and A. Courville, Deep learning, Cambridge, MA: MIT Press, 2016.

[3] K. Kar, Mastering Computer Vision with TensorFlow 2.x, May, 2020.

[4] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, p. 533–536, 1986.

[5] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.,* vol. 9, p. 1735–1780, 1997.

[6] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016.

[7] I. L. Orăşan and C. D. Căleanu, "ARM Embedded Low Cost Solution for Implementing Deep Learning Paradigms," Timisoara, 2020.

[8] A.-A. Erofei, C.-F. Druţa and C. D. Căleanu, "Embedded Solutions for Deep Neural Networks Implementation," Timisoara, 2018.

[9] R. MIRSU, S. MICUT, C. CALEANU and D. B. MIRSU, "Optimized Simulation Framework for Spiking Neural Networks using GPU's," vol. 12, pp. 61-68, 2012.

[10] L. Lai, N. Suda and V. Chandra, "CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs," January 2018.

[11] M. Ayi and M. El-Sharkawy, "RMNv2: Reduced Mobilenet V2 for CIFAR10," Las Vegas, NV, USA, 2020.

[12] J. Lee, S. Kang, J. Lee, D. Shin, D. Han and H.-J. Yoo, "The Hardware and Algorithm Co-Design for Energy-Efficient DNN Processor on Edge/Mobile Devices," *IEEE Trans. Circuits Syst.,* Vols. 67-I, p. 3458–3470, 2020.

[13] I. L. Orăşan, C. Seiculescu and C. D. Caleanu, "Benchmarking TensorFlow Lite Quantization Algorithms for Deep Neural Networks," Timisoara, 2022.

[14] G. Huang, Z. Liu, L. V. D. Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," Honolulu, 2017.

[15] J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, "Squeeze-and-Excitation Networks," *IEEE Trans. Pattern Anal. Mach. Intell.,* vol. 42, p. 2011–2023, 2020.

[16] I. L. Orasan, A.-I. Bublea and C.-D. Caleanu, "Deep Learning-Based Eye Gaze

Estimation for Automotive Applications Using Knowledge Distillation," *IEEE Access,* vol. 11, p. 120741–120753, 2023.

[17] V. Isaac–Chassande, A. Evans, Y. Durand and F. Rousseau, "Dedicated Hardware Accelerators for Processing of Sparse Matrices and Vectors: A Survey," vol. 21, pp. 1-26, 2024.

[18] TensorFlow, "Model optimization," [Online]. Available: https://www.tensorflow.org/model_optimization. [Accessed 6 July 2024].

[19] H. U. Draz, M. I. Ali, M. U. G. Khan, M. Ahmad, S. Mahmood and M. A. Javaid, *An Embedded Solution of Gaze Estimation for Driver Assistance using Computer Vision,* 2021.

[20] C. Buciluǎ, R. Caruana and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2006.

[21] J. Gou, B. Yu, S. J. Maybank and D. Tao, "Knowledge Distillation: A Survey," *Int. J. Comput. Vis.,* vol. 129, p. 1789–1819, 2021.

[22] G. Hinton, O. Vinyals and J. Dean, "Distilling the Knowledge in a Neural Network," *arXiv e-prints,* p. arXiv:1503.02531, March 2015.

[23] L. Ding, "Artificial intelligence solutions running on STM32," [Online]. Available: https://www.st.com/content/dam/specialevents-assets/electronica-china-2023/st-edge-ai-english.pdf. [Accessed 7 July 2024].

[24] I. L. Orășan, C. Seiculescu and C. D. Căleanu, "A Brief Review of Deep Neural Network Implementations for ARM Cortex-M Processor," *Electronics,* vol. 11, p. 2545, August 2022.