Inference and Performance Analysis of Convolutional Neural Networks used for Human Gesture Recognition on IoT-Devices

Björn Sievers*, Sebastian Hauschild† and Horst Hellbrück†
Luebeck University of Applied Sciences, Germany
Department of Electrical Engineering and Computer Science
*Email: bjoern.sievers@stud.th-luebeck.de
†Email: sebastian.hauschild, horst.hellbrueck@th-luebeck.de

***Abstract*—The steady growth in computing power of microprocessors recently enabled the implementation of rudimentary neural networks on these devices. This has drawn the focus of scientists to this still a very unexplored topic. In this paper, we investigate Convolutional Neural Networks implemented on a microcontroller for the real-time classification of human gestures. We examine the effects of the complexity of the neural network on the inference time, classification accuracy, and energy demand. The neural networks are trained and tested on a generated dataset, which consists of 10 labeled gestures, generated from a triaxial accelerometer. The neural networks are deployed on the Adafruit Feather Bluefruit Sense board. This board hosts the nrF52840 SoC which is built around a Cortex-M4F-based microcontroller. The measurement results show that more complex neural networks achieve higher accuracies, but involve higher energy requirements and longer inference times.**

***Index Terms*—Edge Computing, Convolution Neural Network(CNN), Microcontroller**

I. Introduction

The topic of Deep Learning (DL) in connection with the Internet of Things (IoT) devices has gained a lot of attention in recent years in the field of research as well as in the commercial segment [1]. A contributing element is the ongoing advancement of semiconductor technology and the associated increase in the computational capabilities of microcontroller units (MCUs). The increase in the performance of MCUs in recent years has made it possible to integrate neural networks (NNs) on resource-constrained end devices such as smartphones or sensor systems. When the data processing occurs close to or directly on the end device instead of being forwarded to a cloud server, we refer to this as edge computing [2], [3].
An implementation of NNs at or close to the end device entails some advantages compared to conventional IT solutions. Edge computing reduces the latency which enables inferences (i.e., the evaluation of the sensor data) to be performed in real-time. This is of major importance in some applications. For example, the evaluation of camera data in autonomous driving applications requires a minimal latency [3]. An implementation close to the end device also increases the scalability of the IoT sensor network, as the evaluation of the data is directly performed on the node. This enables that just the metadata and not the raw sensor data has to be transmitted to the server, thus bandwidth-intensive data transmission is reduced. Moreover, the privacy of the user is increased, since valuable or personal data for the user is no longer transmitted to a cloud server to be processed [2], [4]. Thus, edge computing introduces some benefits, however, it still faces some challenges in implementing neural networks on IoT devices. Even though the computational power of MCUs has increased in recent years it is still extremely limited compared to cloud servers, desktop computers, Application-Specific Integrated Circuits (ASICs), or Neural Processuing Units (NPUs) [5]. Furthermore, MCU systems have limited memory capacities. Typical MCU's have a flash memory with a size of 1-2 MB and less than 1 MB of RAM available. Therefore, the size of a neural network will be limited in terms of the computational complexity of a neural network that is implemented on a microcontroller. However, some applications may require high energy efficiency, scalability, or need to operate on a low budget. Therefore, in certain cases, the use of microcontrollers is the desired solution [3].

In this paper, we focus on the investigation of deployed Convolutional Neural Networks (CNNs) on a resource-constrained microcontroller-based sensor system for real-time recognition of hand gestures. We use CNNs for the classification of human gestures based on sensory data. Human gesture recognition is done by using sensory data from a 3-axis IMU in the form of an accelerometer. We train the CNNs based on a training dataset consisting of up to 10 individual gestures. Our work focuses on the examination of the model complexity and its effects on inference time, accuracy and energy demand.

The paper is structured as follows. Section II starts with the discussion of related work. Section III gives an overview of the software libraries and the dataset used to build and train the neural networks. We further provide information on the hardware platform used to deploy and analyze the neural networks in this Section. In Section IV, we present the archi-

tectures of the CNNs which will be deployed on our hardware platform as well as the estimated floating-point operations required per layer. In Section V, we analyze our deployed CNNs with regards to accuracy metrics, inference time, and energy demand. Section VI summarizes our conclusions.

II. Related Work

In this research, time-series acceleration data is to be classified into different gestures. A promising candidate is the CNN since the use of CNNs to analyze time-series data is widespread. It is typically used to classify speech signals [6], natural language processing [7], images or objects inside images [8] [9]. Typical neutral networks such as multilayer perceptron neural networks (MLPs) are also used to classify data. The advantage of using CNNs over to traditional MLPs is that CNNs factor in spatial information of the input signal, while typical MLPs are translation invariant. This typically enables CNNs to achieve higher accuracies compared to MLPs on this type of data [10]. Therefore, we concentrate on the investigation of CNNs in this work.

Yuqing Chen et. al. [11] investigated human activity recognition, using a CNN, based on raw tri-axial accelerometer sensor data. In their experiment, their model achieved an average accuracy of $93.8\,\%$ based on a dataset consisting of eight different activities to classify and approximately 32000 total labeled samples. They show with this work that the use of CNNs for the classification of time-series data, in their case in the form of human activity recognition, is possible. The CNN model of their work is designed for mobile devices, i.e., smartphones. However, processing units of a mobile device, such as smartphones, exceed the available resource capacities of a microcontroller system.

Juraj Dudak et. al. [12] proposed a CNN which is used for the classification of six different motion patterns by using tri-axial accelerometer data. Their network achieved an accuracy of $88\,\%$ and is deployed on a microcontroller. However, the focus of their work lies in the implementation process of the CNN on an STM32L432 MCU. Furthermore, in their application, a completely new sampling batch is acquired before a classification is performed. It is not explicitly stated how long the CNN needs for the inference. In case a real-time system is required, too long inference times may cause that data points can not be sampled in time, and thus information of the process is lost.

Warden et. al. [13] proposed in their work a CNN model that is implemented on a microcontroller for the classification of human gestures based on raw tri-axial accelerometer data. Their model distinguishes between 4 different gestures with an accuracy of $93.2\,\%$. Their work, however, focuses on the construction process and generally shows that CNNs are implementable on microcontrollers, but they do not elaborate on the resource demands of this network on the microcontroller.

We have adopted the basic idea of human gesture recognition from [13] and conduct further research on the subject. We explore how a change in the number of sequential convolutional layers (i.e., the depth of the network) affects the accuracy of the classification and the average inference time required. We also use up to 10 gestures in our work to research how the individual neural networks behave in terms of their classification accuracy as classification complexity increases. We further explore the energy consumption of the neural networks on our test hardware platform that is required per inference.

III. Materials and Methods

In this section, we discuss the software tools used for the implementation of the neural networks on the microcontroller. Next, we describe the hardware used and its specifications, on which we implemented the neural networks. Finally, we present the process applied to obtain the dataset to train the neural networks.

A. Software Tools

We use the open-source library Tensorflow [14], developed and introduced by Google, as backend in combination with Keras [15] to build and train neural networks. The Tensorflow library has an experimental module "TensorFlow Lite" which enables the conversion of standard Tensorflow neural network models into neural network models optimized for the deployment on microcontroller systems.

B. Hardware Platform

We inspect the behavior of CNNs on the target hardware platform Adafruit Feather Bluefruit Sense. This microcontroller board implements an nRF52840 SoC from Nordic Semiconductor. The SoC is built around an ARM Cortex-M4 CPU with 64MHz clock frequency and has a 32-bit floating-point unit. Additionally, the microcontroller comes with 1 MB flash, and 256 KB SRAM memory.

C. Dataset

To train and evaluate the designed CNNs models we recorded acceleration data of 10 different gestures from 5 volunteers. The individual gestures to be classified by the CNN are shown in Figure 1. We used the "LSM6DS33" tri-axial accelerometer, which is embedded on the "Adafruit Feather Bluefruit Sense", to record the gestures. We instructed our test subjects to hold the recording device in a stationary position, for a small period, before and after performing the gesture. The purpose of this is to minimize the introduction of incorrect data points into the dataset. The gestures were recorded with a sampling frequency of 26 Hz and a measurement range of $\pm\,4$G, where $1G = 9.81\frac{m}{s}$. Approximately 400 individual labeled samples were recorded per gesture. In addition to the 10 recorded and predefined gestures, we have acquired data in which movements have been performed that are not associated with any predefined gesture. This sensor data is used to allow the CNNs to classify for an "unknown" gesture. Thus if, for example, no movements are executed while running the neural network, the neural network will classify this as an unknown gesture.

We divided the recorded raw data into a training dataset, validation dataset and a test dataset with a split of $60\,\%$, $20\,\%$ and $20\,\%$ per gesture respectively. After splitting the recorded

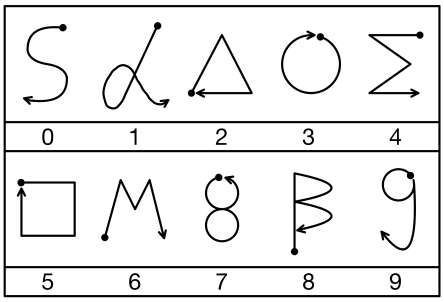

Fig. 1: List of gestures which are represented in the dataset

raw date we applied the script used in[13] for processing and synthetic dataset expansion. This script applies padding (i.e., a necessary generated number of data points is appended to the beginning or the end of a time series.) to convert the raw sensor data of the gestures to a common data length of 128 samples per gesture. With the used sampling rate of 26Hz this corresponds to a length of 4.92 seconds per gesture. Furthermore, in this script, the dataset for training the CNNs is artificially extended by augmentation to generate more data for training the CNN models. The data is artificially augmented by adding random noise, amplifying the signal, warping it in time and by slightly changing the signals offset. After the data has been preprocessed the training dataset consists of 68925 samples. The validation and test dataset hold 918 and 922 samples, respectively.

IV. Investigated Neural Network Structures

In this section, we describe the architectures of the evaluated CNNs and estimate the contributing complexity of each layer and the corresponding total complexity of the CNNs.

A. Architectures

We focus on the research of convolutional neural networks. We decided to research four different convolutional neural network architectures (Conv2, Conv3, Conv4, and Conv5). A detailed overview for each individual architecture is given in Table I, Table II, Table III, and Table IV. The Filters@Kernel size/stride column for convolutional layer or max-pooling layer indicates the number of filters applied, the filter kernel size, and with which stride size the filter is moved over the input signal, respectively. For a more general overview of the architecture of the convolutional neural networks, a visual representation of the Conv3 network is shown in Figure 2. In general, all architectures follow the same classical CNN format structure with the exception that the amount of sequential convolutional layers and max-pooling layers changes. The layer count of the networks is increased to investigate the impact of the increasing complexity of the neural networks on the inference time and the overall performance.

The input of all CNN's consists of a 128x3 long time sequence consisting of the x, y, z components of the accelerometer sensor data. All CNN's are joined by two fully-connected layers after the convolution layers for high-level reasoning. The first fully-connected layer consists of 16 neurons and the last layer consists of 2 up to 11 neurons, depending on the desired number of training gestures. The softmax function is utilized for classification in the last fully-connected layer.

TABLE I: Conv2 architecture: Consists of two convolutional layers each followed by a max-pooling layer.

#	Type	Filters @ Kernel size / stride	Output Size	Parameters	FLOPs
	Input	-	1@128x3	-	-
1	Convolution	8@4x3x1 / 1	8@128x3	104	76.80k
2	Max pooling	3x3 / 3	8@42x3	-	3.02k
3	Convolution	16@3x1x1 / 1	16@42x1	400	32.93k
4	Max pooling	3x1 / 3	16@14x1	-	0.67k
5	FC	16 neurons	16	3600	7.18k
6	FC + Softmax	n neurons	n	(16+1)n	32·n+n
Sum				4104 +(16+1)·n	120.60k +32·n+n

TABLE II: Conv3 architecture: Consists of three convolutional layers each followed by a max-pooling layer.

#	Type	Filters @ Kernel size / stride	Output Size	Parameters	FLOPs
	Input	-	1@128x3	-	-
1	Convolution	8@4x3x1 / 1	8@128x3	104	76.80k
2	Max pooling	3x3 / 3	8@42x3	-	3.02k
3	Convolution	16@3x1x1 / 1	16@42x1	400	32.93k
4	Max pooling	3x1 / 3	16@14x1	-	0.67k
5	Convolution	16@3x1x1 / 1	16@14x1	784	21.73k
6	Max pooling	3x1 / 3	16@5x1		0.24k
7	FC	16 neurons	16	1296	2.58k
8	FC + Softmax	n neurons	n	(16+1)n	32·n+n
Sum				2584 +(16+1)·n	137.97k +32·n+n

TABLE III: Conv4 architecture: Consists of four convolutional layers each followed by a max-pooling layer.

#	Type	Filters @ Kernel size / stride	Output Size	Parameters	FLOPs
	Input	-	1@128x3	-	-
1	Convolution	8@4x3x1 / 1	8@128x3	104	76.80k
2	Max pooling	3x3 / 3	8@42x3	-	3.02k
3	Convolution	16@3x1x1 / 1	16@42x1	400	32.93k
4	Max pooling	3x1 / 1	16@14x1	-	0.67k
5	Convolution	16@3x1x1 / 1	16@14x1	784	21.73k
6	Max pooling	3x1 / 1	16@5x1	-	0.24k
7	Convolution	16@3x1x1 / 1	16@5x1	784	7.76k
8	Max pooling	3x1 / 1	16@2x1	-	0.10k
9	FC	16 neurons	16	528	1.04k
10	FC + Softmax	n neurons	n	(16+1)n	32·n+n
Sum				2600 +(16+1)·n	144.29k +32·n+n

TABLE IV: Conv5 architecture: Consists of five convolutional layers each followed by a max-pooling layer.

#	Type	Filters @ Kernel size / stride	Output Size	Parameters	FLOPs
	Input	-	1@128x3	-	-
1	Convolution	8@4x3x1 / 1	8@128x3	104	76.80k
2	Max pooling	3x3 / 3	8@42x3	-	3.02k
3	Convolution	16@3x1x1 / 1	16@42x1	400	32.93k
4	Max pooling	3x1 / 1	16@14x1	-	0.67k
5	Convolution	16@3x1x1 / 1	16@14x1	784	21.73k
6	Max pooling	3x1 / 1	16@5x1	-	0.24k
7	Convolution	16@3x1x1 / 1	16@5x1	784	7.76k
8	Max pooling	3x1 / 1	16@2x1	-	0.10k
9	Convolution	16@3x1x1 / 1	16@2x1	784	3.10k
10	Max pooling	2x1 / 1	16@1x1	-	0.03k
11	FC	16 neurons	16	272	0.53k
12	FC + Softmax	n neurons	n	(16+1)n	32·n+n
Sum				3128 +(16+1)·n	146.91k +32·n+n

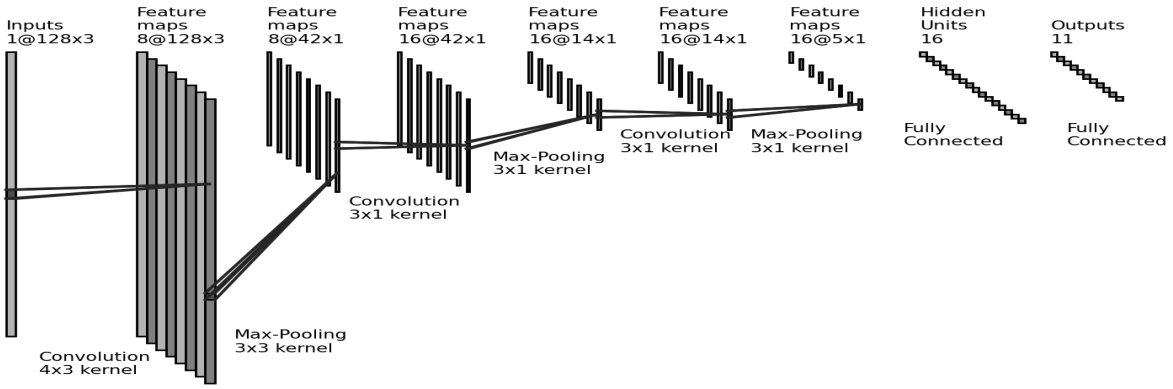

Fig. 2: Model illustration of the structure of a Convolutional Neural Network based on the selected 3-Conv layer architecture.

B. Computational Complexity

In literature, the number of floating-point operations (FLOPs) per executed inference is determined to indicate the computational complexity of neural networks. It is only an estimation since non-linear effects during operation of the CNNs are not considered and hard to model. Our proposed neural networks consist of several convolutional layers, max-pooling layers, and fully connected layers. The estimated FLOPs for these layers are determined by the following equations. Equation 1 and Equation 2 are applied to determine the FLOPs for a convolutional layer and max-pooling layer, respectively.

$$FLOPs_{CNN} = 2k_H k_W C_{in} C_{out} H_{in} W_{in} + H_{out} W_{out} C_{out} \quad (1)$$

$$FLOPs_{MaxPool} = k_H k_W H_{out} W_{out} C_{out} \quad (2)$$

Where k_H and k_W describe the size of the used filter kernel in height and width, respectively. C_{in} describes the number of feature maps of the input signal into the Conv layer and C_{out} the number of feature maps after passing through the Conv layer. H and W describe the height and width of the feature maps. From Equation 1, it is observed that the complexity of the convolutional layer depends on kernel size, the number of input and output feature maps, and the input size of the feature maps. Equation 3 below is applied to calculate the required FLOPs when a fully connected layer is used.

$$FLOPs_{FullyConnected} = 2N_{in} N_{out} + N_{out} \quad (3)$$

Where N_{in} corresponds to the number of neurons of the layer before the fully-connected layer and N_{out} corresponds to the number of neurons of the fully-connected layer [16].

V. Results and Discussion

In this section, we evaluate our proposed architectures from Section IV. First, we analyze the accuracy of the CNNs when the classification task complexity is iteratively increased followed by a comparison of the CNNs performance when trained with the entire dataset. Secondly, we investigate the inference time of the CNNs. Lastly, we evaluate the energy consumption of each CNN.

A. Accuracy

We studied the neural networks in terms of their classification accuracy, looking at two aspects. First, during the training process of the CNNs, we increased the number of gestures to be classified in the dataset and observed the effect on the accuracy of the CNNs with an increasing number of gestures to be classified. In the second study, we compared the CNNs that were trained with the trained entire dataset (10+1 gestures) by observing the accuracy, precision, recall, and F1-Score metrics of each CNN. The performance metrics we decided to apply to evaluate the CNNs are determined as follows [17]:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \cdot 100\% \quad (4)$$

$$Precision_i = \frac{TP_i}{TP_i + FP_i} \cdot 100\% \quad (5)$$

$$AvgPrecision = \frac{1}{n} \sum_{i=0}^{n} Precision_i \cdot 100\% \quad (6)$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i} \cdot 100\% \quad (7)$$

$$AvgRecall = \frac{1}{n} \sum_{i=0}^{n} Recall_i \cdot 100\% \quad (8)$$

$$F1Score = 2 \cdot \frac{AvgRecall \cdot AvgPrecision}{AvgRecall + AvgPrecision} \cdot 100\% \quad (9)$$

where i represents one kind of gesture to classify. TP, TN, FP, and FN represent the amount of true positives, true negatives, false positives, and false negatives, respectively. Precision expresses the ratio of data points that a NN predicts to be relevant and that were actually relevant. Whereas recall is applied to assess the ability of a NN to identify all relevant cases within a data set. The F1-Score is the harmonic mean resulting from the average recall and average precision values. Accuracy is applied to describe how good a model performed in general correctly identifying a gesture out of the complete dataset.

Figure 3 shows the results of the experiment on the accuracy behavior of the four CNNs with increasing classification complexity. It shows that the CNNs have relatively consistent accuracies when classifying up to 7 gestures to a large extent. As soon as 8 or more gestures are classified, a significant performance degradation of the Conv2 layer in accuracy is seen, while the other CNNs continue to achieve relatively constant and high accuracies.

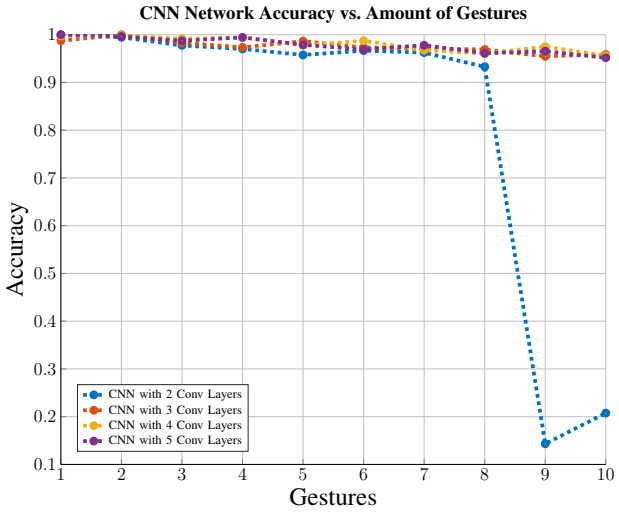

Fig. 3: Accuracy vs. Gesture amount to be classified

The performance of the CNNs when trained on the full dataset is shown in Table V. It also shows that the Conv2 network yields inadequate results in all areas of the selected performance metrics. Comparing the other three CNNs, it can be seen that with a higher depth of the network, i.e., a higher complexity, the performance marginally increases in all domains.

TABLE V: Classification performance results of CNN models when trained with complete dataset

Network	Precision	Recall	F1 Score	Accuracy
Conv2	12.4 %	18.0 %	14.7 %	20.7 %
Conv3	94.8 %	94.9 %	94.8 %	94.6 %
Conv4	95.3%	95.7 %	95.5 %	95.2 %
Conv5	95.6 %	96.2 %	95.9 %	95.6 %

Comparing the Conv3 network with the Conv5 network complexity shown in Table II and Table IV, it can be said that a $1\,\%$ increase in accuracy was achieved by a $6.48\,\%$ increase in computational complexity with respect to required FLOPs per inference. Signs of saturation of the CNNs accuracy may be observed.

These observations show that rudimentary CNNs can be used to classify human gestures. However, it also shows that if CNNs are kept too rudimentary or optimized with respect to low computational complexity, high accuracies may not be achieved.

B. Inference Time

Next, we measured how the inference time of CNNs are affected when the CNN architecture is increasing in depth and thus in complexity. We have defined the gesture recognition application as a real-time application. Since the neural networks were trained with 26 Hz training data, the hyperperiod of the system on the microcontroller was defined to be 38.46 ms. This corresponds to 26 hyperperiods per second. To meet the real-time requirements, the required computing time of the individual task within a period must not exceed the defined hyperperiod. We determined the required computation time of the MCU for each task using timestamps. The measured results are presented in Figure 4. In Figure 4, it is observable that the inference in each network requires the most computing time within a hyperperiod. This highlights yet again the high resource demands of neural networks and the associated challenges of implementing them in resource-limited systems for real-time applications. However, all deployed networks were able to meet the given constraints so that no other processes would get affected or the defined real-time hyperperiod was exceeded by the set of tasks. Furthermore, based on Figure 4, it is evident that the inference time required by the CNNs increases approximately proportionally with the increase in complexity. For better understanding, we show this observation in Figure 5. It behaves, as aforementioned, approximately proportional, since during the inference process on the microcontrollers non-linear effects occur that are difficult to model. These effects are for example caching of results in RAM or loading of weights from the program memory.

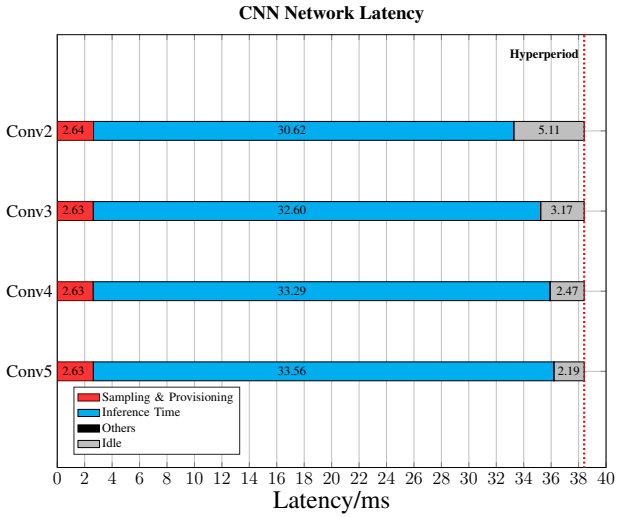

Fig. 4: Bar plot: Total latency time of one classification cycle.

We further observed the change in inference time for each CNN when the amount of gestures to be classified is increased. From Table I, Table II, Table III, and Table IV, we observe that the fully-connected layers contribute a minor proportion to the overall computational complexity of the CNNs. The last fully-connected layer is kept variable in the number of neurons and thus the number of gestures to be classified. For

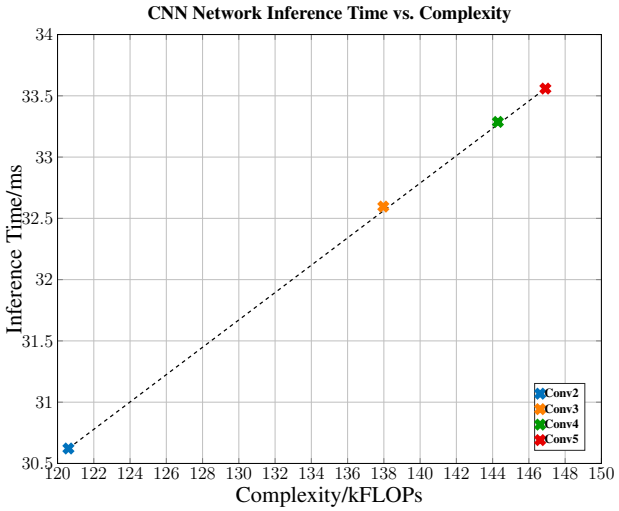

Fig. 5: Inference Time vs. Complexity of Neural Network

this layer, the amount of FLOPs increases linearly with the number of neurons. The measurement of the inference time shows that the increase in the number of neurons also results in an almost linear increase in the required computation time. The measured inference time of the Conv3 network, for example, with two neurons in the output layer required 32.50 ms while the same network with eleven neurons in the output layer required 32.60 ms. Thus, the difference between these two configurations is almost negligible compared to the computational complexity of the other network layers. This behavior was also observed for the other network configurations with great similarity and may be seen in Figure 6.

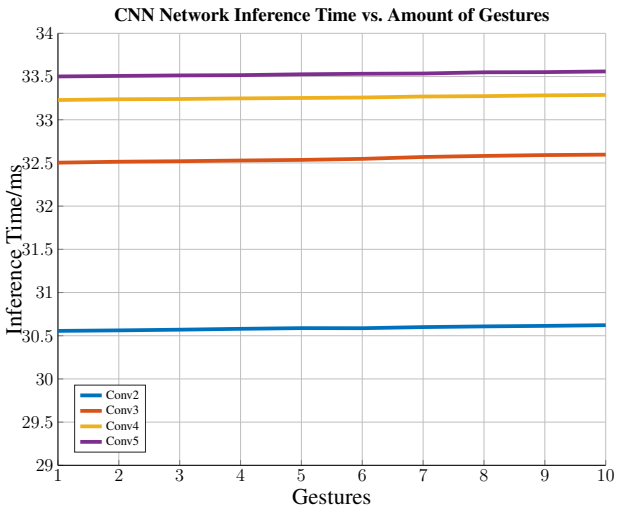

Fig. 6: Inference Time vs. amount of gestures to be classified

C. Energy Consumption

We estimated the average energy demand of the neural networks by measuring the average power consumption of the neural networks. For this purpose, we switched off other processes such as the data acquisition in the program except the inference process of the neutral network. The resulting program runs in a continuous loop with a CNN inferring on pseudo data. We did this to ensure direct comparability of the neural networks, as inequalities may occur due to different idle times. A power consumption of $P_{cont} = 52.6\,mW$ was measured for all four neural networks, when performing inferences in a continuous mode. From this, the average required power demand per hyperperiod can be estimated as follows:

$$P_{avg} = \frac{1}{t_{hyperperiod}} \int_0^{t_{inference}} P_{cont}\, dt \tag{10}$$

where $t_{inference}$ is the required inference time of the neural network. It has to be mentioned, we assume an ideal sampling process and idle state in this estimation. This means that additional processes do not require any additional energy in the system.

TABLE VI: Power requirements of each CNN over one hyperperiod

Network	Average Power P_{avg} [mW]	Average Inference Time t [ms]
Conv2	41.88	30.62
Conv3	44.59	32.60
Conv4	45.53	33.29
Conv5	45.90	33.56

From Table VI, as instinctively expected, longer inference times require higher average power demands due to the increased computational complexity of the CNNs. Thus, this points out that the implementation of neural networks introduces a compromise between accuracy and power consumption since more complex networks have longer inference times and, thus, the microcontroller is not able to spend long periods in IDLE states.

VI. Conclusion and Future Work

We have shown that the deployment of neural networks on microcontrollers involves trade-offs. For higher accuracies, a more complex network is needed, resulting in longer inference times. Increased inference times result in higher energy demand of the microcontroller system and thus lower battery runtimes when the IoT device is powered by batteries. If the focus of the application is more on the energy demand of the system, the inference times ought to be kept shorter. However, this can lead to a reduction in the accuracy of the application, since the neural network's complexity might be reduced to reach desired systems specifications.

In a future study it would be of interest to find out how much and to what degree exceeding the inference time to the desired sampling rate affects the accuracy of the classification.

Furthermore, during our research, we maintained the hyperparameters of the convolutional layers almost the same in each network. From equation 1 it may be seen that, for example, changing the kernel size of the filters or the number of filter maps will affect the number of FLOPs required per inference. Future studies have to investigate how the variation of these

parameters affects the performance of the CNNs as opposed to changing the "depth" of the network.

Acknowledgments

This publication is a result of the research of the Center of Excellence CoSA and funded by the Joachim Herz Foundation (Project: PASBADIA). Horst Hellbrück is adjunct professor at the Institute of Telematics of University of Lübeck.

References

[1] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018. DOI: 10.1109/comst.2018.2844341.

[2] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019. DOI: 10.1109/jproc.2019.2921977.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016. DOI: 10.1109/jiot.2016.2579198.

[4] G. Premsankar, M. D. Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018. DOI: 10.1109/jiot.2018.2805263.

[5] M. Capra, B. Bussolino, A. Marchisio, M. Shafique, G. Masera, and M. Martina, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, vol. 12, no. 7, 2020, ISSN: 1999-5903. DOI: 10.3390/fi12070113. [Online]. Available: https://www.mdpi.com/1999-5903/12/7/113.

[6] D. Palaz, M. Magimai.-Doss, and R. Collobert, "Convolutional neural networks-based continuous speech recognition using raw speech signal," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4295–4299. DOI: 10.1109/ICASSP.2015.7178781.

[7] T. Young, D. Hazarika, S. Poria, and E. Cambria, *Recent trends in deep learning based natural language processing*, 2018. arXiv: 1708.02709 [cs.CL].

[8] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].

[10] P. E. Novac, A. Castagnetti, A. Russo, B. Miramond, A. Pegatoquet, F. Verdier, and A. Castagnetti, "Toward unsupervised human activity recognition on microcontroller units," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, 2020, pp. 542–550. DOI: 10.1109/DSD51259.2020.00090.

[11] Y. Chen and Y. Xue, "A deep learning approach to human activity recognition based on single accelerometer," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2015, pp. 1488–1492. DOI: 10.1109/SMC.2015.263.

[12] J. Dudak, M. Kebisek, G. Gaspar, and P. Fabo, "Implementation of machine learning algorithm in embedded devices," in *2020 19th International Conference on Mechatronics - Mechatronika (ME)*, 2020, pp. 1–6. DOI: 10.1109/ME49197.2020.9286705.

[13] P. Warden and D. Situanayake, *TinyML*. Dec. 2019, ch. Chapter 11,12, pp. 279–354, ISBN: 9781492052043. [Online]. Available: https://www.oreilly.com/library/view/tinyml/9781492052036/ (visited on 12/29/2020).

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, and G. S. C. et. al, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[15] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[16] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *CoRR*, vol. abs/1611.06440, 2016. arXiv: 1611.06440. [Online]. Available: http://arxiv.org/abs/1611.06440.

[17] C. T. Yen, J. X. Liao, and Y. K. Huang, "Human daily activity recognition performed using wearable inertial sensors combined with deep learning algorithms," *IEEE Access*, vol. 8, pp. 174 105–174 114, 2020. DOI: 10.1109/ACCESS.2020.3025938.