Filling the Gap: Fault-Tolerant Updates of On-Satellite Neural Networks Using Vector Quantization

Olga Kondrateva*, Stefan Dietzel†, Maximilian Schambach†, Johannes Otterbach† and Björn Scheuermann‡
*Humboldt-Universität zu Berlin, Berlin, Germany, kondrate@informatik.hu-berlin.de
†Merantix Momentum GmbH, Berlin, Germany, {stefan.dietzel, maximilian.schambach, johannes.otterbach}@merantix.com
‡Technical University of Darmstadt, Darmstadt, Germany, scheuermann@kom.tu-darmstadt.de

***Abstract*—The use of small, low-Earth-orbit satellites enables many novel Earth observation use cases due to their cost efficiency. To cope with the challenging communication environment, machine learning algorithms, such as artificial neural networks, can be applied onboard the satellites. They help to prioritize or preprocess sensor measurements and to reduce the amount of data transmitted to Earth. However, transferring and updating machine learning models to suit changing prioritization requirements poses a number of challenges in itself due to short contact times of satellites with ground stations and lossy communication links. We propose a new transmission mechanism for model updates that retains high performance even when these updates have been only partially transmitted. We achieve this by approximating missing model weights using a vector quantization approach. Using a support structure of quantized vector indices, we can approximate the model with a small amount of data, which is transmitted first, while retaining a high performance. The model performance can then be incrementally improved, as more exact model weights are transmitted to the satellites. Our evaluation shows that this approach significantly outperforms existing baselines.**

I. Introduction

Small satellites are widely used for communication [1] and Earth observation purposes in fields as diverse as disaster management [2], science [3], [4], and economics [5]. Their small size as well as low manufacturing and launch costs allow for rapid development with a short time to market [6]. In addition, they bring flexibility not offered by large and expensive geostationary (GEO) satellites. Different small satellite classes exist [1], the most popular being nanosatellites following the CubeSat standard. A CubeSat is composed of one or several 1U (corresponding to $10\times10\times10\,\mathrm{cm}^3$) units with a mass of at most 2 kg each [7]. The CubeSat standard's main advantage is the availability of commercial off-the-shelf (COTS) components, which allow to considerably simplify the development. Moreover, COTS components are often several generations ahead of their radiation-tolerant counterparts [8], [9] that are traditionally used for larger and more expensive satellites. Thus, CubeSats can leverage more advanced technologies at the cost of reduced reliability.

Communication issues: Large constellations of CubeSats are required to compensate for the limitations and reliability issues resulting from their small mass and the use of COTS components [10], [11]. Such constellations can generate terabytes of data per day, which cannot all be transmitted to Earth due to the limited communication capabilities of CubeSats, which usually operate in low Earth orbit (LEO). Whereas GEO satellites appear static over a fixed position on Earth, LEO satellites are typically only visible four to five times per day for periods about ten minutes [12]. Thus, the collected data needs to be buffered until the next orbital pass. Also, the availability of ground stations is limited. Costs for deployment of a mission-dedicated ground station segment or using a commercial ground segment operator are often not affordable for common small satellite missions [13], [12]. In addition, licensing procedures can cause considerable delays due to the lack of uniform standards, as evidenced by Amazon Ground Stations not being able to operate for almost a year due to licensing issues [14]. Finally, according to the simulation results reported by Denby et al. [15], downloading raw data does not scale as the constellation population grows. Thus, simply increasing the number of ground stations does not solve the problem in the case of large constellations.

Another important problem is that the low mass of CubeSats limits their power budget and antenna size [1]. CubeSats usually use the X band for downlink communication, achieving data rates from hundreds of Mbps up to several Gbps [16], [17]. For the uplink, however, lower S band frequencies are used, resulting in data rates of only around a few hundred kbps [12]. Finally, the rapid development of new sensor technologies should be taken into account. The current growth of sensor data generation rates is not compensated for by a corresponding growth of download speed [18]. In practice, this gap becomes even larger due to high packet loss, which is reported by some satellite missions to be up to 88% [19].

Therefore, increasing the autonomy level on the satellite side and reducing the dependency of small satellite systems on communication with Earth are crucial for future development of small satellite systems.

Onboard processing using neural networks: In the last years, we have seen a growing interest in using neural networks for various onboard processing tasks [18], [20]. For example, neural networks can be used for onboard classification of payload data ensuring that only important data is transmitted

ISBN 978-3-903176-57-7 ©2023 IFIP

to Earth [8], [18]. Moreover, they can be used for satellite operation systems [18], [20]. Examples include collision avoidance [21], automatic pose estimation [22], and communication [23]. Substantial laboratory tests [8], [22], as well as in-orbit demonstration missions [24], have been performed showing that currently available solutions for running neural network models on embedded devices meet the mass and power restrictions of CubeSat missions.

Neural networks are achieving excellent performance in a wide range of fields, e. g., computer vision, medical science, and many others. However, their application onboard satellites with limited resources poses new challenges. To achieve state-of-the-art performance, neural networks require millions of parameters and extensive training that cannot be performed on CubeSats. Thus, neural network models need to be trained on Earth before they are transmitted [9].

Once the model is operational, periodic retraining on Earth and subsequent updates of the model parameters on the satellite will be required. One reason is the lack of training data on Earth, which poses a real concern for projects leveraging novel, mission-dedicated sensors [9], [24]. In this case, a model has to be initially trained with synthetic data and subsequently updated as more real-world sensor data becomes available. A similar problem arises when using neural networks for satellite operation systems that rely on measurements or environment monitoring results. In addition, model updates are required when adapting to new classification requirements (e. g., change of target classes), which can be expected to happen quite often if a satellite is carrying multiple sensors or in the case of commercial constellations providing Earth monitoring as a service. Depending on a particular use case, the model already in use may be retrained to improve its performance, or it may be replaced by a completely new model.

Model update challenges: Given the communication limitations – namely, short contact times, low bandwidth, and high packet loss rates – model changes become a non-trivial task. In many cases, multiple orbital passes are required to transmit a new model. Moreover, the passes will have different quality, since the maximum data rate can only be achieved when the satellite is at the shortest path [12]. According to our simulation results, which are presented in section Section IV, 51 hours are required to transmit a common neural network model (VGG16) containing around 15 million parameters given a typical uplink data rate. Even in the case of smaller model sizes of around 1 million parameters, more than 3 hours are required, even when assuming that data is transmitted continuously. According to our simulation results, however, the period between two passes can last up to 12.5 hours, which contributes significantly to the effective transmission time.

This leaves us with two naïve baseline options: 1) We could delay deployment of the new model until it is fully transmitted. This, however, may be prohibitive in the case of time-critical applications (e. g., wildfire monitoring or disaster management), a considerable change in classification task, or if the update concerns satellite operation systems. 2) The new model could already be deployed using a subset of the transmitted model parameters, resulting in a severely degraded performance, even if only a small subset of parameters is missing.

Model compression techniques, such as those proposed in [25], [26], [27] and many others, may seem to be an obvious solution. But they do not address the fundamental problem, namely, how to cope with model updates that have only been partially transmitted. In addition, the majority of the proposed compression approaches are based on repeated re-training of a neural network and therefore require extensive computations to maintain state-of-the-art performance. This is especially critical in cases of time-sensitive applications or where the training data is not available in a single centralized location (e. g., in federated learning).

Our approach: To address these problems, we propose a novel transmission mechanism that updates the new model parameters incrementally in a way that allows to achieve reasonable classification results even with a partially transmitted new model. To this end, we use a model-independent codebook that we generate using a vector quantization approach and assume to be preloaded on the satellite. Each time the model is updated, we first transmit an index structure that references the codebook's entries to approximate the new model's parameters. Afterwards, we incrementally transmit the exact updated model parameters. Since the index structure already encodes the new network's main characteristics, sufficient classification performance is achieved much earlier when compared to transmitting batches of exact model weights without using the index structure.

Our contributions can be summarized as follows:

- We propose a novel transmission approach that allows to update a neural network incrementally.
- We demonstrate that a vector-quantization-based codebook can be used across models to achieve efficient, incremental updates.
- We evaluate our approach for a variety of classification datasets and model architectures, as well as in a realistic communication scenario using the ESTNeT [28] simulator.

The remainder of this paper is organized as follows. In Section II, we summarize existing work with different optimization goals. We provide a detailed explanation of our approach in Section III. In Section IV, we evaluate our approach using a range of common neural network models, and we summarize our conclusions in Section V.

II. Related Work

Efficient over-the-air transfer of neural networks has not been extensively investigated in the literature. As a seemingly straight-forward solution, *compression* of neural networks by model pruning, quantization, low-rank factorization, and knowledge distillation has been studied in the past using a variety of approaches. However, these mainly focus on memory and computational efficiency.

Low-rank decomposition techniques achieve model compression by factorizing the weight matrices of neural networks into products of lower-dimensional ones, drastically reducing their memory footprint [29], [30]. Similar techniques also exist in

the case of CNNs used for image data, where convolution filters are decomposed into tensor products of lower-rank filters [31].

In model pruning, the number of parameters of a large neural network is reduced, e. g. by cutting connections between neurons [25], resulting in a sparse architecture, or by eliminating full sets of weights corresponding to a specific feature map [32], [33]. Furthermore, pruning can be combined with existing compression techniques such as arithmetic coding, Huffman coding, or by using hash tables [34], [35], [36]. These pruning techniques are able to achieve similar performance as compared to their dense counterparts, yet at a much lower memory and computational footprint. However, they usually come with a severely increased training complexity – often due to an iterative training-pruning procedure – limiting their application in time-critical scenarios (e. g., wildfire monitoring or disaster management).

Similarly, large neural networks can be used as teachers for a smaller student network, known as knowledge distillation [37]. Here, the prediction results of a (trained) large neural network with very good performance is used to guide the training of a much smaller model, the so-called student. Knowledge distillation has achieved impressive results [38], [39], in particular in recent advances in self-supervised training in computer vision [40], [41]. Nevertheless, distillation again requires complex and time-consuming training strategies in order to compress the large network into a smaller one.

Jankowski et al. [42] combine several of the previously discussed approaches – namely pruning and distillation – into a deep training and transmission scheme, dubbed AirNet, adapted to small bandwidth and power consumption constraints. However, they focus on noisy channel transmission which they counter by using noise injection during the network's training in order to make the models more robust, while our work's focus lies on limited availability of transmission windows and time-critical applications. In a similar line of work, Fujihashi et al. [43] extend the AirNet scheme to the federated setup which is, however, not applicable to the considered case due to the lack of on-satellite training data availability.

Quantization techniques for neural networks have also been investigated. Usually, the weights of neural networks are represented using 32 bit precision floating point. To this end, various scalar quantization methods have been proposed to compress and accelerate neural networks [26]. For example, deterministic rounding and binarization techniques have been explored [44], [45], [46], [47], [48], [49]. Furthermore, stochastic quantization methods, such as random rounding [44], [48], and probabilistic quantization methods [50], [51] have been proposed.

Assuming 32 bit values, the maximum compression rate achievable by scalar quantization is 32. Higher compression rates are achievable by vector quantization. Gong et al. [27] achieve 16–24 × compression rate with only 1% loss of accuracy by applying product quantization to fully connected layers. Wu et al. extend their approach to convolution layers [52].

Finally, in order to make use of more efficient hardware, such as field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs), neural network architectures have to be translated to a reduced precision, often using 8 bit or fewer. While a wide variety of works exist, these face their own unique challenges, such as using fixed-point arithmetic or weight scaling, making specific hardware-adapted architectures necessary [53], [54], [55], [56].

III. Fault-tolerant Updates of Neural Networks

We propose an efficient incremental model update mechanism for use in Earth observation (EO) settings using LEO satellites. Specifically, we take into account communication challenges of neural network updates or even full model replacements on the satellite. Our approach ensures reasonable behavior even in the case of a partially updated state.

We consider two phases of satellite operation shown in Figure 1: the *initialization phase* and the *operational phase,* which we explain in the following.

The initialization phase is shown on the left in Figure 1. It will typically happen before launch. a) The original neural network model NN_1 is trained on Earth and stored on the satellite. b) We calculate a separate data structure and store it on the satellite to enable efficient incremental updates during the operational phase. More specifically, we compute a codebook CB_{NN_1} consisting of entries in the form of weight vectors that approximate commonly occurring groups of the neural network model's weights.

The codebook entries are calculated using a vector quantization approach that clusters model weights, represented as flat vectors, and stores the cluster centroids in the codebook with a corresponding integer index. Our mechanism's performance improvements are based on the codebook allowing for a well-performing compression and reconstruction of different neural networks NN_i despite the codebook being derived from NN_1. Note that the codebook structure only serves to improve later model updates – it is not used to compress the initial model as it is stored on the satellite.

Our main contribution is to leverage the codebook for efficient incremental updates during the operational phase, as shown on the right in Figure 1. As classification requirements change, the new model NN_2 is first trained on Earth. Then, we compute the approximation of the NN_2's weights using the codebook CB_{NN_1}. As we will show, CB_{NN_1} allows a suitable approximation of NN_2's model weights, which eliminates the need to transmit a new codebook to the satellite.

1) We approximate NN_2 as NN_2° using codebook CB_{NN_1} by mapping NN_2's weights to their closest representation in the codebook. 2) This results in an index structure $\mathcal{I}_{\mathrm{NN}_2}$, where the index entries are pointers to the codebook entries. 3) We finally transmit this index structure to the satellite. Since the codebook CB_{NN_1} is already available on the satellite, the index structure is sufficient to reconstruct NN_2°. The index structure's compact size allows for efficient transmission, resulting in a fast but approximate reconstruction of the new model via NN_2°. After the index structure is transmitted, the exact weights of NN_2 follow. 4) We first transmit the weights of the first convolution and the last fully connected layers, as well as the batch normalization layers. Generally, these layers contain

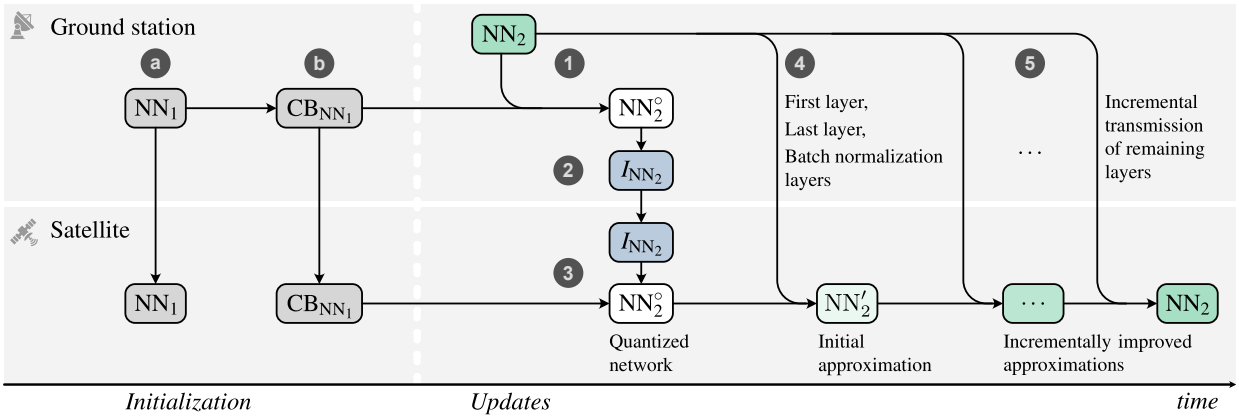


Fig. 1: Overview of the initialization phase and updates during the operational phase.

a comparatively small number of parameters, e. g., $0.17\,\%$ of the total number of parameters in case of the VGG-16 model trained on CIFAR-10, which we use in our evaluation. However, they have a considerable impact on a neural network's performance. 5) All other weights are transmitted in random order to avoid large gaps. The order of the weights as well as the architecture of the neural network being transmitted are assumed to be system parameters that are stored on the satellite during the initialization phase. Whenever a new batch of exact network weights arrives at the receiver, the values of the codebook approximations in NN_2° are replaced by these weights.

At each point during the incremental update process, the neural network at the satellite consists of a mixture of exact and approximate weights, the latter given by the values of the centroids. The performance of the network gradually improves as more and more exact weights arrive at the satellite. Because the approximation NN_2° already encodes the new network's main characteristics, sufficient classification performance is achieved much earlier when compared to transmitting batches of exact model weights without using the index structure. Steps 1–5 can be repeated to transmit further model updates NN_3, NN_4, and so forth. Next, we describe the major components of our approach – vector quantization and codebook computation – in more detail.

A. Vector quantization

In order to calculate the codebook *CB*'s entries, we use a vector quantization technique. Quantization [57] is a lossy compression method used to reduce an input space's cardinality by partitioning it into a number of non-overlapping subspaces. To achieve compression, all values of a given subspace are represented by a single value: its centroid index. The set of all centroids defines the codebook *CB*. We first explain the concept of vector quantization in general before providing details on how we apply it to neural networks.

In vector quantization, a vector $v \in \mathbb{R}^D$ is mapped to a centroid $c_i \in \mathbb{R}^D$ from the codebook $CB = \{c_1, \ldots, c_K\}$ of size K using a standard clustering algorithm. Compression is accomplished by replacing the input vector with a single value representing the nearest centroid's index. We achieve a higher compression rate by increasing the vectors' dimensionality. However, additional complexity required for computing the codebook limits this approach's applicability to high-dimensional data.

Applying product quantization [58] can cope with scalability issues introduced by high dimensionality. The idea is to decompose the input vector space into a Cartesian product of subspaces and to quantize each subspace separately. The input vector v is split into m disjoint subvectors $\{x_1, \ldots, x_m\}$ of dimension $D_{split} = D/m$, where D is assumed to be divisible by m. Each subvector x_i is quantized separately using the codebook CB_i associated with the corresponding subspace. The concatenated subcodewords represent the input vector.

While using vector quantization codebooks to reduce the memory footprint [27] and computational requirements [52] of models has been discussed in literature, we apply the concepts in a novel way to design an efficient transmission mechanism.

B. Codebook computation

We now describe how to use product quantization to compute the codebook *CB* for a neural network. Recall that the goal is to derive a list of centroids that are then used to approximate sets of similar vectors. We first derive vectors by flattening and grouping the neural network's weights. Second, we apply a clustering approach to calculate the centroids, which compose the codebook. Starting with the first step, we explain the simpler case, a fully connected layer, before extending the concept to convolution layers, which are commonly used for Earth observation (EO) tasks.

Consider a fully connected layer l of size $D_{in} \times D_{out}$, where D_{in} and D_{out} denote the number of input and output feature maps, respectively. Let $W_l \in \mathbb{R}^{D_{in} \times D_{out}}$ denote the weight matrix of the corresponding layer. First, we partition W_l into M sub-matrices of size $D_{split} \times D_{out}$, assuming $D_{in}/D_{split} = M$:

$$W_l = \begin{bmatrix} W_l^1 \\ \vdots \\ W_l^M \end{bmatrix} \tag{1}$$

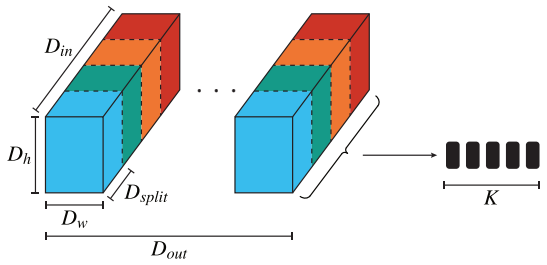


Fig. 2: Example cut of a convolution kernel for $M = 4$.

Normally, when product quantization is applied [58], a separate codebook $CB^m \in \mathbb{R}^{K \times D_{split}}$ is computed for each sub-matrix W_l^m, $m = 1, \ldots, M$ to minimize the quantization error. However, we assume that the codebook is computed based on the initial neural network NN_1 rather than the new neural network NN_2 that is being quantized for transmission. Therefore, no obvious mapping between codebooks and subspaces necessarily exists, so we combine the subspaces into a single space W_l^* of layer l,

$$W_l^* = \left[W_l^1 \cup \ldots \cup W^M\right] . \tag{2}$$

Convolution layers, core building blocks of convolutional neural networks (CNNs), can be used to compute a codebook in a similar way, as shown in Figure 2. Consider a convolution layer of size $D_h \times D_w \times D_{in} \times D_{out}$, where D_h and D_w denote the spatial dimensions of the convolution kernel and D_{in} and D_{out} the number of input and output features, respectively. As with the fully connected layer, we first split each convolution kernel $W_l \in \mathbb{R}^{D_h \times D_w \times D_{in}}$ along its channel dimension into several sub-matrices of size $D_h \times D_h \times D_{split}$. Then, we combine the vectors extracted from the kernel for a fixed spatial position $(i, j) \in \{1, \ldots, D_h\} \times \{1, \ldots, D_w\}$ into a single matrix W_l^* in the same way as described for the fully connected layers.

Note that each kernel dimension can potentially be chosen for this splitting procedure. Following related work [27], [52], we restrict our consideration to splitting along the input channel dimension, as detailed above.

The matrices $W_1^*, \ldots, W_L^*$ of a neural network consisting of L layers are concatenated into a single matrix W_{all}^*:

$$W_{all}^* = [W_1^* \cup \ldots \cup W_L^*] \tag{3}$$

We compute a single codebook $CB \in \mathbb{R}^{K \times D_{split}}$ using K-means clustering. That is, we find a partition of W_{all}^* into clusters,

$$W_{all}^* = S_1 \sqcup \ldots \sqcup S_K , \tag{4}$$

where $\sqcup$ denotes the disjoint union, with centroids $c_1, \ldots, c_K$, by minimizing distance d of all vectors $v \in W_{all}^*$ to their respective centroids,

$$\min \sum_{k=1}^{K} \sum_{v \in S_k} \left\| v - c_k \right\|_2^2 . \tag{5}$$

The final codebook CB comprises the set $\{c_1, \ldots, c_K\}$.

C. Index structure computation

We next compute an index structure for a neural network based on the codebook CB, which consists of K vectors of length D_{split} each. We assume the codebook CB to be available both at the satellite and the ground station.

We first split the weight matrices into vectors of length D_{split} as described in Section III-B. Each vector v is quantized using its nearest centroid c_i in the codebook CB,

$$\hat{v} = c_i , \quad i = \arg\min_{c \in CB} \|v - c\|_2^2 . \tag{6}$$

To compress the neural network, all centroids are replaced by their codebook indices. The index structure's size for a fully connected layer is

$$\log_2(K) D_{out} \frac{D_{in}}{D_{split}} , \tag{7}$$

where $\log_2(K)$ is the number of bits needed to encode the indices (i. e., the size of the codebook) and $D_{out} \frac{D_{in}}{D_{split}}$ corresponds to the number of vectors, assuming that the layer is split along the input dimension. Similarly, a convolution layer's index structure size is

$$\log_2(K) D_{out} D_h D_w \frac{D_{in}}{D_{split}} . \tag{8}$$

Thus, the index structure size depends on the choice of the size of the codebook K and vector length D_{split}.

IV. Evaluation

Our main evaluation metric is the classification accuracy achieved when parts of a changed model are available at the satellite. Therefore, we implement our algorithms using a realistic simulation and machine learning environment. We report Top-1 accuracy, that is, the percentage of all classification tasks for which the neural network selected the correct class as most likely classification. All experiments are conducted ten times with different random seeds; the error bars show 95% confidence intervals.

First, we evaluate the selected models independent of communication effects to isolate *our mechanism's influence on accuracy.* We evaluate the accuracy achieved for different percentages of the original weights that we assume to be received by the satellite. For our approach, missing weights are replaced by the quantized values as described in Section III, and for the baseline, missing weights are replaced by zeros. Second, we evaluate how *different parameter choices,* namely the vector length D_{split} and the number of the codebook centroids K affect the performance of our approach and the size of the index structure. For the vector length, we consider values of 4, 8, and 16, and for the codebook size, we consider 64, 128, 256, and 256. Third, we use the ESTNeT simulator [28] to assess the benefits of the proposed transmission mechanism in a *realistic communication scenario* by comparing the required transmission times of our approach versus the baseline.

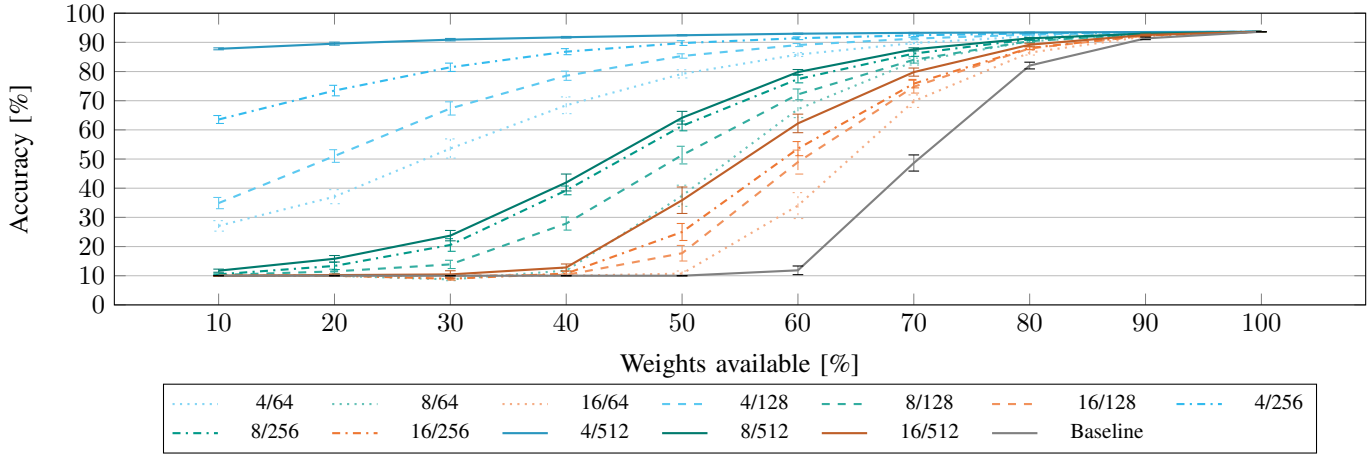


Fig. 3: Results for the VGG-16 model trained on CIFAR-10 for different vector length/codebook size combinations.

A. *Comparison of different models*

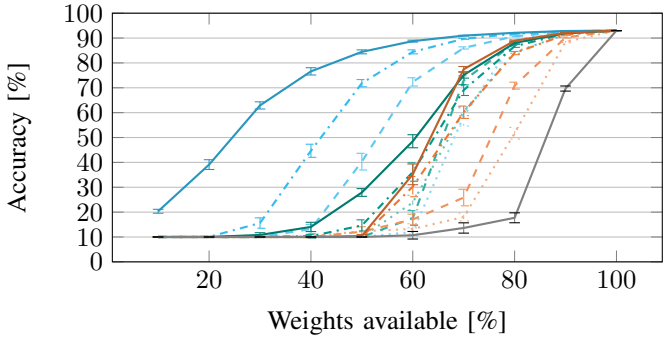
We use a wide set of network architectures pre-trained on different datasets; all models are implemented using Keras with TensorFlow as backend. To demonstrate our approach's ability to reuse codebooks from different neural networks, the codebook for all experiments is computed from VGG-16 trained on the Imagenet dataset [60].

In Figure 3, we present the results for the VGG-16 model [61] trained on the CIFAR-10 dataset [62] for different values of vector lengths and number of centroids. The x-axis shows the percentage of weights available at the satellite and the y-axis shows the Top-1 accuracy. We denote parameter combinations as: ⟨vector length⟩/⟨codebook size⟩. The accuracy of the baseline starts improving only after 60% of the weights become available and reaches values within 10% of the model's maximum possible accuracy of 93.66% after 90% of the weights are available. The performance of our approach considerably outperforms the baseline for all parameter combinations. The results can be grouped according to the vector length. As expected, the performance improves considerably when choosing smaller vector lengths. The reason being that we use the Euclidean distance to compute the index structure, which is known to be problematic when the dimensionality increases. Nevertheless, even the largest vector length of 16 significantly outperforms the baseline. A notable improvement in accuracy is already seen when 60% of the original weights become available. Depending on the number of centroids, the performance of our approach for a vector length of 16 lies between 34.04% and 62.21%, whereas the baseline reaches only 11.85%. We can achieve a further noticeable improvement when setting the vector length to 8. Then, it is possible to achieve accuracies within 10% of the model's highest possible accuracy with only 70% for the original weights. As expected, the best performance is reached when the vector length is set to 4. The best overall result is achieved for a codebook size of 512 centroids. In this case, the accuracy of 87.78% is reached with only 10% of the original weights available. Furthermore, it can be seen that, compared to the vector lengths of 8 and 16, a higher improvement can be achieved, when the number of centroids increases. Also, the confidence intervals are smaller for a vector length of 4, which means that our approach gets more stable for smaller vector lengths.

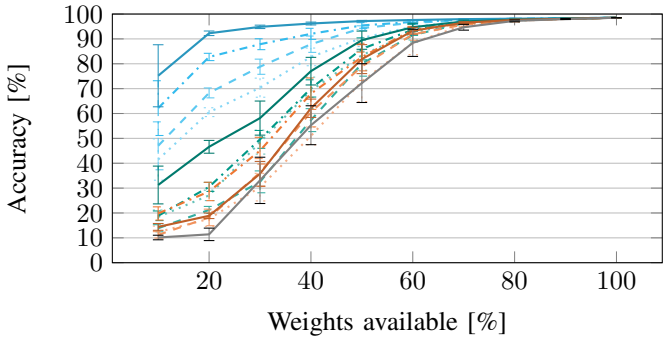
Next, we present results for the RESNET-50 model [63] in Figure 4a. Compared to VGG-16, this model is larger and has a more complicated structure. The difference is easily seen when looking at the two models' baselines. Whereas, in the case of VGG-16, it is possible to reach 82.05% accuracy with 80% of the original model weights, RESNET-50 achieves only 17.71%. Although the results of our approach are influenced in a similar way, it still significantly outperforms the baseline. In contrast to the VGG-16 model, the results cannot be clearly separated according to the vector length, and therefore, the number of centroids becomes more important. For example, the parameter combination $16/128$ shows a better result than $4/64$ for 60% of the original weights available.

To evaluate the performance of our approach for smaller models, we consider LeNet5 [64] trained on MNIST [65] (Figure 4b). Again, we compare the baselines of LeNet5 and VGG-16 to estimate the complexity of the model and the dataset. Evidently, we are dealing with a much simpler model, since 60% of the weights are sufficient to achieve the accuracy of 88.41%. Although the baseline performs much better, our approach considerably improves the accuracy. For example, the accuracy of 92.26% can be achieved with only 20% of the original model weights using the parameter combination $4/512$. Similar to RESNET-50, the results for the vector lengths 8 and 16 cannot be clearly separated into groups. We hypothesize that this behavior is due to the model's small size making it more susceptible to quantization effects. The larger size of the confidence intervals compared to both VGG-16 and RESNET-50 confirms this assumption. The results for a codebook size of 512 centroids, however, still show reliable performance, significantly outperforming the baseline.

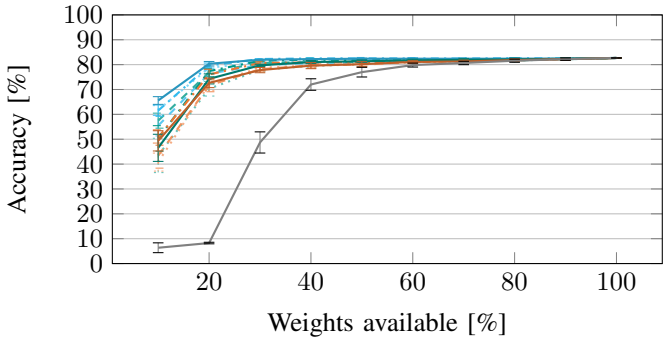
To show that our approach is applicable to satellite imagery, we use the model proposed by Makantasis et al. [59] trained on



(a) ResNet-50 model trained on CIFAR-10.



(b) LeNet5 model trained on MNIST.



(c) Makantasis et al.'s model [59] trained on the Indian Pines dataset.

Fig. 4: Different vector length/codebook size combinations for additional neural network models. For legend see Figure 3.

the hyper-spectral dataset Indian Pines [66] (Figure 4c). We are dealing with a rather small model, which can be challenging, as seen in the LeNet5 results. Again, our approach significantly outperforms the baseline, and it follows the same trends already discussed for the previous examples.

B. Influence of parameter choices

Next, we evaluate the influence of the vector length and number of centroids on the index structure size. Figure 5 shows the results for VGG-16 based on Equations (7) and (8). For a model size of around 56 MB, the index structure size varies between 0.69 and 4.14 MB, depending on the parameter choice. This corresponds to 1.23–7.39% of the total model size. As the number of clusters determines the number of bits

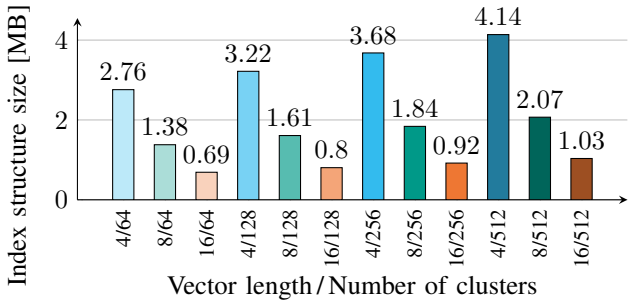


Fig. 5: Comparison of index structure sizes.

TABLE I: Link Budget Parameters

Orbit height	600 km
Carrier frequency	2150 MHz
Receive channel bandwidth	750 kHz
Transmission bitrate	500 kbit/s
Satellite antenna gain	6 dBi
Ground station antenna gain	18.9 dBi
Ground station transmission RF power	50 W

necessary to encode the indices, the size of the index structure increases logarithmically with the number of centroids. The vector length determines the number of indices included into the index structure. The size of the index structure decreases linearly with increasing vector lengths.

The trade-off between index structure size and model performance should be considered for optimal parameter choices. Intuitively, it could be expected that the performance improves with increasing numbers of clusters and decreasing vector length. Our results, however, show that the influence of the parameters depends on a particular model and dataset.

C. Comparison of transmission time

Finally, we evaluate the VGG-16 model's performance using the ESTNeT simulator [28] based on OMNeT++ and the INET libraries [67]. ESTNeT extends OMNeT++ to model characteristics unique to space-terrestrial communication and considers a realistic communication model that takes into account interference based on a signal-to-interference-and-noise (SINR) model. The simulation parameters are summarized in Table I. We consider a single three-unit (3U) CubeSat satellite and a single ground station located in Würzburg, Germany. For transmissions, we use the S-band, which is commonly used for satellite uplink communication [12].

For the underlying communication protocol, we use a lightweight acknowledgment mechanism. The ground station sends updates in packets of 200 bytes each and waits for an acknowledgement from the satellite before sending the next packet. If no acknowledgment is received before the timeout, the last packet is resent to the satellite.

Figure 6 shows the simulation results. The x-axis shows the elapsed transmission time, whereas the y-axis again shows the corresponding Top-1 accuracy. During two periods of more than 10 hours, no communication has been possible. This

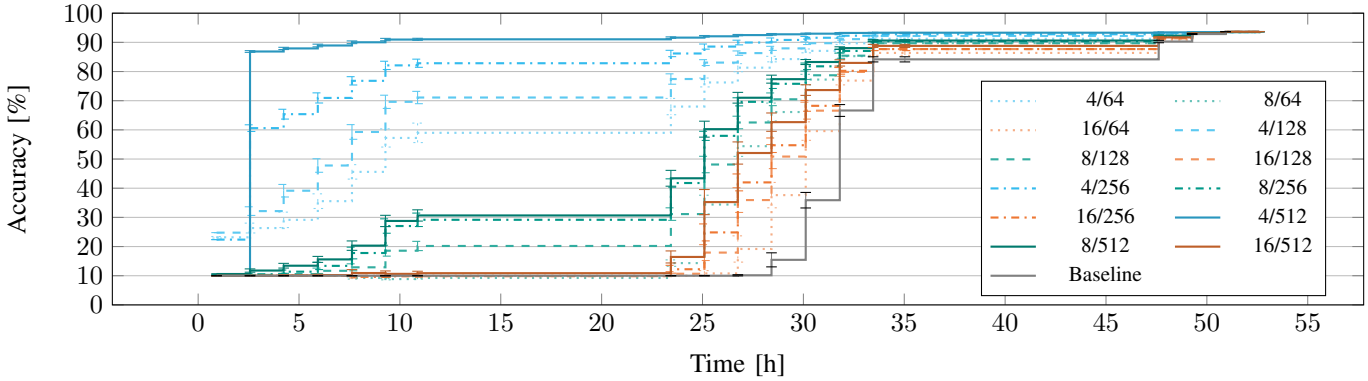


Fig. 6: Transmission times for VGG-16 trained on CIFAR-10 using a simulated satellite-terrestrial communication link.

underlines the importance of incrementally updating the model to achieve a reasonable performance. For the baseline case, more than 45 hours are needed to reach an accuracy within 10% of the full model accuracy. Our approach achieves its best performance for parameter combinations with a vector length of 4. Notably, it is possible to reach accuracies between 58.98% and 91.04% before the first long communication break. At the beginning of the second period without communication, an accuracy within 10% of the full model accuracy can be reached using all parameter combinations. Our results suggest that, for the evaluated scenario, the impact of the index structure size can be neglected.

V. Conclusion

We presented a new transmission mechanism for efficient updates of neural network parameters for satellite applications. Our approach achieves high accuracies even when only a part of the new model's exact weights are transmitted. We leverage vector quantization to compute a lightweight supporting structure, which is transmitted first and serves as basis for incremental transmissions of exact weights. Notably, we can build a vector-quantization-based codebook using the original model, store it on the satellite, and re-use it after model updates.

We evaluated our approach using a wide range of neural network architectures and compared it to a baseline case. The evaluation results show that our approach allows to improve performance up to 87% when only 10% of the original weights are transmitted. In general, increasing the number of centroids and reducing the vector size leads to a better performance. Benefits can vary depending on the particular model architecture. The size of the index structure also depends on the quantization parameters, varying between 1.23% and 7.39% of the total model size, which can be neglected for our setup. Finally, results for a realistic scenario using the ESTNeT simulator show that our mechanism significantly outperforms the baseline.

The presented scheme is not only applicable to Earth-satellite communication, but can be generalized to other communication networks that are characterized by small bandwidth and spotty connections. Examples of such networks are edge-devices such as smart robot platforms or drones and more.

Acknowledgements

This work has been co-funded by the LOEWE initiative (Hesse, Germany) within the emergenCITY center.

References

[1] N. Saeed, A. Elzanaty, H. Almorad, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini, "CubeSat communications: Recent advances and future challenges," *arXiv:1908.09501*, 2019.

[2] P. Barmpoutis, P. Papaioannou, K. Dimitropoulos, and N. Grammalidis, "A review on early forest fire detection systems using optical remote sensing," *Sensors*, vol. 20, no. 22, 2020.

[3] A. Budianu, A. Meijerink, and M. Bentum, "Swarm-to-Earth communication in OLFAR," *Acta Astronautica*, vol. 107, 2015.

[4] A. Poghosyan and A. Golkar, "CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions," *Progress in Aerospace Sciences*, vol. 88, 2017.

[5] P. Singh, P. C. Pandey, G. P. Petropoulos, A. Pavlides, P. K. Srivastava, N. Koutsias, K. A. K. Deng, and Y. Bao, "Hyperspectral remote sensing in precision agriculture: Present status, challenges, and future trends," in *Hyperspectral Remote Sensing*, P. C. Pandey, P. K. Srivastava, H. Balzter, B. Bhattacharya, and G. P. Petropoulos, Eds. Elsevier, 2020.

[6] K. Woellert, P. Ehrenfreund, A. J. Ricco, and H. Hertzfeld, "CubeSats: Cost-effective science and technology platforms for emerging and developing nations," *ASR*, vol. 47, no. 4, 2011.

[7] *CubeSat design specification*, The CubeSat Program, Cal Poly SLO, 2022, rev. 14. [Online]. Available: https://www.cubesat.org/cubesatinfo

[8] A. P. Arechiga, A. J. Michaels, and J. T. Black, "Onboard image processing for small satellites," in *National Aerospace and Electronics Conf. (NAECON)*, 2018.

[9] G. Furano, G. Meoni, A. Dunne, D. Moloney, V. Ferlet-Cavrois, A. Tavoularis, J. Byrne, L. Buckley, M. Psarakis, K.-O. Voss, and L. Fanucci, "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, 2020.

[10] D. Walker, L. Leung, V. Beukelaers, S. Chesi, H. Yoon, and J. Egbert, "ADCS at scale: Calibrating and monitoring the Dove constellation," 2018.

[11] J. Cappaert, "Building, deploying and operating a CubeSat constellation – Exploring the less obvious reasons space is hard," 2018.

[12] D. Vasisht, J. Shenoy, and R. Chandra, "L2D2: Low latency distributed downlink for LEO satellites," in *ACM SIGCOMM Conf.*, 2021.

[13] G. Pandolfi, R. Albi, M. Puglia, Q. Berdal, M. Degroote, M. Messina, M. Ruben, R. Di Battista, M. Emanuelli, M. Daniele, D. Chiuri, C. Thierry, and M. Scaringello, "Solution for a ground station network providing a high bandwidth and high accessibility data link for nano and microsatellites," 09 2016.

[14] M. Harris, "Is Amazon's satellite ground station service ready for primetime?" *IEEE Spectrum*, June 2019.

[15] B. Denby and B. Lucia, "Orbital ddge computing: Nanosatellite constellations as a new class of computer system," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. New York, NY, USA: Association for Computing Machinery, 2020.

[16] K. Devaraj, R. Kingsbury, M. Ligon, J. Breu, V. Vittaldev, B. Klofas, P. Yeon, and K. Colton, "Dove high speed downlink system," in *Small Satellite Conf.*, 2017.

[17] K. Devaraj, M. Ligon, E. Blossom, J. Breu, B. Klofas, K. Colton, and R. Kingsbury, "Planet high speed radio: Crossing Gbps from a 3U CubeSat," in *Small Satellite Conf.*, 2019.

[18] G. Furano, A. Tavoularis, and M. Rovatti, "AI in space: Applications examples and challenges," in *Int. Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2020.

[19] C. Nogales, B. Grim, M. Kamstra, B. Campbell, A. Ewing, R. Hance, J. Griffin, and S. Parke, "MakerSat-0: 3D-printed polymer degradation first data from orbit," 08 2018.

[20] V. Kothari, E. Liberis, and N. D. Lane, "The final frontier: Deep learning in space," *arXiv:2001.10362*, 2020.

[21] G. J.L. and C. C., "On-board collision avoidance applications based on machine learning and analytical methods," in *8th European Conf. on Space Debris, SA/ESOC*, 2021.

[22] V. Leon, G. Lentaris, E. Petrongonas, D. Soudris, G. Furano, A. Tavoularis, and D. Moloney, "Improving performance-power-programmability in space avionics with edge devices: VBN on Myriad2 SoC," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 3, mar 2021.

[23] S. K. Johnson, D. Chelmins, D. Mortensen, M. J. Shalkhauser, and R. Reinhart, *Lessons learned in the first year operating software defined radios in space*.

[24] G. Giuffrida, L. Fanucci, G. Meoni, M. Batič, L. Buckley, A. Dunne, C. van Dijk, M. Esposito, J. Hefele, N. Vercruyssen, G. Furano, M. Pastena, and J. Aschbacher, "The Φ-sat-1 mission: The first on-board deep neural network demonstrator for satellite Earth observation," *IEEE Trans. Geosci. Remote Sens.*, vol. 60, 2022.

[25] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," *NeurIPS*, 1989.

[26] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv:1808.04752*, 2018.

[27] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *arXiv:1412.6115*, 2014.

[28] A. Freimann, M. Dierkes, T. Petermann, C. Liman, F. Kempf, and K. Schilling, "ESTNeT: A discrete event simulator for space-terrestrial networks," *CEAS Space Journal*, vol. 13, 2021.

[29] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," *NeurIPS*, 2014.

[30] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *ICLR*, 2015.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[32] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *NeurIPS*, 2015.

[33] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *ICCV*, 2017.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *arXiv:1510.00149*, 2015.

[35] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *ICML*, 2015.

[36] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinč, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand *et al.*, "DeepCABAC: A universal compression algorithm for deep neural networks," *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 4, 2020.

[37] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NeurIPS Deep Learning and Representation Learning Workshop*, 2015.

[38] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," in *ICLR*, 2015.

[39] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *CVPR*, 2017.

[40] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *ICML*, 2021.

[41] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *CVPR*, 2021.

[42] M. Jankowski, D. Gündüz, and K. Mikolajczyk, "AirNet: Neural network transmission over the air," in *Int. Symposium on Information Theory (ISIT)*, 2022.

[43] T. Fujihashi, T. Koike-Akino, and T. Watanabe, "Federated AirNet: Hybrid digital-analog neural network transmission for federated learning," *arXiv:2201.04557*, 2022.

[44] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *NeurIPS*, 2015.

[45] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *ICML*, 2015.

[46] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *European Conf. on Computer Vision (ECCV)*, 2016.

[47] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv:1606.06160*, 2016.

[48] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," in *ICLR*, 2018.

[49] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *ICLR*, 2018.

[50] D. Soudry, I. Hubara, and R. Meir, "Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights," *NeurIPS*, 2014.

[51] J. Achterhold, J. M. Köhler, A. Schmeink, and T. Genewein, "Variational network quantization," in *ICLR*, 2018.

[52] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *CVPR*, 2016.

[53] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," in *ICLR Workshops*, 2016.

[54] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *ICML*, 2016.

[55] Q. Chen, C. Xin, C. Zou, X. Wang, and B. Wang, "A low bit-width parameter representation method for hardware-oriented convolution neural networks," in *Int. Conf. on ASIC (ASICON)*, 2017.

[56] N. Mitschke, M. Heizmann, K.-H. Noffz, and R. Wittmann, "A fixed-point quantization technique for convolutional neural networks based on weight scaling," in *Int. Conf. on Image Processing (ICIP)*, 2019.

[57] R. M. Gray and D. L. Neuhoff, "Quantization," *Trans. Inf. Theory*, vol. 44, no. 6, 1998.

[58] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, 2010.

[59] K. Makantasis, K. Karantzalos, A. Doulamis, and N. Doulamis, "Deep supervised learning for hyperspectral data classification through convolutional neural networks," in *IEEE Int. Geoscience and Remote Sensing Symposium (IGARSS)*, 2015.

[60] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *CVPR*, 2009.

[61] S. Zagoruyko, "92.45% on CIFAR-10 in torch," http://torch.ch/blog/2015/07/30/cifar.html, 2015.

[62] A. Krizhevsky, V. Nair, and G. Hinton, "CIFAR-10 dataset," https://www.cs.toronto.edu/~kriz/cifar.html, 2009.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *1512.03385*, 2015.

[64] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, 1998.

[65] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, 2012.

[66] M. F. Baumgardner, L. L. Biehl, and D. A. Landgrebe, "220 band AVIRIS hyperspectral image data set: June 12, 1992 Indian Pine test site 3," Sep 2015. [Online]. Available: https://purr.purdue.edu/publications/1947/1

[67] A. Virdis and M. Kirsche, *Recent Advances in Network Simulation*. Springer, 2019.