Waist Tightening of CNNs: A Case study on Tiny YOLOv3 for Distributed IoT Implementations

Isaac Sánchez Leal
isaac.sanchezleal@miun.se
Mid Sweden University
Sundsvall, Sweden

Eiraj Saqib
Mid Sweden University
Sundsvall, Sweden

Irida Shallari
Mid Sweden University
Sundsvall, Sweden

Axel Jantsch
TU Wien
Vienna, Austria

Silvia Krug*
Mid Sweden University
Sundsvall, Sweden

Mattias O'Nils
Mid Sweden University
Sundsvall, Sweden

ABSTRACT

Computer vision systems in sensor nodes of the Internet of Things (IoT) based on Deep Learning (DL) are demanding because the DL models are memory and computation hungry while the nodes often come with tight constraints on energy, latency, and memory. Consequently, work has been done to reduce the model size or distribute part of the work to other nodes. However, then the question arises how these approaches impact the energy consumption at the node and the inference time of the system. In this work, we perform a case study to explore the impact of partitioning a Convolutional Neural Network (CNN) such that one part is implemented on the IoT node, while the rest is implemented on an edge device. The goal is to explore how the choice of partition point, quantization method and communication technology affects the IoT system. We identify possible partitioning points between layers, where we transform the feature maps passed between layers by applying quantization and compression to reduce the data sent over the communication channel between the two partitions in Tiny YOLOv3. The results show that a reduction of transmitted data by 99.8% reduces the network accuracy by 3 percentage points. Furthermore, the evaluation of various IoT communication protocols shows that the quantization of data facilitates CNN network partitioning with significant reduction of overall latency and node energy consumption.

CCS CONCEPTS

• **Computing methodologies** → *Machine learning*; **Distributed artificial intelligence**; • **Computer systems organization** → *Embedded systems.*

KEYWORDS

Internet of Things, smart camera, convolutional neural networks, CNN partitioning, intelligence partitioning

*Also with IMMS Institut für Mikroelektronik und Mechatronik-Systeme gemeinnützige GmbH (IMMS GmbH).

CPS-IoT Week Workshops '23, May 9–12, 2023, San Antonio, TX, USA

ACM ISBN 979-8-4007-0049-1/23/05.
https://doi.org/10.1145/3576914.3587518

ACM Reference Format:
Isaac Sánchez Leal, Eiraj Saqib, Irida Shallari, Axel Jantsch, Silvia Krug, and Mattias O'Nils. 2023. Waist Tightening of CNNs: A Case study on Tiny YOLOv3 for Distributed IoT Implementations. In *Cyber-Physical Systems and Internet of Things Week 2023 (CPS-IoT Week Workshops '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3576914.3587518

1 INTRODUCTION

Vision sensors alongside the development of advanced image processing algorithms such as CNNs, have enabled large-scale deployments of smart camera IoT nodes. The range of application cases is expandingthem including areas of industrial monitoring [1], smart city, and healthcare monitoring [11] among others. On the one hand, to meet the accuracy demands of such design cases, neural networks with deeper and wider architectures have been developed. On the other hand, we have IoT nodes that rely on limited resources such as computational capabilities, memory, and energy consumption. The contrasting nature of these two aspects raises the issue of where we should implement the CNN models: on the node, the edge device, the cloud server, or distributed among them.

However, the design of such nodes remains challenging. The two major contradicting aspects are on the one hand, to meet the accuracy demands of the application and on the other hand, build systems around the typically limited resources of IoT nodes such as computational capabilities, memory, and energy consumption. While the accuracy requirement leads to the development of neural networks with deeper and wider architectures this increases the resource consumption on IoT nodes, if the network is implemented on the node itself. Therefore, the CNN analysis is often facilitated in the cloud or at the edge rather than the node itself. Various CNN models such as e.g. Tiny YOLO have been developed specifically for use on resource-constrained devices. To further reduce the computational cost of CNNs optimization techniques such as pruning [9] or quantization [15] have been proposed to minimize the network size and thus reduce the memory and execution time requirements. This can enable the deployment of the models on the nodes [12]. Another option is to apply CNN partitioning [17] and thus offload part of the model execution to another node at the cost of additionally having to transfer the intermediate results. Various approaches exist for this trying to optimize execution time or energy consumption of the node. These approaches typically target one specific solution to either split the network between edge and cloud or dynamically adapt the offloading to multiple nodes. What is missing is

however a systematic review of the design space between possible partitioning points, applied communication technology and options to reduce the data exchanged between the partitions in order to identify optimal partitioning strategies.

To address this issue, we explore the design space for a given application scenario by partitioning the required CNN between the IoT node and the edge. We chose a caregiver tracking system [3] trained with the MIUN-Feet dataset [16] as an application example. In this application, the camera is located at the wheelchair with the goal to give control information to the wheelchair to follow the caregiver. This leads to a system design where real-time constraints apply to the decision making for the CNN. Therefore, offloading the data to the cloud is infeasible, while performing all calculations at the IoT node located on the wheelchair might be inefficient in terms of energy consumption. Our goal is to identify the partitioning configuration that provides the optimal energy consumption for the IoT node as the sum of local processing and communication energy. To explore the design space further, we analyze how data reduction techniques at the partition point can create a waist tightening effect to further reduce the required communication energy by limiting the data amount to be transferred.

Existing studies consider the selection of the partitioning point location in the CNN as well as options to split the amount of data. However, to the best of our knowledge, the impact of minimizing the data at the partitioning point has not been considered so far.

2 RELATED WORK

Memory, energy consumption and latency are the key constraints when designing IoT nodes. Regarding computer vision (CV) applications, balancing these requirements is important since the algorithms often feature resource hungry algorithms such as CNNs. Therefore, much attention has been spent on options to optimize the node with design cases ranging from doing all calculations in the IoT node or in the cloud as extreme endpoints. Various approaches target intermediate solutions where only part of the calculations is executed at the respective endpoint. Edge computing added another design space option, by introducing offloading capabilities closer to the IoT node. Offloading approaches partition the vision algorithm and aim for distributed execution. In Shallari et al. [13], the authors analyze the computation and communication inter effects of partitions between the node and the cloud. They target a smart camera system featuring traditional image processing methods. The same system type is analyzed in [8], where the authors studied the impact of the input data size on the selection of the partition point.

However, system partitioning applies not only to traditional systems but also to CNNs [14]. Two general approaches exist for this: an in-layer approach dividing the calculations at one layer and an approach that splits the CNN between two layers. An in-layer approach allows parallel and/or distributed execution of the calculations at this layer. One example is introduced by Zhuoran et al. [17]. They describe a technique to parallelize convolutional computations and decrease the memory usage and communication costs for initial convolutional layers. Similarly, the authors divided the feature maps and filters along the depth dimension to distribute them among Fog resources in [2]. In [10], the authors proposed an adaptive partitioning algorithm called "DINA-P" for accelerating the process of distributed inference via edge computing. This method involves dividing the convolutional layers into smaller sections and distributing the workload across multiple devices. While these approaches are able to enhance the inference time of the model, they come at the cost of additional data transfers for each part of the partitioned layer. While the communication effort is reduced for a single transmission, the parallel execution with multiple targets adds an overhead to the communication.

Partitioning the CNN model between two layers, results in two parts of the model that can be executed in different locations, requiring only one data transfer. However, depending on the partitioning location, the data amount can be quite large, resulting in high communication cost. In [5], the authors presented "Neurosurgeon" a system that considers latency and energy of inference and communication to dynamically select the optimal partition point in e.g. AlexNet and VGG for image processing. To build the system, they perform a design space analysis but do not consider the option to further reduce the data at the partition point in their analysis. The authors in [7] follow a similar strategy. Their approach introduces additional classifiers at the intermediate partition points to evaluate the impact of early exiting the original CNN model. These exit points can be executed on the node or the server (edge or cloud) and thus provide a faster inference by skipping part of the model. However, the exit-classifier also adds further computational load to the node if executed there while in other cases the communication of the intermediate results remains crucial. To mitigate this, the authors apply quantization to reduce the bit-width of the data from 32 bit to 8 bit. In [11], the authors proposed a similar framework with the goal to mitigate bandwidth availability between the device and an edge server highlighting the importance to analyze the tradeoff between latency and accuracy. Their work finds partition points based on the bandwidth and applies early-exiting to leverage the latency of all in-node inference at the cost of lower accuracy.

These approaches can benefit from further exploration of the data requirement at the partitioning points, as less data to communicate also facilitates good latency even under constraint bandwidth conditions. Therefore, analyzing the options of applying lower than 8 bit-resolution quantization together with appropriate packing as well as compression for different partition points is a crucial amendment to the existing work.

3 METHODOLOGY

We use the CV system of an autonomously driving wheelchair as base for our case study. The original work in [3] trained Tiny YOLOv4 on a custom dataset in order to detect the feet of a caregiver. The focus there was to find a system that is able to solve the tracking task. In this study, we use the same dataset, but rely on Tiny YOLOv3 (TY3) as CNN detector because it is a smaller version of the model without affecting the detection accuracy for this application. The performance of the unchanged TY3 model is used as benchmark and compare it to the partitioning options.

To study the options, we identified suitable partition points in the TY3 architecture. Essentially, we focus on layers that have a sequential data path only. Partitioning the model at points with parallel structures would require to transfer additional data since these structures reuse intermediate information [8]. Thus, we consider

the start and the end of parallel sections, where the data is compact already as partitioning points. In case of TY3, this results in five possible partition points $j = [1..5]$ that are located at the beginning of the model. In addition, we add two more partition points $j = 0$ and $j = 6$ at the beginning and end of TY3 respectively. These two partition points represent the extreme cases with all processing done locally on the IoT node ($j = 6$) and all processing done on the edge device ($j = 0$). The model architecture is depicted in Figure 1.

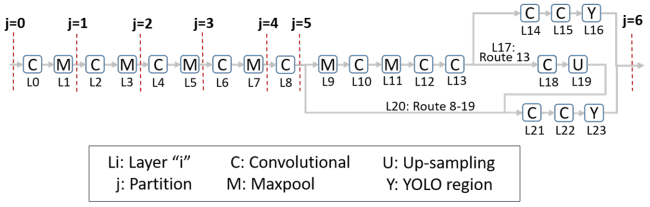


Figure 1: Abstract model highlighting the sequential data path of the Tiny YOLOv3 architecture and the resulting partitioning points.

The goal of the study is then to analyze the partitioning design space so that the node energy and overall latency is minimized. Given a CNN with $N \geq 1$ layers, the energy consumption in the IoT node E_{Node} is derived as the sum of the energy for processing E_P the different layers implemented on the IoT node and the energy for communicating data E_C from the node to the edge device at the partitioning point, which is defined as follows [13]:

$$E_{Node} = \sum_{i=1}^{j} (E_P(t_i, P)) + E_C(c_j, C) \tag{1}$$

where $i \in [1..N]$ denotes the individual CNN layers, t_i is the computation task in layer i, and j is the partitioning point. For a given partition point j, c_j represents the amount of data to be communicated with technology C. E_P and E_C are the functions for the processing and communication energy, respectively. P refers to the processing platform hardware characteristic. Similarly, the latency depends on the in-node processing, the communication duration for the given technology as well as the edge device processing.

The state of the art contains several works that explored the partitioning design space. However, to the best of our knowledge additional options to reduce the data that is sent between the two partitions have not been studied. Instead, the approaches select a partition point with minimal data as provided by the model architecture. There exist further options to reduce the data amount to be sent which could have a significant impact on the energy consumption of the IoT node and the system latency. We explore the impact of quantization to reduce the bit-precision, a suitable packing of data into bytes to avoid padding, and additional compression. While these methods result in additional processing latency and energy cost, our hypothesis is that the reduction of the data is worth the effort. As a result, we add the corresponding operations in between the layers at the partition point j as shown in Figure 2.

During an inference run, the calculations of the layers until the partitioning point j are implemented in the IoT node. Then, we quantize the output feature map d_j^b, pack it into bytes, and use ZIP-compression on the packed data to further reduce the volume

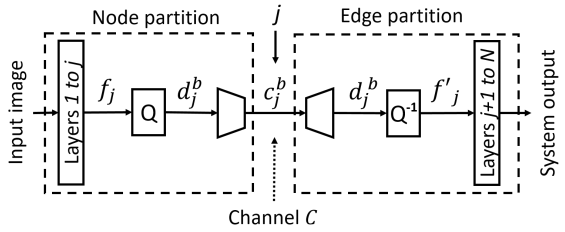


Figure 2: Schematic of a partitioning point with added functionality for handling data reduction on the node partition and unpacking the data at the edge partition.

of data c_j^b sent over communication channel C. Consequently, once arriving on the edge node, the data c_j^b is uncompressed, unpacked into d_j^b, and converted back to a floating point representation, before the remaining layers of TY3 are processed on the edge device. The communication technology has an impact on the latency and energy consumption. We consider Bluetooth Low Energy, WiFi and LTE as popular communication technologies for IoT applications, with a requirement for higher bandwidth.

In order to decrease the amount of information in feature maps at the partitioning layer, we employed quantization methods. These methods are applied to a data distribution consisting of all feature maps in a partitioning layer for all images in the dataset. The data distribution is identified by its non-symmetrical nature, with a notable skew to the right-hand side. The first quantization method is a variation of Min-Max normalization and involves reducing the information to different levels of bit-precision, ranging from 8 bits to 1 bit in increments of 1 bit. This method utilizes a shift value as a reference point and scales the data points to the dynamic range of the data distribution, as defined by:

$$d_j^b = \left\lfloor \frac{f_j - S}{max - S} \times 2^{b-1} \right\rfloor, \quad \text{with } b \in [1 - 8]. \tag{2}$$

d_j^b is the quantized data at partition point j, b denotes the bit-precision, and f_j is the floating point representation of the data at the output of layer before partition point j. S defines the shift level produced by the value of the average between the maximum (max) and minimum (min) values of the dynamic range of the feature maps and is calculated as $S = (max - min)/2$. To convert the quantized value back to a floating point representation, we apply

$$f'_j = d_j^b \times \left(\frac{max - S}{2^{b-1}} \right) + S, \quad \text{where } b \in [1 - 8]. \tag{3}$$

Upon applying the first quantization method, we discovered that as we decreased the bit width, the model's precision decreased drastically. This is attributed to the fact that the method does not consider where the majority of the data is concentrated in the distribution, resulting in quantization levels being evenly dispersed throughout the dynamic range. Consequently, opting for 1-bit quantization sets the majority of the data to 0 and only a few to 1, thereby rendering feature maps incapable of displaying features. To mitigate this, we implement a second quantization method exclusively for the 1-bit width scenario. The second quantization method

utilizes the mean value (μ) of the data distribution to determine the threshold during quantization. This, in turn, facilitates the segregation of the data distribution at the mean point of the dynamic range, enabling a more equitable allocation of data points. As a result, even with 1-bit precision, feature maps can exhibit features. The second quantization method is characterized by:

$$d_j^b = \lfloor (f_j - \mu) \,/\, max \rfloor, \quad \text{with } b = 1. \tag{4}$$

and to convert the data into back to the floating point representation

$$f'_j = \left(d_j^b \times max\right) + \mu \quad where, b = 1. \tag{5}$$

The resulting feature map f'_j is fed as input into the first layer of the CNN model after the partition point j at the edge device. $f'_j \neq f_j$ due to quantization.

Quantization reduces the amount of information in data, which often requires data packing to effectively reduce the size of the data. Data packing involves grouping multiple data into a single 8-bit number. The packing factor varies depending on the quantization level. For instance, a 4-bit quantization results in a packing factor of 2, as two 4-bit data are combined into one 8-bit data.

The interface (cf. Figure 2) requires additional processing steps per partition point and quantization method applied. Subsequently, for quantization method 1 we create 8 new versions of the model for each partition point, one per bit-precision level under test. For quantization method 2, we create one additional model per partition point. In addition, we consider partitions 0 and 6, where for the former we send the JPEG compressed input images and for the latter we send the final result after inference. This corresponds to unaltered models, that are however executed in different locations. Thus, we consider $5 \cdot 8 + 5 + 1 = 46$ model implementations. For each model variant, the initial CNN model M is partitioned and implemented in two parts, one part for implementation on the node, $M_{Node}^{1..j}$, and the other part for implementation in the edge, $M_{Edge}^{j+1..N}$. The two models together with the interface, I, between the two parts result in a transformed CNN model M':

$$M' = M_{Node}^{1..j} \cup I \cup M_{Edge}^{j+1..N} \tag{6}$$

The model M' is then re-trained to adopt the original weights to the new transformed model, which in the results section compared to the initial model M, which is TY3 in our case study. For partition point $j = 0$, we study the impact of different JPEG compression qualities as well, to also show potential reduction options at this point. Table 1 lists the combinations considered in this analysis.

Table 1: Experiment configuration.

Reduction methods	j^*	Bit precision	Compression quality	Model implementations
JPEG	0	N/A	100,50,30,20,10	Initial
Quantization 1	1-5	1-8	N/A	40 retrained
Quantization 2	1-5	1	N/A	5 retrained
N/A	6	N/A	N/A	Initial

*j = Partitioning number

4 RESULTS

We introduce partition points to TY3 by adding the interface layers to the original model as described in Section 3. All model variants were trained with the MIUN-feet dataset [16]. The values of the processing latency and energy (E_P) of the node use the implementation carried out by Ivanov [4]. It uses a Neural Compute Stick 2 hardware accelerator to run the network. The compression latency and energy were measured on a Raspberry Pi using an UNI-T multimeter tester and considering the size of the data after quantization and packaging. The communication delay and energy (E_C) were calculated using the models in [6] and the processing time in the edge node is measured on a NVIDIA GeForce RTX 3090 GPU. Table 2 shows the energy and latency measurements per TY3 block between the partition points.for Our analysis focused only on the energy consumption at the node (E_{Node}), and not the energy utilization at the edge node (E_{Edge}).

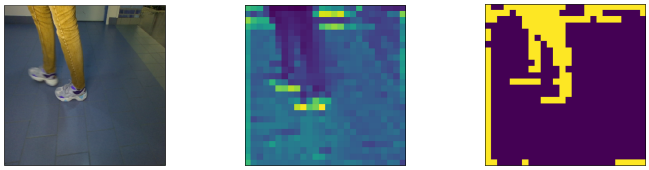
Table 2: Processing energy and latency per CNN section

j^*	E_{Node} (mJ)	L_{Node} (mS)	L_{Edge} (mS)	**Size** $(Bytes)$
0	–	–	–	2 076 672
1	14.74	8.22	3.25	2 768 896
2	6.50	3.00	1.18	1 384 448
3	3.59	1.44	0.57	692 224
4	3.21	1.29	0.51	346 112
5	3.59	1.46	0.58	692 224
6	59.05	25.14	9.92	50

*j = Partitioning number

We train a TY3 model with the MIUN-feet dataset with no partitioning as referencereference. This shows a Mean Average Precision mAP of 99.11% for Intersection over Union IoU = 0.5 and confidence threshold 0.45. This matches the performance reported by [3].

Figure 3 illustrates the impact of different bit precision to the feature maps. Figure 3b shows a initial 32-bit feature map at partition $j = 4$, while Figure 3c shows a 1-bit feature map using quantization method 2. This shows that the feet can be identified in both, despite the significant data reduction after quantization. Therefore, method two preserves enough information for later detection of the feet.



(a) Input image **(b) 32-bits** **(c) 1-bit**

Figure 3: Input image and feature maps with different bit precision.

Figure 4 shows the accuracy of all model variants with respect to the altered bit-precision and resulting data amount per partition point. Regarding the first partitioning point, $j = 0$, i.e. where the TY3 model is implemented on the edge device, the data transferred to the edge is compressed using JPEG compression. In this case,

we evaluated the compression ratios in Table 1 against the overall detection accuracy of the TY3.

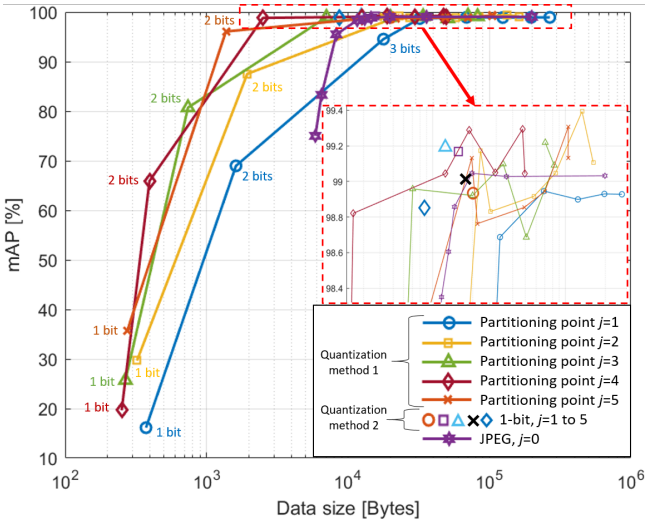


Figure 4: This graph compares the accuracy of 46 models with the size of the data, at the partition point and at the system input.

For partitioning points with $j \in [1..5]$ the quantization and compression interface is introduced, as described in the method section, resulting in 45 variations of the model. Retraining the models managed to keep most of the accuracy values close to the initial model (mAP=99.11%). In order to evaluate the model variants, we chose an accuracy threshold of 96%. Models considered for implementation should have an mAP above this threshold.

Regarding the achieved accuracy, we observe that despite retraining the models, the accuracy of the model drops due to quantization and the drop becomes significant when using quantization method 1 with a bit-precision of 3-bit or lower. However, when applying quantization method 2, we are able to retain a good accuracy even for the 1-bit-precision case. This is because quantization method 2 is fairer in selecting which levels correspond to which values.

As expected, the data c_j is reduced as we reduce the bit precision. If we compare the 1-bit precision feature maps, the data is larger when applying method 2. The better distribution of the data values using method 2 limits the compression efficiency of ZIP.

Figure 5 shows latency and energy for seven model variants, one all-at-the-server, five with distributed processing and the all-in-node case. In addition, the effects of compression for each partitioning point are part of the analysis. From Figure 4, we chose the model with JPEG compression quality 20 which shows the highest data reduction (97.8%) while maintaining a performance (98.6%) over the mAP threshold. For minimum energy consumption under the accuracy constraint of mAP $\geq$ 96%, we use the bit precision which results in the best mAP per partition point. For the first two points ($j = 1$ and $j = 2$), 1 bit quantization is used along with quantization method 2. For the remaining points ($j = 3$ to $j = 5$), quantization method 1 is used, with partition points $j = 3$ and $j = 4$ using 3 bit and partition point $j = 5$ using 2 bit.

Depending on the communication technology, the behavior in the latency and energy differs. The best design solution for latency

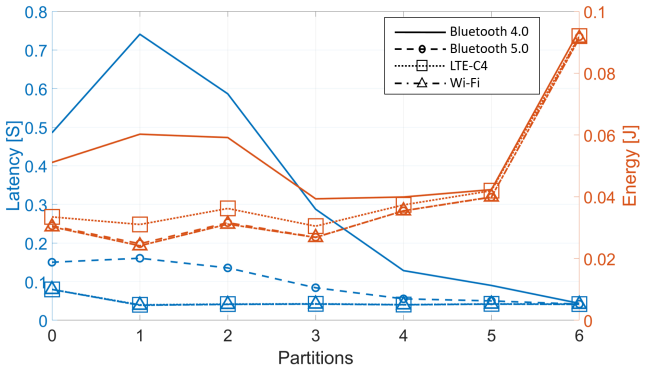


Figure 5: Latency and energy after applying partitions to the system and sending the data with different communication technologies.

is partition point $j = 5$ for Bluetooth 4.0 and 5.0. For LTE-C4 and Wi-Fi the best is the partition point $j = 1$. Regarding energy, the best partition solutions are the partition point $j = 3$ for Bluetooth 4.0 and LTE-C4 and partition point $j = 1$ Bluetooth 5.0 and Wi-Fi.

For each communication technology, several valid partitioning options exist that improve both latency and energy consumption compared to partition point $j = 0$, even with the low JPEG quality setting. Comparing to the all-in-node processing case ($j = 6$), partitioning is better from the energy perspective but only with Wi-Fi the latency is enhanced too. Table 3 summarizes the gains comparing the best solutions. The best partitioning variants use Wi-Fi to transfer the data at partition point j=1. Wi-Fi is also the best option for cases without partitioning.

Table 3: Advantages of the partitioning design solution. All variants considered use Wi-Fi.

	All In-Server	All In-Node
Energy saving	x1.26	x3.8
System speed-up	x2.05	x1.05

5 DISCUSSION

When comparing partitioned model variants to the performance of applying JPEG compression only at the node and executing the model at the edge device, the data is almost 10 times larger than the best partition solution, for a similar accuracy. There are however differences in the performance depending on the partition point and the applied bit-precision. The quantization of the feature maps in combination with compression can result in a significant reduction of data with almost unaffected accuracy in terms of mAP. However, there are other cases that do not work at all.

The solution that gives the lowest data amount is partition point $j = 5$ using 2-bit data representation. At this point, the data is reduced by 99.8%. However, this point shows an accuracy only slightly above the threshold of 96% and is the only partition point showing a good accuracy with 2-bit resolution. This clearly shows the need to explore the design space. Other partition points with high data reduction and high accuracy are the partition points $j = 3$

and $j = 4$ using a 3-bit representation. These provide an accuracy of 98.96% and 98.82% respectively while reducing the data by 98.98% for partition point $j = 3$ and 99.28% for partition point $j = 4$.

Most 2-bit and all 1-bit representations of the first quantization method show a significantly reduced accuracy, rendering the method infeasible. Analyzing these cases shows that a large portion of the feature maps will be constantly set to zero due to the distribution of the values. This could be interpreted as accidental pruning of the feature map and thus we lose too much information in these cases. Using quantization method 2, we adopt the quantization level to the average distribution of the feature maps. This shows a clear improvement of the mAP for 1-bit representations compared to the first method, but it also increases the data size. The main explanation is that the pruning effect in method 1 improves the performance of the applied ZIP compression since several values in the feature maps are set to zero. However, the data is still reduced significantly in that case as well, with good accuracy results.

The data reduction additionally frees communication bandwidth. Based on the shown reduction potential, we analyzed whether the cost to quantize, pack and compress the data is actually enhancing the overall system performance due to the reduced communication effort. Regarding the overall latency the dominant factor is communication. This is not a surprise since CV systems typically have to handle large amounts of data, resulting in longer communication activity. As for the node energy, the processing energy is the main contributor for high bandwidth communication protocols such as Wi-Fi and LTE-C4. This clearly highlights the need for design space evaluations. Our best model variant is able to achieve a better performance in terms of energy and latency despite the additional cost for the Interface I implementation compared to the extreme cases $j = 0$ and $j = 6$. This shows the potential to further enhance other tools that so far only focus on splitting the CNNs at a specific partition point rather than actively reducing the data. To further analyze the potential of such interface layers, we plan to apply it to further models and datasets. We do expect similar findings, since the steps are not depending on the structure of the CNN analyzed.

The results for 1-bit resolution using the quantization method two and the fact that method one gives the lowest data for 2- and 3-bit resolution with high accuracy show that further enhancements of our method are possible. Considering that the low data for the 2- and 3-bit resolution results from a bias in the quantization based on the input data, one could try to actively use this information to build a pruning method for the corresponding filters and combine that with an efficient 1-bit representation of the remaining data. We thus plan to investigate such an interface to minimize the data amount to transfer and thus the communication cost.

6 CONCLUSIONS

Node off-loading on CNN-based systems can be difficult due to the large amount of data handled by the system. In this paper, we showed a solution exploring quantization and compression at different points in the CNN. We were able to reduce the amount of data between partitions in a Tiny YOLOv3 network by up to 99.8%.

In an IoT scenario, where a feet positioning system is used as a control signal for a powered wheelchair, we were able to reduce the energy consumption at the node by partitioning the model between node and edge accuracy. Additionally, we decreased the overall latency of the system when compared to sending JPEG compressed images directly to the edge service, by identifying the best communication option. In future work, we plan to generalize the approach to other CNN networks and investigate distribution-specific quantization methods in combination with pruning and compression of feature map connections between CNN layers to retain accuracy.

REFERENCES

[1] Jovani Dalzochio, Rafael Kunst, Edison Pignaton, Alecio Binotto, Srijnan Sanyal, Jose Favilla, and Jorge Barbosa. 2020. Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges. *Computers in Industry* 123 (2020), 103298. https://doi.org/10.1016/j.compind.2020.103298

[2] Swarnava Dey, Arijit Mukherjee, Arpan Pal, and P Balamuralidhar. 2018. Partitioning of cnn models for execution on fog devices. In *Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing (CitiFog'18)*. ACM, Shenzhen, China, 19–24.

[3] Cristian Vilar Giménez, Silvia Krug, Faisal Z Qureshi, and Mattias O'Nils. 2021. Evaluation of 2D-/3D-Feet-Detection Methods for Semi-Autonomous Powered Wheelchair Navigation. *Journal of Imaging* 7, 12 (2021), 255. https://doi.org/10.3390/jimaging7120255

[4] Matvey Ivanov. 2021. *Embedded Machine Learning Demonstrator*. Bachelor Thesis. TU Wien. https://publik.tuwien.ac.at/files/publik_296007.pdf

[5] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629. https://doi.org/10.1145/3037697.3037698

[6] Silvia Krug and Mattias O'Nils. 2019. Modeling and comparison of delay and energy cost of IoT data transfers. *IEEE Access* 7 (2019), 58654–58675. https://doi.org/10.1109/ACCESS.2019.2913703

[7] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. ACM, London, United Kingdom, 15 pages. https://doi.org/10.1145/3372224.3419194

[8] Isaac Sánchez Leal, Irida Shallari, Silvia Krug, Axel Jantsch, and Mattias O'Nils. 2021. Impact of input data on intelligence partitioning decisions for IoT smart camera nodes. *Electronics* 10, 16 (2021), 1898. https://doi.org/10.3390/electronics10161898

[9] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. (2016). https://doi.org/10.48550/arXiv.1608.08710 arXiv:arXiv preprint arXiv:1608.08710

[10] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. 2020. Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. IEEE, Toronto, ON, Canada, 854–863. https://doi.org/10.1109/INFOCOM41043.2020.9155237

[11] Yali Nie, Paolo Sommella, Marco Carratù, Matteo Ferro, Mattias O'nils, and Jan Lundgren. 2022. Recent Advances in Diagnosis of Skin Lesions using Dermoscopic Images based on Deep Learning. *IEEE Access* 10 (2022), 95716–95747. https://doi.org/10.1109/ACCESS.2022.3199613

[12] Rick Pandey, Sebastian Uziel, Tino Hutschenreuther, and Silvia Krug. 2023. Towards Deploying DNN Models on Edge for Predictive Maintenance Applications. *Electronics* 12, 3 (2023). https://doi.org/10.3390/electronics12030639

[13] Irida Shallari, Silvia Krug, and Mattias O'Nils. 2020. Communication and computation inter-effects in people counting using intelligence partitioning. *Journal of Real-Time Image Processing* 17, 6 (2020), 1869–1882.

[14] Irida Shallari, Isaac Sánchez Leal, Silvia Krug, Axel Jantsch, and Mattias O'Nils. 2021. Design Space Exploration for an IoT Node: Trade-Offs in Processing and Communication. *IEEE Access* 9 (2021), 65078–65090. https://doi.org/10.1109/ACCESS.2021.3074875

[15] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2020. RCE-NN: a five-stage pipeline to execute neural networks (cnns) on resource-constrained iot edge devices. In *Proceedings of the 10th International Conference on the Internet of Things*. ACM, Malmö, Sweden, 1–8.

[16] Cristian Vilar. 2021. MIUN dataset för semi-autonom manövrering av eldrivna rullstolar. https://doi.org/10.5878/k44d-3y06

[17] Zhuoran Zhao, Kamyar Mirzazad Barijough, and Andreas Gerstlauer. 2018. Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2348–2359. https://doi.org/10.1109/TCAD.2018.2858384