Supplementary Material

A. Reference Quantization by Rounding (INT)

As reference, we use integer-rounding-based (INT) quantization as, *e.g.*, in [5, 6, 27], being depicted in Figure A.1, where during inference the input data is simply rounded to the nearest integer value element-wise. During training, however, no quantization is applied but the quantization error is simulated by addition of random uniform noise $n \sim \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$. To allow fair comparison, the same encoder and decoder architectures are applied both for INT and Ω quantization.

As for Ω quantization, also for INT quantization the average bitrate is determined by the entropy model described in Section B, as is done by [5]. Further, we again control the average bitrate with our proposed feature map masking function $\mathbf{M}()$ varying the number of feature maps in the bottleneck to perform rate adaptation during inference. The bitrate calculation from Section 3.3 remains applicable, but the mean cross entropy (MCE) receives only $\boldsymbol{P}(\hat{\boldsymbol{r}}')$ as single input and is now (instead of (8))

$$\text{MCE}(\hat{\boldsymbol{r}}', \boldsymbol{P}) = -\sum_{i\in\mathcal{I}} \frac{\log_2(P_i)}{H_{\mathcal{X}} \cdot W_{\mathcal{X}}}, \tag{A.1}$$

with the probability estimate $\boldsymbol{P} = (P_i) \in [0,1]^{H_{\mathcal{R}}\times W_{\mathcal{R}}\times C_{\mathcal{R}}}$, $i \in \mathcal{I} = \{1, ..., H_{\mathcal{R}} \cdot W_{\mathcal{R}} \cdot C_{\mathcal{R}}\}$ and

$$P_i = \int_{\hat{r}'_i-\frac{1}{2}}^{\hat{r}'_i+\frac{1}{2}} p(x)\,\mathrm{d}x \tag{A.2}$$

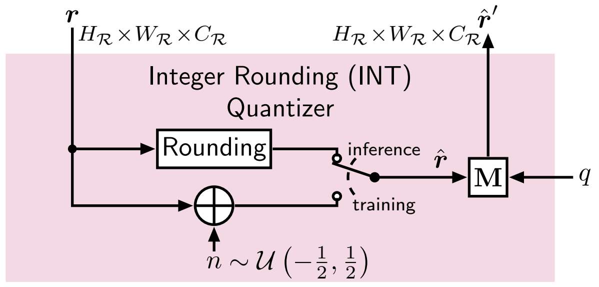
$$= F(\hat{r}'_i + \tfrac{1}{2}) - F(\hat{r}'_i - \tfrac{1}{2}) \tag{A.3}$$

integrating over the univariate probability density function (PDF) $p(x)$, resulting in a difference of cumulative distribution functions (CDFs) $F()$. Only the non-masked feature maps count into the bitrate calculation. While in the Ω quantizer, the quantized feature map pixels $\hat{\boldsymbol{z}}_i$ and $\hat{\boldsymbol{z}}'_i$ are S-dimensional and therefore require a multivariate entropy model, in the INT reference $\hat{r}_i$ and $\hat{r}'_i$ are scalar and employed in the integration limits over the univariate $p(x)$.

B. Entropy Models

Since the bottleneck representations are different for Ω and INT quantizers, also the entropy models with their inputs and outputs differ to allow optimal interaction with the quantizers.

Entropy Model for the One-Hot Max Quantizer. The entropy model used for Ω quantization can be seen in Figure B.1. We use a convolutional neural network (CNN) with an input tensor $\mathbf{1} \in \{1\}^{H_{\mathcal{R}}\times W_{\mathcal{R}}\times 1}$ consisting of ones with resolution $H_{\mathcal{R}} \times W_{\mathcal{R}}$ to generate an intermediate output having the same dimensions as $\hat{\boldsymbol{r}}$. The subsequent probability (Prob) layer is similar to the first part of the Ω quantizer: First, the number of feature maps $C_{\mathcal{R}}$ is adjusted to C' by a 1×1 convolution, second, the last dimension is split into $C_{\mathcal{Z}}$ bottleneck feature maps and the quantizer dimension S, and third, the representation is scaled with the learnable parameter ξ to adjust the hardness of the softmax output, and processed by the softmax function to obtain values in the interval $[0,1]$ which add up to 1 over the quantizer dimension. The output of the entropy model has the same dimensionality as $\hat{\boldsymbol{z}}$ and yields the probability of the occurrence of a 1 in $\hat{\boldsymbol{z}}$ for each feature map pixel and quantizer dimension.



$\boldsymbol{r}$
$H_{\mathcal{R}}\times W_{\mathcal{R}}\times C_{\mathcal{R}}$
$H_{\mathcal{R}}\times W_{\mathcal{R}}\times C_{\mathcal{R}}$
$\hat{\boldsymbol{r}}'$
Integer Rounding (INT) Quantizer
Rounding
inference
training
$\hat{\boldsymbol{r}}$
M
q
$n \sim \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$

Figure A.1. **Integer-rounding-based (INT) quantizer** as being used in Figure 2 as **reference** approach (replacing the Ω quantizer) with the residual $\boldsymbol{r}$ as input and the masked quantized residual $\hat{\boldsymbol{r}}'$ as output. During inference, the bottleneck feature map data $\boldsymbol{r} = (r_i)$ is rounded element-wise (r_i) to the nearest integer $\hat{r}_i$, while during training, the quantization error is simulated by uniform noise $n \sim \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)$.

Entropy Model for the Integer Rounding Quantizer. The univariate entropy model for the INT quantization is adopted from [6] and can be seen in Figure B.2. Here, the probability estimate is advantageously obtained from the difference of the cumulative distribution functions $F()$ (CDFs) at the decision boundaries, which always lie at $\hat{r}'_i \pm \frac{1}{2}$ for the INT quantizer. The quantized bottleneck representation $\hat{\boldsymbol{r}}'$ is used as input of the entropy model, resulting in $\boldsymbol{P} = \boldsymbol{P}(\hat{\boldsymbol{r}}')$. With the flattened feature map pixels $\boldsymbol{x} \in \mathbb{R}^{H_{\mathcal{R}}\cdot W_{\mathcal{R}}}$ from one feature map, each filter of the CDF uses the filter function

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{g}(\text{softplus}(\boldsymbol{H})\boldsymbol{x}^{\mathsf{T}} + \boldsymbol{B}) \tag{B.1}$$

with the function $\text{softplus}(h) = \ln(1 + e^h)$ — also called smooth rectifier unit, guaranteeing positive outputs — being applied element-wise and the columns of the matrix $\boldsymbol{B} = (B_{k,\ell}) = (\boldsymbol{b})$ consisting of equal parameter vectors $\boldsymbol{b} = (b_k)$. Here,

$$\boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{y} + \tanh(\boldsymbol{A}) \odot \tanh(\boldsymbol{y}) \tag{B.2}$$

is the activation function, where $\odot$ is the element-wise multiplication, the columns $\boldsymbol{a} = (a_k)$ of the matrix $\boldsymbol{A} =$

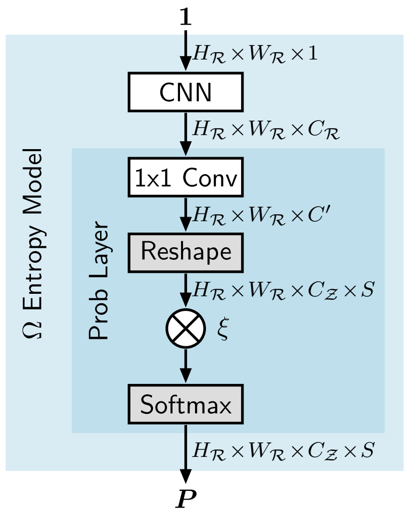

1
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times 1$
CNN
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C_{\mathcal{R}}$
1x1 Conv
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C'$
Reshape
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C_{\mathcal{Z}} \times S$
ξ
Softmax
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C_{\mathcal{Z}} \times S$
$\boldsymbol{P}$
Ω Entropy Model
Prob Layer

Figure B.1. **Ω quantization entropy model** being embedded in Figure 2. For each pixel in the quantized bottleneck representation $\hat{\boldsymbol{z}}' = (\hat{z}'_{i,s})$, the entropy model provides the probability $P_{i,s}$ of $\hat{z}'_{i,s}$ being 1, in total $\boldsymbol{P} = (P_{i,s})$.

$(A_{k,\ell}) = (\boldsymbol{a})$ consisting of equal parameter vectors, and all operations are performed for each feature map separately. The CDF is completed by a sigmoid activation function. The parameter matrix $\boldsymbol{H}$ and parameter vectors $\boldsymbol{a}$, $\boldsymbol{b}$ have different dimensionalities for the individual filters, such that the data dimensions noted in Figure B.2 result, *e.g.*, for the first filter $\boldsymbol{f}(\boldsymbol{x})$ we have $\boldsymbol{H} \in \mathbb{R}^{3\times1}$, $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$, $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3\times H_{\mathcal{R}} \cdot W_{\mathcal{R}}}$. For each feature map, an individual set of the parameters $\boldsymbol{H}$, $\boldsymbol{a}$, $\boldsymbol{b}$ exists, having same dimensionality and being optimized separately during training. For the second and third filters $\boldsymbol{f}()$, we have $\boldsymbol{H} \in \mathbb{R}^{3\times3}$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^3$ within $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3\times H_{\mathcal{R}} \cdot W_{\mathcal{R}}}$. For the last filter $\boldsymbol{f}()$, we have $\boldsymbol{H} \in \mathbb{R}^{1\times3}$ and $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}$ within $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{1\times H_{\mathcal{R}} \cdot W_{\mathcal{R}}}$.

Practical Advantage of Ω Quantization Entropy Model. With the presented entropy model architecture for Ω quantization based on convolutional layers, given a fixed input image size $H_{\mathcal{X}} \times W_{\mathcal{X}}$, — as could be the case, *e.g.*, when processing sensor data — the discrete prior probability $\boldsymbol{P}$ of the Ω quantization entropy model is constant since the input of the entropy model is a constant tensor of ones. Hence, in an application with known fixed input image size, the entropy model output can be pre-stored and read out of memory during inference to save computation resources and storage for the weights of the entropy model (in case the weights require more memory than the prior probability, which depends on the image size). For the INT quantization entropy model, in contrast, pre-storing the probabilities $\boldsymbol{P}(\hat{\boldsymbol{r}}')$ is not possible since they depend on the input and have to be computed during inference.

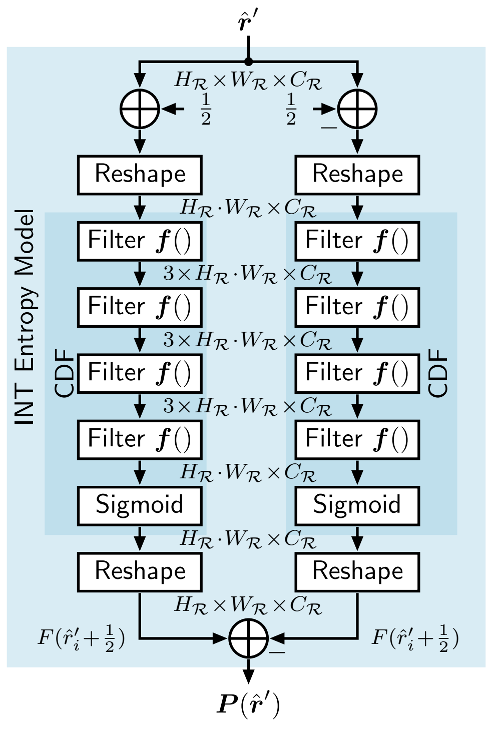

$\hat{\boldsymbol{r}}'$
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C_{\mathcal{R}}$
$\frac{1}{2}$
$\frac{1}{2}$
$-$
Reshape
Reshape
$H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Filter $\boldsymbol{f}()$
$3 \times H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Filter $\boldsymbol{f}()$
$3 \times H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Filter $\boldsymbol{f}()$
$3 \times H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Filter $\boldsymbol{f}()$
$H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Sigmoid
Sigmoid
$H_{\mathcal{R}} \cdot W_{\mathcal{R}} \times C_{\mathcal{R}}$
Reshape
Reshape
$H_{\mathcal{R}} \times W_{\mathcal{R}} \times C_{\mathcal{R}}$
$F(\hat{r}'_i + \frac{1}{2})$
$F(\hat{r}'_i + \frac{1}{2})$
$-$
$\boldsymbol{P}(\hat{\boldsymbol{r}}')$
INT Entropy Model
CDF
CDF

Figure B.2. **INT quantization entropy model** from [6] being embedded in Figure 2 for the reference approach. For each pixel in the quantized bottleneck representation $\hat{\boldsymbol{r}}' = (\hat{r}'_i)$ the entropy model provides the probability P_i of $\hat{r}'_i$, in total $\boldsymbol{P}(\hat{\boldsymbol{r}}') = (P_i)$ by learning a cumulative distribution function (CDF).

C. Datasets and Hyperparameters

MNIST. We use the MNIST data set [20] with small network architectures allowing for a multitude of experimental results and train for a maximum of 1,000 epochs in this setting — depending on early stopping — with a batch size of 512.

Kodak. We also use the OpenImages data set [19] both for training and validation as in [26, 27], where preprocessing follows [26], and follow conventions by testing on the Kodak dataset [10]. Here, we train for a maximum of 10 epochs with a batch size of 32 and a crop size of 256. We pad images with arbitrary given resolutions since input images should have a resolution being dividable by the overall downsampling factor due to downsampling by strided convolutions.

Further Hyperparameters. During training, we set the learning rate of the Adam optimizer [18] to 10^{-3} for MNIST and to 10^{-4} for OpenImages, in both cases with a weight decay of 10^{-5}. The hyperparameter β controlling the hardness of the one-hot max approximation is 100. We did not optimize the seed.

Table D.1. **Model parameters and number of operations** for Ω and INT quantization for the **MNIST architecture**. Bold values are from the settings being used. For INT quantization, $C_{\mathcal{Z}}$ corresponds to $C_{\mathcal{R}}$.

$C_{\mathcal{Z}}$	S	Parameters Ω	Parameters INT	MACs in millions Ω	MACs in millions INT
8	2	16,913	**15,913**	1,066	**1,063**
	4	18,481		1,075	
	8	21,617		1,091	
	16	27,889		1,123	
	32	40,433		1,188	
	256	**216,049**		**2,096**	
16	2	18,489	**16,809**	1,074	**1,068**
	4	21,625		1,091	
	8	27,897		1,123	
	16	40,441		1,188	
	32	65,529		1,318	
	256	**416,761**		**3,134**	
32	2	21,641	**18,601**	1,090	**1,080**
	4	27,913		1,123	
	8	40,457		1,188	
	16	65,545		1,317	
	32	115,721		1,577	
	256	**818,185**		**5,210**	

D. Model Architectures

For MNIST and Kodak, we use different model architectures varying in complexity. We use the notations from [34] and [38] to describe our networks.

MNIST Architectures. Let `c3s1-k` denote a 3×3 convolution-ReLU layer and `t3s1-k` a 3×3 transposed convolution-ReLU layer both with stride 1 and k output feature maps. The encoder can be described as follows: `c3s3-16`, `c3s2-16`, `c3s2-`$C_{\mathcal{R}}$ with $C_{\mathcal{R}} = 32$ if not stated otherwise. The decoder architecture is `t3s2-16`, `t5s2-8`, `t2s2-1` with final tanh activation. The convolutional neural network for the entropy model applied with the Ω quantizer consists of a single `c3s1-`$C_{\mathcal{R}}$ layer. For the subsequent probability layer and the entropy model architecture for the INT quantizer please see Section B.

Kodak (Full-Scale) Architectures. Here, `C3s1-k` and `T3s1-k` denote a 3×3 convolution-*ChannelNorm*-ReLU layer and a 3×3 transposed convolution-*ChannelNorm*-ReLU layer, respectively, both with stride 1 and k output feature maps. We further define a downsampling layer `d-k` to be `C3s2-k`, an upsampling layer `u-k` to be `T3s2-k` with output padding of 1 in all directions, and a ResNet block [15] `R-k` with ChannelNorm. The encoder is then `C7s1-60`, `d-120`, `d-240`, `d-480`, `d-`$C_{\mathcal{R}}$ with $C_{\mathcal{R}} = 960$ if not stated otherwise. The decoder is described as $8 \times$ `R-`$C_{\mathcal{R}}$, `u-480`, `u-240`, `u-120`, `u-60`, `c7s1-3` without

Table D.2. **Model parameters and number of operations** for Ω and INT quantization for the **Kodak architecture**. Bold values are from the settings being used. For INT quantization, $C_{\mathcal{Z}}$ corresponds to $C_{\mathcal{R}}$.

$C_{\mathcal{Z}}$	S	Parameters in millions Ω	Parameters in millions INT	MACs in 10^9 Ω	MACs in 10^9 INT
480	2	150.7	149.4	1,861	1,812
	4	153.5		1,884	
	8	159.0		1,929	
	16	170.1		2,020	
	32	**192.3**		**2,202**	
	64	236.6		2,566	
960	2	153.5	**150.8**	1,884	**1,820**
	4	159.0		1,929	
	8	170.1		2,020	
	16	192.3		2,202	
	32	236.6		2,566	
	64	325.2		3,293	

final normalization and activation. Finally, the CNN for the entropy model for Ω quantization consists of `C3s1-480`, `C3s1-`$C_{\mathcal{R}}$, while the INT quantizer again uses the entropy model described in Section B.

E. Number of Parameters and Complexity

Parameters. As the Ω quantizer transforms the input into the feature map dimension $C_{\mathcal{Z}}$ and the quantizer dimension S, the number of parameters is *naturally* larger than for the INT quantizer, since the INT quantizer does not change the dimensions internally. Tables D.1 and D.2 show the parameters of the models with Ω and INT quantization for the MNIST and Kodak architectures, respectively. In contrast to the Ω architectures, the quantizer dimension S has no effect on the number of parameters of the INT quantizer and entropy models. As can be seen from the tables, the Ω quantizer autoencoder in general uses more parameters than the INT quantizer autoencoder. Considering that the INT quantizer essentially merely rounds values to the nearest integer value and does not require parameters for this, this is not surprising. While in Table D.1 it can be seen that differences are substantial for the playground MNIST architecture (although still all networks are below 1 M parameters), Table D.2 shows *for the practically more relevant Kodak dataset that the autoencoder with Ω quantizer (192.3 M) consumes "only" about 28% more parameters than the one with INT quantizer (150.8 M).* The encoder and decoder have 7,120 and 7,865 parameters, respectively, for the MNIST architecture such that the number of parameters is evenly portioned, and 5,522,400 and 138,276,003 parameters, respectively, for the Kodak architecture, showing the Kodak architecture is much more unevenly distributed mainly due to the ResNet blocks in the decoder.

Computational Complexity. Tables D.1 and D.2 show the number of operations for the MNIST and Kodak architectures, respectively, as numbers of multiply-accumulate (MAC) operations. We find that the Ω quantizer in general requires more operations than the INT quantizer as it is more complex mainly due to the 1×1 convolution. Again, while in Table D.1 it can be seen that the differences are substantial for the playground MNIST architecture (although at only about 1,000 ... 5,000 M operations per image), Table D.2 shows for the practically more relevant Kodak dataset that the Ω quantizer autoencoder (2,202 GMACs) has to be *compared to the INT quantizer autoencoder (1,820 GMACs), which is "only" an about 22% higher computational complexity for the Ω quantizer autoencoder.* For the MNIST architecture, the encoder and decoder require 59.3 and 997.6 MMACs, respectively, while the imbalance results from the larger kernel size in the second decoder layer (see Section D), and for the Kodak architecture, encoder and decoder require 154.6 and 1,649.6 GMACs, respectively, also showing a more complex decoder again due to the ResNet blocks in the decoder.