Overview of the Neural Network Compression and Representation (NNR) Standard

Heiner Kirchhoffer, Paul Haase, Wojciech Samek, *Member, IEEE*, Karsten Müller, *Senior Member, IEEE*, Hamed Rezazadegan-Tavakoli, *Member, IEEE*, Francesco Cricri, Emre Aksu, Miska M. Hannuksela, *Member, IEEE*, Wei Jiang, *Member, IEEE*, Wei Wang, *Member, IEEE*, Shan Liu, *Senior Member, IEEE*, Swayambhoo Jain, *Member, IEEE*, Shahab Hamidi-Rad, Fabien Racapé, and Werner Bailer, *Member, IEEE*

***Abstract*—Neural Network Coding and Representation (NNR) is the first international standard for efficient compression of neural networks (NNs). The standard is designed as a toolbox of compression methods, which can be used to create coding pipelines. It can be either used as an independent coding framework (with its own bitstream format) or together with external neural network formats and frameworks. For providing the highest degree of flexibility, the network compression methods operate per parameter tensor in order to always ensure proper decoding, even if no structure information is provided. The NNR standard contains compression-efficient quantization and an arithmetic coding scheme (DeepCABAC) as core encoding and decoding technologies, as well as neural network parameter pre-processing methods like sparsification, pruning, low-rank decomposition, unification, local scaling and batch norm folding. NNR achieves a compression efficiency of more than 97% for transparent coding cases, i.e. without degrading classification quality, such as top-1 or top-5 accuracies. This paper provides an overview of the technical features and characteristics of NNR.**

***Index Terms*—Neural Network Compression, Neural Network Representation, MPEG Standards, Machine Learning**

I. Introduction

THE Neural Network Compression and Representation standard (NNR) is the first standard by the ISO/IEC Moving Picture Experts Group (MPEG) standardization working group that targets the efficient compression and transmission of neural networks. The NNR standard provides a compression efficiency of up to 97% for transparent coding use cases, i.e. without degrading the classification and inference capability of the respective neural network. This is reflected by the obtained evaluation results, where compression efficiency in terms of compressed bitrate vs. original neural network bitrate is analyzed. Here, performance metrics for relevant use cases in multimedia for the original as well as decoded and reconstructed network are used, such as constant top-1 and top-5 classification accuracy for image classification. In addition, much higher coding gains can be obtained if the classification accuracy is allowed to drop, as reflected by the rate-classification curves.

The demand for efficient compression of neural networks has grown exponentially in recent years, as machine learning and artificial intelligence have evolved and these methods have been incorporated into almost every technical field, such as medical applications, transportation, network optimization, big data analysis, surveillance, speech, audio, image and video classification, and many more [1]. Furthermore, the neural network architectures have developed towards much more complex structures with increasing number of layers and neurons per layer, such that current architectures already contain several hundreds of millions of weight parameters. An additional factor for the exponential growth is the development of use cases itself. While in simple scenarios, a neural network is trained, transmitted once to an application device and used there for inference, new scenarios of federated learning demand for continuous communication between many devices [2], [3]. Accordingly, such use cases require the best compression technology with highest coding gain in order to minimize the overall communication traffic.

Therefore, the NNR standard is designed to provide the highest compression efficiency for deep neural network by combining preprocessing methods for data reduction, quantization and context-adaptive arithmetic binary coding (DeepCABAC). The standard supports the most common neural network formats, such as PyTorch[1], TensorFlow[2], ONNX [4] or NNEF [5] in two different ways: Either, NNR is used independently by compressing all parameter tensors of a neural network and including the respective network structure or connection graph into the NNR bitstream, or NNR is used within an external framework by also coding neural network parameters tensor-wise, while all structure data is handled by the framework.

The paper is organized as follows: Section II presents related

Manuscript received <Month> xx, 202x; etc.

H. Kirchhoffer, P. Haase, W. Samek and K. Müller are with Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, 10587 Berlin, Germany (e-mail: {karsten.mueller, heiner.kirchhoffer}@hhi.fraunhofer.de)

H. Rezazadegan-Tavakoli, F. Cricri, E. Aksu and M. M. Hannuksela are with NOKIA Technologies, Finland.

W. Jiang, W. Wang and S. Liu are with Tencent, Palo Alto, CA, USA

F. Racapé, S. Jain, S. Hamidi-Rad are with InterDigital AI Lab, Los Altos, California 94022, USA.

W. Bailer is with JOANNEUM RESEARCH, 8010 Graz, Austria.

[1] https://pytorch.org/

[2] https://www.tensorflow.org/

technology from the field of neural network compression. Section III describes coding tools and features. The high-level syntax of the NNR standard is described in Section IV, while Section V introduces preprocessing and parameter reduction methods. The NNR core compression methods are described in Section VI with quantization methods, followed by Section VII with the arithmetic coding engine. In Section VIII, the compression efficiency is demonstrated with respective NNR coding results and finally, the paper is concluded in Section IX.

II. Related Work

Starting with the development of deep neural networks, early works already showed the redundancy of the parameters in these models, and that the full set of parameters can be predicted from a fraction of them [6]. NNs are often not trained from scratch, and it has been shown that models obtained from retraining a model or training multiple specializations using transfer learning have similar parameter statistic as the base model [7]. Reducing the redundancy of the model parameters involves typically three steps: (i) reduction of parameters, for example by eliminating neurons (pruning), reducing the entropy of a tensor (sparsification) or decomposing/ transforming a tensor, (ii) reducing the precision of parameters (i.e. quantization), and (iii) performing entropy coding. Han et al. were one of the first to describe this complete compression pipeline [8].

A large number of variants of algorithms for obtaining NNs with a reduced number of parameters have been proposed, which is for example evident from a recent survey that analyses and compares 81 NN pruning papers [9]. The *Lottery Ticket Hypothesis* [10] postulates that for a dense randomly initialized feed forward NN there are subnetworks that achieve the same performance at similar training cost. Several recent works address this hypothesis, showing that such an optimal subnetwork can be obtained by pruning without further training [11] or finding subnetworks that can be re-trained from an early iterate [12]. While removing entire neurons or channels will directly impact inference complexity, parameter reduction resulting in sparser tensors requires specific support on the target platform. Recently, support for efficient operations on sparse tensors is increasing, e.g. for GPUs [13].

Quantization can be applied in a straight forward way to the parameters of an NN. For many commonly used networks using 8 bits or even less results in no or only small performance degradations, in particular, if the quantized network is then fine-tuned. Several works show that performance loss can be avoided when quantization is already considered in the training process (requiring a differentiable quantization function) [14] or when learning the quantization function with the network [15]. Recent work proposes an asymptotic-quantized estimator (AQE) to estimate the gradients in networks with highly quantized weights (e.g. 1 bit), ensuring smoothness and differentiability during back-propagation [16]. In order to gain not just model size but also inference efficiency from quantization, support on the target architecture is needed. Neural network inference using 8, 4 or even 1 bit fixed point representations has been an active research topic in recent years, supported first on FPGAs [17] and meanwhile also on general purpose hardware such as GPUs (e.g., NVIDIA TensorRT [18]). Deep learning frameworks have added support for fixed point inference, as well as varying the precision for different parts of the NN (mixed precision).

The borders between compressing a trained NN, possibly with fine-tuning, and training a smaller network for the same task are not sharp. The approach of training a more compact network on the outputs of a larger trained network is known as knowledge distillation or teacher-student learning and predates deep learning [19]. Taking this further to training a set of models and selecting the best one results in performing network architecture search (NAS). Although less computationally expensive NAS methods have been proposed recently [20], NN compression approaches require a more efficient approach resulting in a single model. Another issue for retraining is the requirement to access (the original) training data, which may not be feasible in all use cases. An interesting approach in this direction is performing compression with a synthetic dataset [21].

One issue that makes creating compact NNs for efficient inference challenging is the dependency on the characteristics of the target hardware. A framework for dynamic inference on resource constrained hardware, including input- and resource dependent dynamic inference mechanisms, allowing to meet specific resource constraints, has been proposed recently [22]. First steps are being made towards network compression methods that output representations prepared for later specialization to the target platform [23]. However, one challenge is still the complexity of measuring the impact of certain modifications of the network in terms of speed and energy consumption on the target platform. First frameworks such as Deep500 [24] have been proposed, but are still in an early stage.

While NN compression is still a very active research area, building blocks that are common to most approaches have emerged. For each of them, established methods and best practices can be found in literature. In order to facilitate adoption of these tools beyond the research community, standardizing interfaces and providing reference implementations are required.

III. NNR Coding Tools and Features

Achieving compact representations of trained neural networks addresses two main goals: (i) providing efficiency when the NN is stored or transmitted and (ii) allowing for resource-efficient inference. The importance of these goals depends on the specific use case. For example, for frequent updates in a federated training scenario involving nodes in a cloud infrastructure the efficient transmission is most important, while for infrequent deployments of a trained NN to an embedded device supporting efficient inference is crucial. Methods addressing the second goal also support the first one, and need to output a representation that can be used directly for inference (at least on specific target platform), while methods addressing the first one will have additional coding steps requiring decoding at the receiving end.

The NNR coding design thus addresses these two core goals, i.e. coding efficiency of any neural network, ease of transport and good implementability due to parallelization as well as interoperability with common NN frameworks. It allows efficient compression of original NNs, as well as certain preprocessing steps for NN parameter reduction.

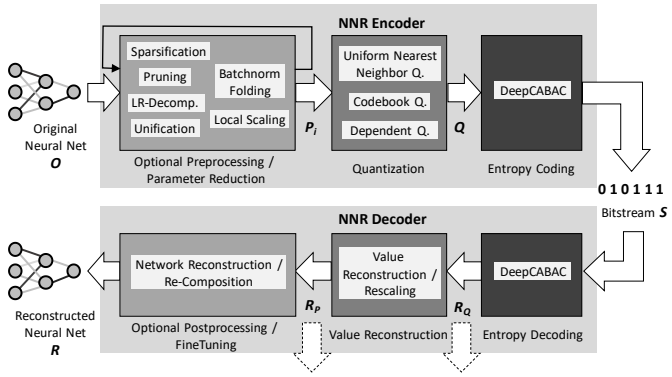


Fig. 1 NNR Overview.

Fig. 1 provides an overview of the NNR coding and decoding process. Starting from an original network O, which can have any NN architecture, parameter reduction may be applied to the NN. Multiple such parameter reduction methods (e.g., structure pruning and tensor sparsification) may be applied subsequently, resulting in a preprocessed network P_i (after i such operations). Some or all of the remaining parameters in P_i are then quantized, resulting in the network Q. In order to obtain a bitstream S for storage or transmission of the model, entropy coding is applied to the quantized parameters. In the decoding process, entropy decoding provides a reconstructed model R_Q. As entropy coding is a lossless operation, R_Q is equivalent to Q. If the target platform supports inference with the quantized representation(s) used in R_Q, this NN can be used for inference. Otherwise, the parameter values need to be reconstructed to their original representation, resulting in R_P (note that even if the precision of values is the same as in P_i, the models will only be equivalent but not identical due the information loss during quantization). R_P can be directly used for inference, unless any of the sparse tensor representation used cannot be processed on the target platform. In this case fully reconstructing the network R is required, which does not contain any tensor or structure representations different to O. The coding gain is evaluated as compression ratio $c_r = S/O$ of bitstream size over original network size, or alternatively as compression efficiency $c_e = 1 - c_r$.

A. Coding Pipelines

As different use cases focus on different competing requirements (e.g., coding efficiency vs. inference complexity), and the usefulness of certain encoding tools may depend on the intended target platform, there is no single optimal coding pipeline that optimally serves all intended use cases. NNR is thus designed as a toolbox of coding tools, from which appropriate coding pipelines can be assembled by selecting tools for each of the three stages in the process shown in Fig. 1. Some of the tools are alternatives for addressing neural network models with different types of characteristics, while other tools are designed to work in sequence.

Parameter reduction methods process a model to obtain a compact representation, for which the NNR standard specifies the most widely used: *Sparsification* produces a sparse representation of the model, e.g., by replacing some weight values with zeros. *Unification* produces groups of similar parameters in order to lower the entropy of model parameters by making them similar to each other. *Pruning* reduces the number of parameters by eliminating parameters or group of parameters. *Decomposition* changes the structure of the weight tensors to obtain a more compact representation. The parameter reduction methods can be combined or applied in sequence, e.g., performing pruning and sparsification.

Parameter quantization methods reduce the precision of the representation of parameters. The methods include *uniform quantization*, *codebook-based quantization* and *dependent scalar quantization*. If supported by the inference engine, the quantized representation can directly be used for more efficient inference. For storage and transmission, it also prepares the data for entropy coding. Parameter quantization can be applied to the outputs of parameter reduction methods as well as to source models. Entropy coding methods encode the results of parameter quantization methods. These coding tools are presented in detail in Sections V-VII.

B. Interoperability with Exchange Formats

The representation of a trained NN consists of two main components: (i) the description of the topology of the NN, i.e. the definition of the layers, their types, sizes and the connections between them, and (ii) the parameter values such as weights and biases, grouped into tensors. NNR focuses on the second component, aiming to replace raw parameter tensors with more efficient representations. The first component is well covered by the native formants of common deep learning frameworks (most notably, TensorFlow[3] and PyTorch[4]). In order to improve interoperability, two exchange formats have been proposed: (i) Open Neural Network Exchange Format ONNX [4], with a serialized format is based on protobuf[5], with strings identifying types of elements in the graph, and widely supported as import/export format by different frameworks. (ii) Neural Network Exchange Format (NNEF) [5], which is an effort by the Khronos group to define an exchange format to use networks trained with different frameworks for inference on different platforms. However, apart from basic support for quantization, these formats currently do not support compressed model representations.

Here, the NNR standard is complementary to these efforts and achieves interoperability with topology information represented in native formats as well as with exchange formats. The standard thus allows carrying topology information defined in any of these formats as part of an NNR bitstream containing

[3] https://www.tensorflow.org/
[4] https://pytorch.org/
[5] https://developers.google.com/protocol-buffers

compressed parameter tensors (see Section IV for details). For the exchange formats, NNR also proposes a way to carry compressed parameter tensors in these formats, replacing the uncompressed representation for some or all tensors of the NN.

C. Decoding Methods

The NNR standard provides decoding methods for specific network tensor types, such as integer, scaled integer, or floating point parameter representations. A decoding method denoted NNR_PT_INT provides decoding functionality for parameters that are arrays or tensors of integer values. A decoding method NNR_PT_FLOAT extends NNR_PT_INT by adding quantization step size Δ that is multiplied with each decoded integer value yielding scaled integers (which are usually represented as float values). This quantization step size is derived from an integer quantization parameter qp and an integer parameter qp_density as follows:

$$mul = 2^{qp\_density} + (qp \,\&\, (2^{qp_{density}} - 1), \qquad (1)$$

$$\Delta = mul \cdot 2^{(qp \gg qp_{density}) - qp_{density}}. \qquad (2)$$

Here, ">>" represents the bitwise right-shift operator. Both values qp and qp_density are signaled in the bitstream. As can be seen from equations (1) and (2), a qp of 0 corresponds to Δ = 1, negative qp values to Δ < 1, and positive qp values to Δ > 1. The qp_density controls the granularity of the step size. More precisely, increasing the qp by $2^{qp\_density}$ corresponds to doubling Δ. For example, a typical value for qp_density would be 3 allowing 7 intermediate step size values between Δ and 2Δ.

A further decoding method NNR_PT_BLOCK jointly decodes several related parameter arrays or tensors including local scaling parameters, biases, batch norm parameters, and weights. This is the basis for enabling techniques like batch norm folding or local scaling.

The decoding methods NNR_PT_FLOAT and NNR_PT_BLOCK can furthermore be combined with an integer codebook. I.e., the decoded values are indexes to values of a codebook that contains integer values which are finally multiplied by step size Δ.

D. Parallel Decoding

In order to provide a decoder with the ability of parallel decoding of large tensors, a block scanning and entry point concept is included in the NNR standard. A tensor is first reshaped into 2D and then subdivided into blocks of size NxN where N can be 8, 16, 32, or 64. Such blocks may directly be fed into optimized inference engines that operate on an NxN block size. Decoding of blocks is carried out in row-major order. For each row of blocks (except the uppermost row), entry point information is provided in the bitstream so that a decoder can choose to start the decoding at a particular row or to decode several rows in parallel. This refers in particular to dependent quantization and arithmetic coding methods, where respective absolute values of state variables are signaled in the entry point information to enable parallel block row or sub-tensor decoding independent of previously coded information.

IV. High-Level Syntax

Storage, carriage and distribution of compressed neural networks are important *systems* aspects of NNR. The NNR standard not only defines compression tools for neural networks but also *high-level syntax* for efficient carriage of such compressed data.

Compressed neural network data and related metadata are stored and carried in the form of structured syntax elements which are called *NNR units*. When multiple NNR units are concatenated, they form an *NNR bitstream* as illustrated in Fig. 2 *top*. The NNR bitstream format provides an efficient and well-structured mechanism to carry, signal and exchange compressed neural network representations, either fully or partially (e.g. compressed data of one layer only). With the defined NNR high-level syntax, it is possible to carry compressed neural network information at any granularity as long as such information is uniquely referenceable to the neural network topology (e.g. a tensor, filter, layer, or bias).

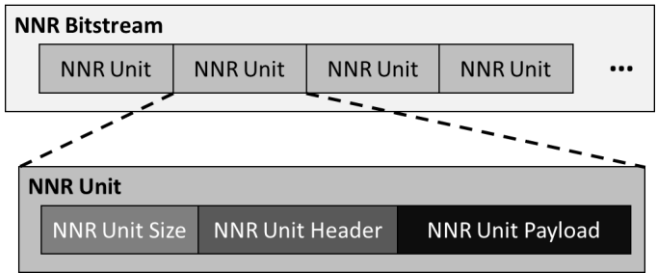


Fig. 2 *Top*: NNR bitstream with NNR units, *Bottom*: NNR Unit Structure.

As presented in Fig. 2 *bottom*, an NNR unit consists of the following syntax elements in the given order:

- NNR unit size, which signals the total byte size of the NNR unit, including the NNR unit size itself.
- NNR unit header, which contains information about the NNR unit type and related metadata.
- NNR unit payload, which contains compressed or uncompressed data related to the neural network.

TABLE I lists different types of NNR units and their descriptions. An NNR bitstream always starts with an *NNR Start Unit*, followed by an *NNR Model Parameter Set Data Unit* and several *NNR Compressed Data Units*. If neural network topology information is signaled to be carried inside an NNR bitstream, then *NNR Topology Data Units* (and optionally *NNR Quantization Data Units*) are also present before any *NNR Compressed Data Unit* that reference them.

Multiple NNR units which are related to each other can be grouped together and carried as a single NNR Unit. Such NNR Units are called *NNR Aggregate Units*. For example, the data that refers to a neural network layer can be aggregated into such a unit. NNR Aggregate Units may have their own parameter sets. If multiple NNR units are aggregated into an NNR Aggregate Unit, then an *NNR Layer Parameter Set Data Unit* is present as the first NNR Unit inside the NNR Aggregate Unit which further provides information about the NNR Aggregate Unit contents. NNR Layer Parameter Set Data Units are active until another one is signaled or until the data boundary of the

containing NNR Aggregate Unit is reached.

TABLE I
NNR UNIT TYPES

Identifier	NNR Unit Type	Description
NNR_STR	NNR start unit	Compressed neural network bitstream start indicator
NNR_MPS	NNR model parameter set data unit	Neural network global metadata and information
NNR_LPS	NNR layer parameter set data unit	Metadata related to a partial representation of neural network
NNR_TPL	NNR topology data unit	Neural network topology information
NNR_QNT	NNR quantization data unit	Neural network quantization information
NNR_NDU	NNR compressed data unit	Compressed neural network data
NNR_AGG	NNR aggregate unit	NNR unit with payload containing multiple NNR units

NNR high-level syntax also enables efficient carriage of compressed neural network data by providing mechanisms for partitioning compressed tensor data into multiple NNR units, as well as indicating whether such NNR units are independently decodable.

NNR Model Parameter Set Data Units and NNR Layer Parameter Set Data Units can also carry additional metadata related to the compressed neural network such as inference performance at different sparsification, pruning, unification and decomposition levels. Moreover, pruning information related to a neural network topology can be signaled inside an NNR Topology Data Unit.

NNR utilizes industry-defined and existing topology representations and enables the carriage of such externally defined information as part of the NNR bitstream. Such data is carried inside the NNR Topology Data Units and NNR Quantization Data Units. By utilizing a reference signaling mechanism, different elements and components of a neural network can be compressed, carried in the NNR bitstream and then linked to the neural network topology. In addition to this feature, carriage of NNR bitstream inside different neural network exchange formats, as given in Section III.B, is also defined by the specification.

V. PRE-PROCESSING AND PARAMETER REDUCTION

Instead of coding and compressing a neural network in its original form, one of the preprocessing and parameter reduction methods can be applied. The methods defined in the NNR standard are described in the following subsections.

A. Sparsification

Sparsification refers to a group of technologies that process the parameters or group of parameters to produce a sparse representation of the model by replacing some weight values with zeros. Obtaining weight matrices/tensors that are sparse is one method for achieving parameter reduction.

To sparsify the neural network, some of the convolutional kernels' values, or some of the fully-connected layers' weights are set to zero. A highly-sparse neural network is likely to have low entropy and thus be more compressible by entropy-based encoders such as arithmetic codecs. In addition, inference-time speed-ups can be achieved when sparse matrix multiplications are used [25]. An important step of the sparsification process is to determine which parameters are less important than others, so that they can be set to zero while minimizing the performance loss of the neural network (e.g., the classification accuracy in the case of a classifier neural network).

One common approach is to sparsify the parameters with low absolute values. However, neural networks are usually not trained specifically for the purpose of being sparsified, therefore sparsifying such models can be considered to be sub-optimal. A better approach, adopted into the NNR standard, is to optimize the neural network for the sparsification process by fine-tuning a pre-trained neural network with a custom-designed loss function. A sparsity loss was designed based on the sparsity metric introduced in [26], and is defined as follows:

$$L_{sparsity}(w) = \frac{|w|_1}{|w|_2} + \gamma \frac{|w|_2^2}{|w|_1}, \quad (3)$$

where $|x|_1$ and $|x|_2$ are the l_1 and l_2 norms of x, respectively. During the fine-tuning process, γ is chosen such that $\frac{|w|_2^2}{|w|_1} = \frac{1}{3}\frac{|w|_1}{|w|_2}$. More details on the sparsity loss can be found in [27]. The sparsity loss in eq. (3) is combined with the task loss, i.e., the loss used for pre-training the neural network, such as the cross-entropy loss for a classifier, thus obtaining the total loss used for fine-tuning:

$$L_{total}(w) = L_{task}(w) + \lambda\, L_{sparsity}(w), \quad (4)$$

where λ is set so that $L_{sparsity}(w) = m\, L_{task}(w)$ on a validation dataset, and m is a hyper-parameter which can be tuned to achieve different rate-distortion points – higher m values lead to better robustness to high sparsity ratios and therefore can be used to achieve higher compression. After this fine-tuning process has completed, the actual sparsification is performed by setting the parameter values to zero that are lower than a predefined threshold. This threshold is another hyper-parameter that can be used to achieve different rate-distortion points – higher thresholds lead to higher sparsity and therefore higher compression.

B. Pruning

In modern literature, pruning and sparsification are often used interchangeably. Nonetheless, given context and how removal of weights are done, the two terminologies may refer to different technologies. In the NNR standard, pruning is defined as an operation that reduces the number of parameters by eliminating parameters or groups of parameters. This procedure results in a dense representation which has less parameters in comparison to the original model, e.g., by removing redundant convolution filters from the layers, as described in subsection V.B.1). Second, micro-structured pruning removes weight coefficients to accelerate GEMM computation and is described in subsection V.B.2).

1) General Neural Network Pruning

The NNR standard contains a pruning technology that is combined with the sparsification methods. Algorithm 1 summarizes the three steps.

ALGORITHM 1

Inputs:	Pre-trained network, pruning ratio x, sparsification ratio y
1)	Analyze the neural network weight to determine which weights to be removed from the network by estimating weight importance using eq. (5).
2)	Remove the least important neurons with respect to the pruning ratio x.
3)	Apply data dependent sparsification with regard to sparsification ratio y
4)	Repeat steps 1)-3) when required.

Whenever pruning ratio y is satisfied, step 3 can be reduced to employing only task loss to improve the neural network performance. Steps in Algorithm 1 could be used in a progressive fashion or at once.

The neural network weights are estimated based on a diffusion process over the layers. An example of pruning of convolution filters is provided below, and a similar formulation applies to other type of layers and to group of layers.

Each convolution layer consists of a weight tensor, or filter, denoted $\mathcal{F} \in R^{C_o \times K \times K \times C_i}$ where C_o is the number of output channels, K is the dimension of the convolution kernel, and C_i is the number of input channels.

Under constant input the redundancy in a layer output is modelled by the internal redundant information inside the filter. Thus, by considering an ergodic Markov process between the output channels, graph diffusion is employed to find the redundancy. To this end, given a convolution filter $\mathcal{F}$, a feature matrix $\mathbf{M} \in R^{C_o \times m}$ is obtained where $m = K \times K \times C_i$, via tensor reshape.

Following the ergodic Markov chain with each output channel as one state, the probability of reaching a particular state at the equilibrium is $\pi^T = \pi^T \mathbf{P}$ where $\mathbf{P}$ is the stochastic transition matrix and π is the equilibrium probability of $\mathbf{P}$, corresponding to the left eigenvector $\lambda = 1$. Under equilibrium, the importance could be defined as

$$S = exp(-\frac{1}{\sigma\pi}), \tag{5}$$

where σ is a smoothing factor, that could be equal to the number of output channels. The transition matrix P is determined as

$$\mathrm{p_{ij}} = \frac{\mathrm{e}^{-\mathrm{D}(\mathrm{m_i},\mathrm{m_j})}}{\sum_{z=1}^{c_o} \mathrm{e}^{-\mathrm{D}(\mathrm{m_i},\mathrm{m_z})}}, \tag{6}$$

where m_i is the i-th row of the $\boldsymbol{M}$ and $D(\cdot,\cdot)$ is any distance function of preference. A higher value of S will indicate more dissimilarity, importance and salience for output channel in comparison to the other output channels. To prune the filters, after computing the S, less salient channels are removed.

In step (3) any of the sparsification methods could apply, as shown in section V.A.

2) Micro-structured Pruning

The convolutional computation in DNN is commonly implemented as GEneral Matrix Multiplication (GEMM). For this, the NNR standard applies micro-structured weight pruning, which removes weight coefficients in the micro-structured level to accelerate GEMM computation.

Let W^k be the weight tensor of the k-th layer. W^k is a general 5-D tensor of size $c_1^k \times c_2^k \times n_1^k \times n_2^k \times n_3^k$, where $c_1^k(c_2^k)$ is the number of input (output) channel and n_1^k, n_2^k, and n_3^k give the kernel size. When any of c_1^k, c_2^k, n_1^k, n_2^k or n_3^k equals 1, tensor W^k is reduced to a lower dimension. Micro-structure pruning first reshapes W^k into a 3D tensor of size $c_1^{k\prime} \times c_2^{k\prime} \times n^k$ (*e.g.*, $c_1^{k\prime}=c_1^k$, $c_2^{k\prime} = c_2^k$, $n^k = n_1^k \times n_2^k \times n_3^k$), and then partitions the resized weight tensor into micro-structured blocks (denoted by B_j^k as the j-th block of W^k) of size $b_1^k \times b_2^k \times b_3^k$. Weights within the selected micro-structured blocks are set to 0. A pruning loss $L(B_j^k)$ can be computed for B_j^k as the L_N norm of the absolute of weights in B_j^k (*e.g.*, L_1 as MAE or L_2 as MSE).

The micro-structured blocks can be 3-D, 2-D or 1-D blocks, resulting in different model compression and acceleration effects. When c_1^k cannot be fully divided by b_1^k, c_2^k cannot be fully divided by b_2^k, or n^k can not be fully divided by b_3^k, micro-structured blocks along the boundary of these corresponding dimension will be smaller. That is, $b_1^k \times b_2^k \times b_3^k$ is the maximum size of the micro-structured blocks.

A pruning mask M^k is maintained in the training process with the same shape as W^k, which records whether the corresponding weight coefficients are pruned or not. Given the original target loss L_{train} of the task (categorical cross-entropy for image classification, MSE for image compression, *etc*.), the training process iteratively takes the following two steps:

Step 1: The micro-structured blocks are ranked based on their pruning loss in ascending order. Given a pruning ratio p as a hyperparameter, the top p super-blocks with smallest pruning loss are selected to be pruned.

Step 2: Weight coefficients that are marked by M^k as being pruned are fixed, and the remaining unfixed weight coefficients in W^k are updated through a neural network training process by optimizing the target loss L_{train}.

The micro-structured weight pruning will output an updated model with the same model structure as the input model, where part of the weight coefficients have been structurally removed (pruned). The output model can be directly used in the same way as the input model.

C. Low-Rank Decomposition

In NNR, a network can also be preprocessed by low-rank decomposition of layer weight parameters. For a dense layer with weight-matrix $W \in \mathbb{R}^{m\,x\,n}$ rank $r \leq min\,(m,n)$ approximation is obtained by solving (7).

$$U_W, V_W = \underset{U \in \mathbb{R}^{m\,x\,r}, V \in \mathbb{R}^{r\,x\,n}}{\arg\min} ||W - UV||_F^2 \tag{7}$$

The above problem is efficiently solved by singular value decomposition (SVD) of weight-matrix W. Using the low-rank factors U_W, V_W the weight-matrix W is approximated as $W \approx U_W V_W$ and therefore the total number of parameters are reduced from mn to $r(m+n)$. With rank r approximation the dense layer is effectively converted to a two dense layer of sizes $m \times r$ and $r \times n$ with no bias and non-linearity in between them, and original bias in the second layer. The final rank for low-rank approximation is chosen using a tolerance ε by finding the

minimum rank that leads to $||W - U_W V_W||_F^2 \leq \varepsilon$. In some cases, such tolerance may not be achievable with reduction in the number of parameters. Therefore, if the total number of parameters after low-rank approximation for a given tolerance ε is more than the parameters in the original weight-matrix, skip the low-rank approximation is skipped and the original matrix retained. This happens when $r \geq \frac{mn}{m+n}$.

While the SVD is computationally more expensive than pruning, its computational complexity can be improved by leveraging orthogonality of the singular-vectors. Rank $r+1$ SVD shares first r singular-vectors from rank r SVD and therefore only one pair of singular-vectors is needed for calculating rank $r+1$ low rank approximation.

Consider a convolutional layer for one input channel, such that the general 5-D convolution tensor reduces to a 4-D weight tensor with size $f_h \times f_w \times n_i \times n_o$, where f_h, f_w are the height and width of the convolutional filter, n_i is the number of input channels and n_o is the number of output channel. For low-rank decomposition the 4-D weight tensor of convolutional layer first needs to be reshaped into a 2-D matrix in the following way: First the 3-D filters for each output channel is vectorized into a vector of size $f_h f_w n_i$, followed by stacking the n_o vectors into a matrix of size $f_h f_w n_i \times n_o$. A rank r approximation of this $f_h f_w n_i \times n_o$ matrix reduces the number of parameters from $f_h f_w n_i n_o$ to $r(f_h f_w n_i + n_o)$. Similar to the dense layers, these low-rank factors can be reshaped back into two back-to-back convolutional layers of size $f_h \times f_w \times n_i \times r$ and $1 \times 1 \times r \times n_o$ with no-bias and non-linearity in between them, and original bias in the second convolutional layer.

Since the low-rank approximation effectively reduces a layer into two more computationally efficient layers it leads to equivalent gains in inference complexity proportional to the reduction in number of parameters with a typical off-the-shelf GPU. Another advantage is that it allows for efficient fine-tuning for cases when the performance of DNN drops beyond acceptable limits after low-rank approximation.

D. Unification

Unification in the NNR standard is a generalization of weight pruning. Here, the weight representation is reduced in a structured way that benefits storage and GEMM computation. This approach unifies weights within a selected micro-structured weight block by assigning them a shared absolute value. When this value is zero, the method reduces to the micro-structured weight pruning approach described in Section V.B.2) The micro-structured weight unification method keeps the neuron connections instead of removing them (setting them to zero as in weight pruning), which better preserves the original network structure to provide a balanced model for both compression and task performance. In contrast, the micro-structured weight pruning pursues more aggressive compression effects, *i.e.*, the pruned micro-structured blocks can be completely removed from storage and computation. Both micro-structured weight unification and micro-structured weight pruning are hardware friendly for both model storage and inference computation, in terms of accommodating flexible micro-structured block shapes that are compatible with the underlying inference engine.

Specifically, the weight tensor W^k of the *k*-th layer of the network is reshaped into a 3D tensor of size $c_1^{k\prime} \times c_2^{k\prime} \times n^k$. Here the reshaped weight tensor is further partitioned into super-blocks (denoted by S_i^k as the *i*-th super-block of W^k) of size $s_1^k \times s_2^k \times n^k$. For example, super-blocks of size $64 \times 64 \times n^k$ are selected to be consistent with 3-dimensional Coding Tree Units (CTU3Ds). Each super-block is further partitioned into micro-structured blocks (denoted by $B_{i,j}^k$ as the *j*-th block of S_i^k) of size $b_1^k \times b_2^k \times b_3^k$. Weight unification happens within the selected micro-structured blocks, where its weight coefficients are set to have the same absolute value while maintaining their original signs:

$$v_{i,j,l}^k = \begin{cases} q_{i,j}^k, & \text{if } w_{i,j,l}^k \geq 0, \\ -q_{i,j}^k, & \text{otherwise,} \end{cases} \tag{8}$$

where $v_{i,j,l}^k$ is the newly assigned value for coefficient $w_{i,j,l}^k$ in block $B_{i,j}^k$. Here $q_{i,j}^k$ is computed as the mean of the absolute of weights in $B_{i,j}$: $q_{i,j}^k = avg_{w_{i,j,l}^k \in B_{i,j}^k} |w_{i,j,l}^k|$, and the loss introduced by this unification operation can be measured by a unification loss $L(B_{i,j}^k)$, which can be computed as the L_N norm of the absolute of weights in $B_{i,j}^k$ (*e.g.*, L_2 as MSE). Then the unification loss $L(S_i^k)$ of the super-block S_i^k is computed by averaging $L(B_{i,j}^k)$ across micro-structured blocks in S_i^k.

The micro-structured blocks can be 3D, 2D or 1D, resulting in different model compression and acceleration effects. Also, when c_1^k cannot be fully divided by s_1^k, or c_2^k cannot be fully divided by s_2^k, the super-blocks along the boundary of the corresponding dimension will be smaller. When n^k can not be fully divided by b_3^k, s_1^k cannot be fully divided by b_1^k, or s_2^k cannot be fully divided by b_2^k, the micro-structured blocks along the boundary of the corresponding dimension will be smaller. That is, $b_1^k \times b_2^k \times b_3^k$ is the maximum size of the micro-structured blocks, and $s_1^k \times s_2^k \times n^k$ is the maximum size of the super-blocks.

A unification mask M^k is maintained in the training process, which takes the same processing steps as the pruning mask (see section V.B.2) for details).

The micro-structured weight unification will output an updated model with the same model structure as the input model, where part of the weight coefficients have been structurally changed (unified). The output model can be directly used in the same way as the input model.

E. Batch Norm Folding

Batch norm folding is a technique for reducing redundancy of particular parameters for which interdependencies are known by encoder and decoder. It assumes that the combination of a convolutional or fully-connected layer with a batch norm layer of the following form can be expressed as

$$BN(X) = \frac{W * X + b - \mu}{\sqrt{\sigma^2 + \epsilon}} \circ \gamma + \beta \tag{9}$$

where X is the input, $BN(X)$ is the output, W is the weight

tensor (represented as 2D matrix so that each row corresponds to a neuron), b is a bias parameter, and the remaining parameters are batch-normalization parameters. Note that b, μ, σ^2, γ, and β have the same shape as X and that X is shaped as a transposed vector. Parameter ϵ is a scalar close to zero. Operator $\circ$ denotes element-wise scaling of each row vector of a matrix (or each element of a transposed vector) with the corresponding element of a transposed vector.

Instead of encoding each of the parameters individually, the following transformation can be done before encoding:

$$BN(X) = \alpha \circ W * X + \delta \tag{10}$$

where $\alpha = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$ and where $\delta = \frac{(b-\mu)\circ\gamma}{\sqrt{\sigma^2+\epsilon}} + \beta$. Instead of the original parameters, it is now sufficient to encode W, α, and δ. The decoding method NNR_PT_BLOCK provides the option to signal that batch norm parameters have been folded (by indicating their presence in the bit stream).

F. Local Scaling

Local scaling adds an additional scaling factor to each row vector of the weight tensor (represented as 2D matrix so that each row corresponds to a neuron as also done for batch norm folding). This gives the encoder the option to partly compensate the error introduced by quantizing the weight tensor. The scaling factors are given as a transposed vector s and they can be merged with parameter α of batch norm folding by updating α as follows:

$$\alpha \coloneqq \alpha \circ s \tag{11}$$

Consequently, when batch norm parameters are present and folded, local scaling doesn't introduce a new parameter and the compressed size of the model is virtually unchanged while the model capacity is increased. If an encoder has the ability to fine-tune the model, its capacity can further be increased by only fine-tuning the local scaling parameters after quantization of the weight tensors. For example, local scaling factors can be initialized with a value of 1 and then adapted by means of backpropagation so that the prediction performance of the model is increased.

VI. Quantization

Similar to other coding standards, quantization is used in NNR for controlling the rate-distortion tradeoff, i.e. finding the lowest bitrate at a given accuracy, or vice versa finding the accuracy for a given bit rate. Depending on the type, NNs may already contain inherent integer quantization to be efficiently processed. Thus, in NNR, either an external or internal quantization method is applied. In the first case, the external quantization is directly used, the quantized network is further processed by entropy coding, and associated quantization control parameters are parsed with the NNR bitstream. In the second case, one of the following NNR quantization methods is applied, as described in the following sub-sections.

A. Uniform Nearest Neighbor Quantization

As a straight forward approach, the NNR standard provides nearest neighbor quantization with a uniform reconstruction quantizer (URQ). In a URQ, the reconstruction levels are uniformly spaced and each level is associated to an integer quantization index. Here, the admissible reconstruction levels are integer multiples of a quantization step size Δ. Implicitly, these integer values uniquely identify the quantizer level. Hence, the quantization indices are chosen, such that they directly correspond to the reconstruction level, i.e. zero corresponds to 0, -1 to $-\Delta$ and 1 to Δ, etc.

An input value to be quantized is then mapped to the reconstruction level that is closest the current value and thus the one which minimizes the distortion.

B. Codebook Quantization

In the NNR codebook quantization, each entry of a given parameter tensor is quantized using vector quantization wherein each parameter is assigned to the values in a finite size codebook. Thus, a parameter tensor is represented by its codebook and a mapping tensor with the same size as the parameter. The codebook of size k and the corresponding mapping is obtained by using k-means clustering algorithm on the vectorized parameter tensor. The size of the codebook is chosen based on rate-distortion tradeoff. For a given distortion, the codebook of minimum size is chosen that permits the required distortion. Each parameter tensor has its own distortion and codebook.

The k-means clustering algorithm requires initial codebook values. Typically, this is obtained by computing the empirical cumulative density function (CDF) of the given parameter tensor. While this initialization leads to better precision for value ranges in high-probability regions, it results in poor approximation in low-probability regions. However, some parameters lying in the low probability region are important for the overall performance of an NN. Therefore, a lower-bounded probability density function (PDF)-based initialization is used in which the PDF function is clipped with a lower bound. This ensures that enough initial codebook values from lower probability regions are available.

C. Dependent Quantization

The new NNR standard supports a vector quantization scheme called dependent scalar quantization (DQ), which is also known as trellis-coded quantization (TCQ) [28]-[31]. It generally achieves a higher compression efficiency at the same performance or distortion level.

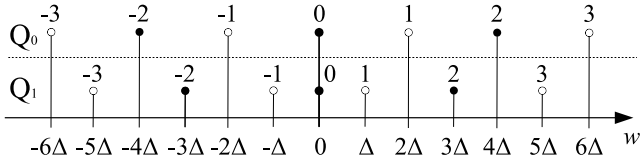


Fig. 3 Quantizer design for dependent quantization.

DQ consists of two major elements. Firstly, two scalar quantizers, Q_0 and Q_1 with distinct sets of reconstruction levels and, secondly, a procedure for switching between them.

As illustrated in Fig. 3 the admissible reconstruction values are represented by integer multiples of a quantization step size Δ for both quantizers, where Q_0 contains all even multiples, Q_1

all odd multiples and both contain zero.

The switching process can be represented by a state machine with 8 states, as illustrated in TABLE II. Each state is associated with one of the scalar quantizers, i.e. states 0, 1, 4, 5 refer to Q_0 and states 2, 3, 6, 7 refer to Q_1. The state transitions are determined by the preceding states and quantization indices, or more precisely a current state value depends on the previous state value and the parity of the previous quantization index. This provides a mechanism to uniquely identify the current state and thus the applied quantizer. Obviously, this process requires the neural network parameters to be reconstructed sequentially in a predefined order which is chosen equally to the coding and scan order. So, given the current state value (and thus the quantizer) as it is output by the state transitioning process, and the transmitted quantization index, the decoder is able to properly reconstruct the current parameter.

TABLE II
STATE TRANSITIONS FOR QUANTIZER SELECTION IN DQ

state		0	1	2	3	4	5	6	7
quantizer used		Q_0	Q_0	Q_1	Q_1	Q_0	Q_0	Q_1	Q_1
next state	parity 0	0	4	5	1	6	2	3	7
	parity 1	4	0	1	5	2	6	7	3

For encoding, the potential transitions can be interpreted as an 8-state trellis. Then, determining the optimal sequence of quantization indices is equivalent to finding the path through the trellis that minimizes a Lagrangian cost function $J = D + \lambda R$ of distortion D (mean squared error) and bitrate R. Since each transition is associated with an RD-cost, the best path is determined using the well-known Viterbi algorithm [32]. For more details the reader is referred to [33].

VII. ENTROPY CODING

In the NNR standard, the quantization indices that are output by the selected quantization method and all other syntax elements are entropy coded using DeepCABAC [34]. This method is based on context-adaptive binary arithmetic coding (CABAC) [35], which was originally developed and optimized for video compression. CABAC is a lossless coding method that provides high compression performance as well as a high flexibility of adaptation and also allows highly efficient implementation, when compared to other entropy coding techniques.

For this, CABAC includes the following three techniques, as described in more detail in the respective subsections: *A.*) Each non-binary symbol or data element to be encoded is decomposed into a series of binary decisions (also called bins), such that a symbol can be uniquely identified. *B.*) A binary probability model (context model) is assigned to each bin. This probability model adapts on-the-fly to the local statistics of the data with each bin that is encoded with the model. *C.*) Finally, each bin is encoded, according to its estimated probability using an arithmetic coding engine.

A. Binarization

A data element is represented by a binary symbol, as required by the binary arithmetic coding engine. Consequently, the non-binary quantization indices need to be decomposed into a series of bins. For this, a parameter tensor is mapped onto a sequence of quantization indices by applying a scan. Then, each quantized neural network parameter is binarized (as depicted in Fig. 4) in the following manner:

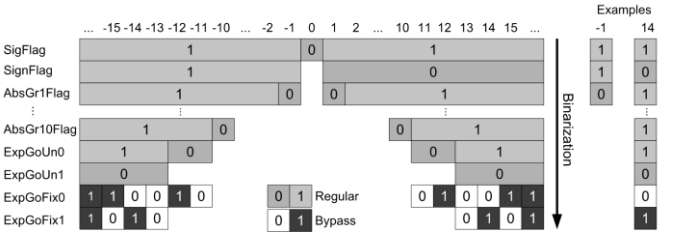


Fig. 4 DeepCABAC binarization scheme.

A first bin, called *SigFlag* (significance flag), determines if the current neural network parameter is significant or not, which means that a bit is assigned which determines if the parameter is 0 or not. Then, if the neural network parameter is not 0, a *SignFlag* denotes the sign of the parameter, which is equal to 1 if the parameter is negative and equal to 0, otherwise. Then, a series of bins, called $AbsGr(n_i)Flag$, are similarly encoded, which determine whether the quantization index is greater than $n_i = 1, 2, \dots, 10$. Thus, whenever an $AbsGr(n_i)$ *Flag* equals 1 it is followed by the next $AbsGr(n_{i+1})$ *Flag*, otherwise, when the flag equals 0, encoding of the current parameter is terminated and no further bits are transmitted. If *AbsGr10Flag* is 1, a reminder is present and encoded using an exponential Golomb code [36], composed of a unary and fixed part. For clarity, the bins of the unary part are denoted as $ExpGoUn(l_a)$ and the bins of the fixed part are denoted as $ExpGoFix(k_a)$. The binarization scheme and two examples are depicted in Fig. 3.

B. Context modeling

Two types of bins, bypass and regular bins, are coded as follows: For bypass bins, one bit per bin is written to the bitstream, e.g. as processed for the fixed part of the exponential Golomb code ($ExpGoFix(k_a)$). Each regular bin is associated with a probability model. The number of bits output by the arithmetic coding engine depends on the estimated probability and is usually lower than one bit per bin on average. Consequently, the compression efficiency highly depends on accurately estimated probabilities. To address this, bins with similar statistics are assigned to the same probability model, which is also known as context modeling. Those bins are grouped based on previously coded bins or a local context as follows: For the syntax element *SigFlag* a set of either three or up to 24 context model candidates is assigned. First, three cases are distinguished by the value of the left neighboring quantization index, i.e. whether it is negative, zero or positive. Then, for the non-DQ case, one context model for each case is assigned. In contrast, for DQ, 8 context models are assigned for each of the three cases, according to the value of the variable 'state' (0…7) as given in TABLE II.

The *SignFlag* employs three context model candidates based on the left neighboring quantization index, analogously to the *SigFlag*. For each of the $AbsGr(n_i)$ *Flags* and $ExpGoUn(l_a)$ a set of two possible candidates is used. The chosen model is

determined by the value 0 or 1 of the preceding *SignFlag*.

C. Arithmetic coding

The coding engine processes bypass and regular bins as follows: For each bypass bin a bit is written to the bitstream, which corresponds to a probability estimate of 0.5. The value of this bit reflects the value of the respective bin. In fact, the arithmetic coding engine is bypassed in this case, which also improves throughput.

For all regular bins, a binary arithmetic coder is employed, i.e. a series of bins is encoded into a single integer value. The integer value represents the outcome of an iterative interval subdivision process. Beginning with a starting interval, for each bin, the respective interval is subdivided according to the probability estimate of the bin being equal to 1 or 0. Then the interval that corresponds to the bin value is chosen and input to the subdivision process for the next bin. Finally, an integer number is chosen, which uniquely determines the outcome interval. Analogously, the bins can be obtained at the decoder by performing the subdivision process synchronously.

VIII. Compression Performance

The NNR compression performance is evaluated for a verification dataset of different neural networks as defined in [37]. An overview of the models is given in [38], which also highlights the corresponding use cases, performance measures, application data and number of parameters. The dataset includes three models (VGG16, ResNet50, MobileNetV2) for image classification, one model (DCase) for audio classification and an image autoencoder (UC12B). The experiments were carried out, using the standard reference software NCTM (Neural network Compression Test Model, version 6.0) [39].

TABLE III
NNR Transparent Coding Results

Model	c_r in %	Top-1 / Top-5 Acc. reconstr.	Top-1 / Top-5 Acc. original	Orig. size (bytes)
VGG16	2.98	70.51 / 89.54	70.93 / 89.85	553.43 M
ResNet50	6.54	74.42 / 91.80	74.98 / 92.15	102.55 M
MobileNetV2	12.18	71.13 / 90.06	71.47 / 90.27	14.16 M
DCase	4.12	58.15 / 92.35	58.27 / 91.85	467.26 k
Model	c_r in %	PSNR / SSIM reconstructed	PSNR / SSIM original	Orig. size (bytes)
UC12B	17.34	29.98 / 0.954	30.13 / 0.956	304.72 k

In TABLE III, the transparent coding results at working points with same classification quality of original and reconstructed pre-trained NNs for Top-1 / Top-5 accuracies and Peak Signal-to-Noise Ratio (PSNR) / Structural Similarity Index Measure (SSIM) for UC12B are given. TABLE III shows that a compression ratio c_r of less than 3% (or vice versa a compression efficiency c_e of more than 97%) can be achieved without accuracy degradation. As an example, the compressed bitstream size is only 2.98% for VGG16 and 4.12% for DCase in comparison to the original size at the same quality.

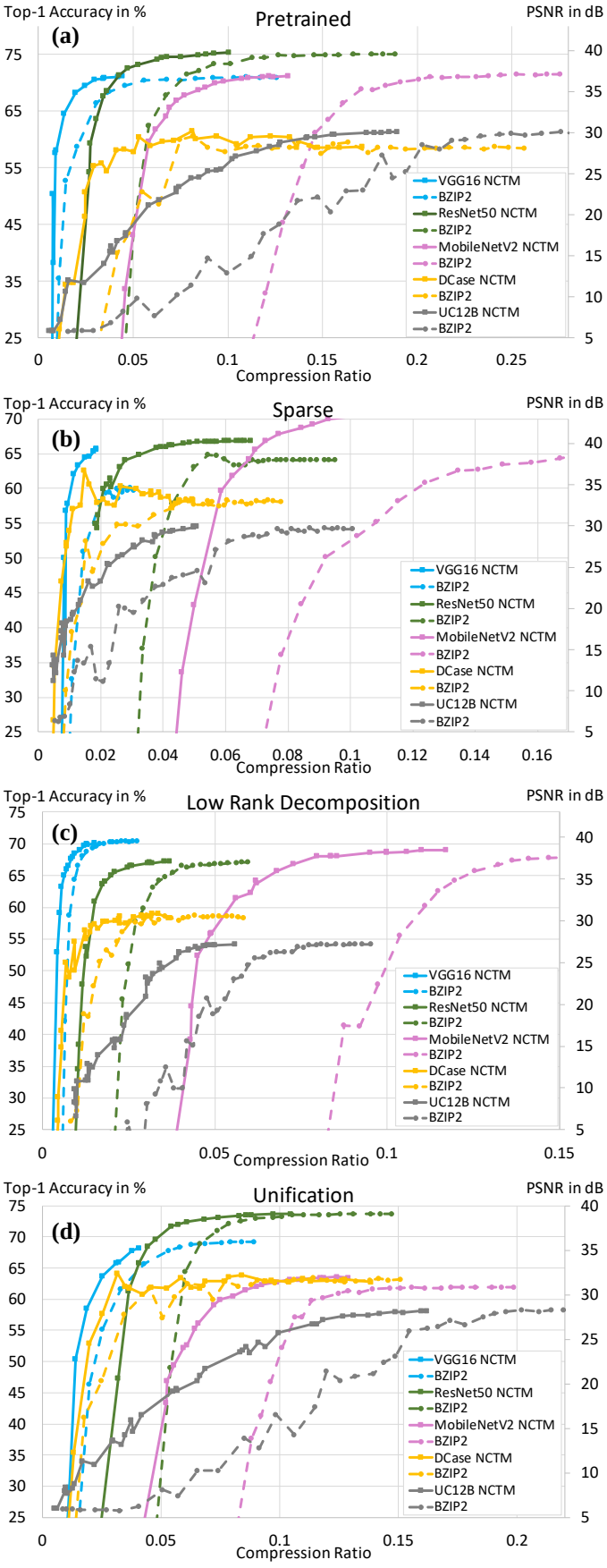


Fig. 5 Compression Ratio-Performance curves for (a) original, (b) sparse, (c) low-rank and (d) unified models for Top-1 Accuracies, and PSNR for UC12B.

Additional benchmark results for different preprocessing methods from section V, i.e. pretraining, sparsification, low rank decomposition and unification, are depicted in Fig. 5 (a) to (d) respectively. Each figure shows compressing ratio c_r with respect to performance measure (Top-1 Accuracy for classification models and PSNR for the image autoencoder) at different working points. These working points are determined by a quantization parameter (QP) that controls the quantization step size. The NNR standard results (NCTM) are denoted by a solid line. A second reference method, that applies uniform quantization according to section VI.A and bzip2 [40] for compression of the quantization indices is shown as dashed lines. Fig. 5 shows, that NNR achieves high compression even for high performance qualities and outperforms comparable methods, significantly. As an example, c_r = 2.98% for VGG16 NCTM, while for VGG16 BZIP c_r = 7.74%, as given in [34]. The graphs also show that much higher compression ratios (far below 3%) of compressed NNs can be achieved for lossy coding scenarios, i.e. when performance decreases are allowed.

IX. Conclusion

The NNR standard for efficient compression of neural networks has been developed and standardized by ISO/IEC MPEG. The standard is developed as a toolbox, and appropriate coding pipelines can be created from the included methods. It can be used either as an independent coding framework or together with external neural network formats and frameworks. For providing the highest degree of flexibility, the network compression methods operate per parameter tensor to always ensure proper decoding, independent of a respective external framework and even if no structure information is provided. In the independent coding case the neural network structure or connection graph is transmitted internally as part of the NNR specification, whereas in the framework-dependent case, structure information is provided by the respective NN framework.

The codec design includes compression-efficient quantization methods, namely uniform reconstruction, codebook and dependent scalar quantization and the DeepCABAC arithmetic coding method as core encoding and decoding technologies. Next, common neural network preprocessing methods for parameter reduction are specified, including sparsification, pruning, low-rank decomposition, unification, batch norm folding and local scaling. Furthermore, the NNR high-level syntax also supports mechanisms for parallel decoding at block-row or sub-tensor level.

NNR achieves a compression efficiency c_e of up to 97% for transparent coding cases, i.e. without degrading classification quality, such as top-1 or top-5 accuracies. In addition, much higher coding gains can be obtained for accuracy-lossy coding cases.

Acknowledgment

The authors would like to thank the experts of ISO/IEC MPEG and in particular the MPEG NNR group for their contributions.

References

[1] K. Ota, M. S. Dao, V. Mezaris, and F. G. B. D. Natale, “Deep learning for mobile multimedia: A survey”, *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 13, no. 3s, pp.34:1–34:22, 2017.

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” 2016, arXiv:1602.05629. [Online]. Available: http://arxiv.org/abs/1602.05629.

[3] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek “Robust and Communication-Efficient Federated Learning from Non-IID Data”, IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 9, pp. 3400-3413, September 2020, doi: 10.1109/TNNLS.2019.2944481.

[4] Open Neural Network Exchange, VERSION 6, 2019-09-19 (https://github.com/onnx/onnx/blob/master/onnx/onnx.proto)

[5] Neural Network Exchange Format, The Khronos NNEF Working Group, Version 1.0.3, 2020-06-12 (https://www.khronos.org/registry/NNEF/specs/1.0/nnef-1.0.3.pdf).

[6] M. Denil et al. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, 2013.

[7] Ziqian Chen, Shiqi Wang, Dapeng Oliver Wu, Tiejun Huang, and Ling-Yu Duan. From Data to Knowledge: Deep Learning Model Compression, Transmission and Communication. Proc. ACM MM, 2018.

[8] S. Ha, H. Mao and W. J. Dall, “Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding,” International Conference on Learning Representations, 2016.

[9] D. Blalock, “What is the State of Neural Network Pruning?” in Proceedings of Machine Learning and Systems (MLSys), 2020.

[10] J. Frankle and M. Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks." International Conference on Learning Representations. 2018.

[11] E. Malach, et al. "Proving the Lottery Ticket Hypothesis: Pruning is All You Need." arXiv preprint arXiv:2002.00585 (2020).

[12] P. Savarese, H. Silva, and M. Maire. "Winning the lottery with continuous sparsification." Advances in Neural Information Processing Systems 33 (2020).

[13] A. Keller at al., “Structural Sparsity - Speeding up Training and Inference of Neural Networks by linear Algorithms,” GPU Technology Conference, 2019, https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9389-structural-sparsity-speeding-up-training-and-inference-of-neural-networks-by-linear-algorithms.pdf

[14] C. Louizos, et al, “Relaxed quantization for discretized neural networks,” In International Conference on Learning Representations (ICLR), 2019.

[15] D. Zhang, Dongqing, et al. “Lq-nets: Learned quantization for highly accurate and compact deep neural networks,” Proceedings of the European conference on computer vision (ECCV), 2018.

[16] J. Chen, et al. "Propagating Asymptotic-Estimated Gradients for Low Bitwidth Quantized Neural Networks." IEEE Journal of Selected Topics in Signal Processing (2020).

[17] Y. Umuroglu et al., “FINN: A Framework for Fast, Scalable Binarized Neural Network Inference”, 25th International Symposium on Field-Programmable Gate Arrays, 2017.

[18] https://developer.nvidia.com/tensorrt

[19] C. Bucila, R. Caruana, and A. Niculescu-Mizil, “Model compression,” In ACM SIGKDD, pp. 535–541, 2006.

[20] D. Stamoulis et al., “Single-Path Mobile AutoML: Efficient ConvNet Design and NAS Hyperparameter Optimization,”. IEEE J. Sel. Top. Signal Process. 14(4): 609-622, 2020.

[21] K. Bhardwaj, N. Suda, and R. Marculescu, “Dream distillation: A data-independent model compression framework,“ Joint Workshop on On-Device Machine Learning & Compact Deep Neural Network Representations at ICML 2019.

[22] Y. Wang, et al., "Dual dynamic inference: Enabling more efficient, adaptive and controllable deep inference." IEEE Journal of Selected Topics in Signal Processing (2020).

[23] H. Cai et al., “Once for All: Train One Network and Specialize it for Efficient Deployment”, International Conference on Learning Representations, 2020.

[24] T. Ben-Nun, “A Modular Benchmarking Infrastructure for High-Performance and Reproducible Deep Learning,” IEEE International Parallel & Distributed Processing Symposium, 2019.

[25] B. Liu, M. Wang, H. Foroosh, M. Tappen and M. Penksy, "Sparse Convolutional Neural Networks," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[26] P. O. Hoyer, “Non-negative matrix factorization with sparseness constraints”, Journal of Machine Learning Research, 5:1457-1469, 2004.
[27] C. Aytekin, F. Cricri, and E. Aksu, “Compressibility loss for neural network weights”, in arXiv:1905.01044, 2019. 2, 3.2.
[28] M. W. Marcellin and T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," in IEEE Transactions on Communications, vol. 38, no. 1, pp. 82-93, Jan. 1990, doi: 10.1109/26.46532.
[29] J. H. Kasner, M. W. Marcellin and B. R. Hunt, "Universal trellis coded quantization," in IEEE Transactions on Image Processing, vol. 8, no. 12, pp. 1677-1687, Dec. 1999, doi: 10.1109/83.806615.
[30] T. R. Fischer and M. Wang, "Entropy-constrained trellis coded quantization," [1991] Proceedings. Data Compression Conference, Snowbird, UT, USA, 1991, pp. 103-112, doi: 10.1109/DCC.1991.213377.
[31] H. Schwarz, T. Nguyen, D. Marpe and T. Wiegand, "Hybrid Video Coding with Trellis-Coded Quantization," 2019 Data Compression Conference (DCC), Snowbird, UT, USA, 2019, pp. 182-191, doi: 10.1109/DCC.2019.00026.
[32] G. D. Forney, "The viterbi algorithm," in Proceedings of the IEEE, vol. 61, no. 3, pp. 268-278, March 1973, doi: 10.1109/PROC.1973.9030.
[33] P. Haase et al., "Dependent Scalar Quantization For Neural Network Compression," 2020 IEEE International Conference on Image Processing (ICIP), Abu Dhabi, United Arab Emirates, 2020, pp. 36-40, doi: 10.1109/ICIP40778.2020.9190955.
[34] S. Wiedemann et al., "DeepCABAC: A Universal Compression Algorithm for Deep Neural Networks," in IEEE Journal of Selected Topics in Signal Processing, vol. 14, no. 4, pp. 700-714, May 2020, doi: 10.1109/JSTSP.2020.2969554.
[35] D. Marpe, H. Schwarz and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 620-636, July 2003, doi: 10.1109/TCSVT.2003.815173.
[36] Jukka Teuhola, “A compression method for clustered bit-vectors.”, Inf. Process. Lett., vol. 7, pp. 308-311, Oct. 1978
[37] “Description of Core Experiments on Compression of neural networks for multimedia content description and analysis”, MPEG document N18574, ISO/IEC JTC 1/SC 29/WG 11, Jul. 2019.
[38] “Evaluation Framework for Compressed Representation of Neural Networks”, MPEG document N17929, ISO/IEC JTC 1/SC 29/WG 11, Oct. 2018.
[39] “Test Model 6 of Compression of Neural Networks for Multimedia Content Description and Analysis”, MPEG document N00017, ISO/IEC JTC 1/SC 29/ WG 04, Oct. 2020
[40] Bzip2, Version 1.0.8, 2020-12-03 (https://sourceware.org/bzip2/)

Heiner Kirchhoffer received the Dipl.-Ing. (FH) degree in television technology and electronic media from the Wiesbaden University of Applied Sciences, Wiesbaden, Germany, in 2005 and the Dr.-Ing. degree from University of Rostock, Rostock, Germany, in 2016. In 2004, he joined the Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin, Germany, where he is currently research associate in the Video Coding & Analytics Department.

Dr. Kirchhoffer has contributed successfully to the H.265/High Efficiency Video Coding (HEVC) and to the H.266/Versatile Video Coding (VVC) standardization activities of the ITU-T Video Coding Experts Group and the ISO/IEC Moving Pictures Experts Group. His research interests include image and video compression, entropy coding, and efficient representations of deep neural networks.

Paul Haase received the B.Sc. degree and the M.Sc. degree in Electrical Engineering from the Technical University of Berlin, Germany, in 2012 and 2015, respectively. In 2013, he joined the Fraunhofer Institute for Telecommunications-Heinrich Hertz Institute, Berlin, Germany, where he is currently research associate in the Video Coding & Analytics Department. He successfully contributed to the H.266/Versatile Video Coding (VVC) standardization activity of the ITU-T Moving Pictures Experts Group. His research interests include efficient representations of deep neural networks, image and video compression and entropy coding.

Wojciech Samek (M’13) received the Dipl.-Inf. degree in computer science from the Humboldt University of Berlin, Berlin, Germany, in 2010, and the Dr. rer. nat. degree from the Technical University of Berlin, Berlin, in 2014. He was a Visiting Researcher with the NASA Ames Research Center, Mountain View, CA, USA. He is currently the Head of the Artificial Intelligence Department, Fraunhofer Heinrich Hertz Institute, Berlin. He is also a PI with the Berlin Institute for the Foundations of Learning and Data (BIFOLD), Berlin, a member of the European Laboratory for Learning and Intelligent Systems (ELLIS), and an Associate Faculty Member with the DFG Graduate School BIOQIC, Berlin.

He has coauthored more than 100 peer-reviewed journal articles and conference papers, predominantly in the areas of deep learning, explainable AI, neural network compression, and federated learning.

Dr. Samek is also an Editorial Board Member of Digital Signal Processing, PLOS ONE, Pattern Recognition and the IEEE Transactions on Neural Networks and Learning Systems, and an elected member of the IEEE MLSP Technical Committee. During his studies, he was awarded scholarships from the Studienstiftung des deutschen Volkes and the DFG Research Training Group GRK1589/1. He is involved in the standardization of MPEG-7 Part 17 and was an organizer of special sessions, workshops, and tutorials on topics such as explainable AI and federated learning at top-tier machine learning and signal processing conferences.

Karsten Müller (M’98–SM’07) received the Dr. Ing. degree in Electrical Engineering and Dipl. Ing. degree from the Technical University of Berlin, Germany, in 2006 and 1997 respectively. He has been with the Fraunhofer Institute for Telecommunications, Heinrich-Hertz-Institut, Berlin, since 1997, where he is currently head of the Efficient Deep Learning group. His research interests include multi-dimensional video coding, efficient federated learning and compression of neural networks.

Dr. Müller has been involved in international standardization activities, successfully contributing to the ISO/IEC Moving Picture Experts Group for work items on visual media content description, multiview, multi-texture, 3D video coding and neural network representation. He co-chaired an adHoc group on 3D Video Coding from 2003 to 2012. Dr. Müller has served as chair and editor in IEEE conferences and associated editor of the IEEE Transactions on Image Processing.

Hamed R. Tavakoli (M'10) received his Dr. Sci. (Tech.) from University of Oulu, Finland in 2014. Before joining Nokia as a Principal Scientist in machine learning, he was with the Aalto University, Finland.

Hamed has been involved in various activities ranging from cognitive vision systems to natural language processing and multimodal information processing.

Francesco Cricri received the PhD in Signal Processing from Tampere University of Technology, Finland, in 2014, and the MSc degree in Telecommunications Engineering from the University of Trento, Italy. He has been with Nokia Technologies, Tampere, Finland, since 2014, where he is currently a principal researcher in machine learning, leading various projects in the field of machine learning, computer vision, and data compression. He is also a Nokia delegate to ISO/IEC Moving Picture Experts Group (MPEG for the standardization of neural network compression technologies.

His current research interests include neural network compression, neural network based image and video compression, few-shots learning, meta-learning, and evaluation of generative models.

Dr. Cricri has published over 24 research papers in international scientific conferences and journals, filed over 90 patent applications and holds over 25 granted patents. He was a recipient of the Best Student Paper Award at the 2011 IEEE International Symposium on Multimedia, and a recipient of the Most Innovative Image Codec Award at the 2020 IEEE International Workshop on Multimedia Signal Processing.

Emre B. Aksu is a research leader at Nokia Technologies, Tampere, Finland. He received his BSc degree in 1995 on Electrical and Electronics Engineering from Bilkent University, Ankara, Turkey, MSc degree in 1998 on Signal Processing from Middle East Technical University, Ankara, Turkey.

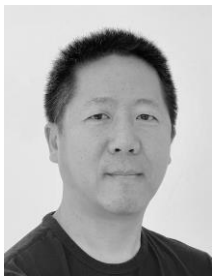
His research team's focus is on emerging media systems and related systems standardization, cloud-based media processing technologies, and media-based applications of machine learning. As a standardization delegate of Nokia, he actively contributes to MPEG systems standards on topics such as coded representation of neural networks (NNR), file formats [ISO Base Media File Format (ISOBMFF), High Efficiency Image File Format (HEIF)], omnidirectional application format (OMAF) and carriage of V3C coded data.

Miska M. Hannuksela (M'03) received his M. Sc. (Eng.) and D. Sc. (Tech.) degrees from Tampere University of Technology, Finland, in 1997 and 2010, respectively.

He has been with Nokia since 1996 in different roles including research manager/leader positions in the areas of video and image compression, end-to-end multimedia systems, as well as sensor signal processing and context extraction. Currently he works as Bell Labs Fellow and Head of Video Research at Nokia Technologies, Tampere, Finland.

Dr. Hannuksela has published over 170 journal and conference papers and more than 1000 standardization contributions in JVET, JCT-VC, JVT, MPEG, 3GPP, and DVB. He has been an editor in several video and systems standards, including High Efficiency Image File Format (HEIF), Omnidirectional Media Format, RFC 3984, RFC 7798, as well as some parts of H.264/AVC, H.265/HEVC, and ISO Base Media File Format. He has granted and pending patents from more than 430 patent families. His research interests include video compression as well as multimedia communication systems and formats. Dr. Hannuksela has several best paper awards and received an award of the best doctoral thesis of Tampere University of Technology in 2009 and Scientific Achievement Award nominated by the Centre of Excellence of Signal Processing, Tampere University of Technology, in 2010. He was an Associate Editor in IEEE Transactions on Circuits and Systems of Video Technology from 2010 to 2015.

Wei Jiang (M'04) received her Ph.D degree in 2010 from the E.E. Department of Columbia University. She got her M.S. and B.E. degree in Automation from Tsinghua University, in 2005 and 2002, respectively. Wei is currently a Principal Researcher in Tencent America. Her research focuses on computer vision and machine learning, including video understanding, object detection/recognition, image restoration, visual computing, deep neural network compression and acceleration, deep feature compression and representation learning. Wei has published over 30 papers in top tier conferences and journals and holds over 20 patents. Wei was elected as IBM's Watson Emerging Leaders in Multimedia in 2008.



Wei Wang (M'20) Wei received his M.S. and B.S. degree from E.E. Department of Fudan University, China, in 1998 and 1995, respectively. He is currently a Principal Researcher in Tencent America. He was formerly a Senior Staff Researcher at Futurewei Technologies. His research interests include high performance computing, image and video compression, neural networks compression and acceleration,

and converting algorithms to product on desktop, embedded or ASIC platforms.

He has been involved in international standardization activities, including contributing to ISO/IEC Moving Picture Experts Group for work items on H.265/HEVC SCC and neural network representation, and to IEEE Data Compression Standards Committee for work item on neural image coding. He served as co-Chair of MPEG neural network representation group.

Shan Liu (M'01–SM'11) received the B.Eng. degree in Electronic Engineering from Tsinghua University, the M.S. and Ph.D. degrees in Electrical Engineering from the University of Southern California, respectively. She is currently a Tencent Distinguished Scientist and General Manager of Tencent Media Lab. She was formerly Director of Media Technology Division at MediaTek USA. She was also formerly with MERL, Sony and IBM, etc.

Dr. Liu has been actively contributing to international standards since the last decade and served co-Editor of H.265/HEVC SCC and H.266/VVC. She has numerous technical proposals adopted into various standards, such as VVC, HEVC, OMAF, DASH and PCC, etc. Technologies and products developed by her and under her leadership have served hundreds of millions of users. Dr. Liu holds more than 200 granted US and global patents and has published more than 80 journal and conference papers. She has been on the Editorial Board of IEEE Transactions on Circuits and Systems for Video Technology since 2018 and received the Best AE Award in 2019. Her research interests include audio-visual, high volume, immersive and emerging media compression, intelligence, transport and systems.

Swayambhoo Jain (M'13) is a researcher at Interdigital AI Lab in Palo Alto, California, USA. He received the Bachelor of Technology degree in Electronics and Communication Engineering from the National Institute of Technology, Calicut, India in 2007. He was a Research and Development Engineer with IBM Systems and Technology Group, Bangalore, India, from 2007 to 2010. He received the M.Sc. degree and Ph.D. degree in Electrical Engineering from the University of Minnesota, MN, USA, in 2012 and 2017, respectively. He worked as Researcher at Technicolor AI Lab in Palo Alto, CA, USA from 2017 till 2019. His general research interests lie in Machine Learning, Signal Processing, and High Dimensional Statistical Inference. His current research focus in on issues related to efficiency and robustness of deep neural networks.

Shahab Hamidi-Rad is a Software Architect at Interdigital AI Lab in Palo Alto, California, USA. Before Interdigital he worked at the Technicolor AI research lab for 4 years. His areas of work and interest include machine learning, deep learning, big data, cloud computing, and software design and development. Before Technicolor, Shahab worked at Harmonic Inc. for 9 years, during which he developed several different video encoding products. His work included design and implementation of a statistical multiplexing system for video broadcast transport streams. During 2001 to 2006, he was with Animatics Corporation, where he worked on controlling smart stepper/servo motors in real-time. He designed and implemented a G-Code based Servo Motion Control System that delivered precise multi-axis contouring.

Fabien Racapé is a senior scientist at the InterDigital AI Lab. He received an engineering degree in physics and a M.Sc. in signal processing and telecommunications from the Grenoble Institute of Technology, France in 2008, and a PhD degree from the National Institute of Applied Science (INSA), Rennes, France in 2011. Prior to InterDigital, Fabien was a researcher at Technicolor R&I France from 2014 to 2019. He also worked at and the French Institute for Research in Computer Science and Control (INRIA) as a research engineer and at the Fraunhofer Institute for Telecommunications – Heinrich-Hertz-Institut, Berlin, Germany, as a post-doctoral fellow (Marie Curie COFUND grant) in 2013 and 2014. He has been involved in H.266/Versatile Video Coding (VVC) and MPEG neural network compression standardization activities. His current research interests include neural network-based video compression, 3D and omnidirectional video coding, as well as compressed neural network representations.

Werner Bailer (M'12) is a Key Researcher at DIGITAL – Institute for Information and Communication Technologies at JOANNEUM RESEARCH in Graz, Austria. He received the Dipl. Ing. degree in Media Technology and Design in 2002 for his diploma thesis on motion estimation and segmentation for film/video standards conversion.

His research interests include audiovisual content analysis and retrieval and machine learning. He regularly contributes to standardization, among others in MPEG, where he co-chairs the ad-hoc group on neural network compression.