7 Supplementary Material

In Section 7.1 we provide a description of the meta-learning algorithm and in Section 7.2 we explain in detail how we overfit starting from meta-learned initializations. We motivate our architecture choice in Section 7.3 and show the influence of L_1 regularization in Section 7.4. Additional ablation studies on the influence of quantization bitwidth and the generalization potential of meta-learned initializations are shown in Section 7.6 and Section 7.7 respectively. A further runtime comparison between our method, JPEG and SOTA RDAEs is presented in Section 7.8. In Section 7.9 we present a complete overview of the hyperparameters used in the different stages of the compression pipeline. Finally, we provide additional qualitative examples comparing our method to JPEG and JPEG2000 in Section 7.10 and Draco in Section 7.11.

7.1 Meta-Learning Algorithm

Meta-learned Initializations for Implicit Neural Representations Algorithm 1 is an adapted version of the algorithm presented in [48]. We modify the notation to match ours and change the objective to image regression instead of signed distance function regression. Generally, the algorithm consists of two loops, the outer loop (lines 7-15) and the inner loop (lines 11-13). The outer loop index i is denoted as a superscript, whereas the inner loop index j is denoted as a subscript. The outer loop is executed for a predefined number of iterations n. First of all, we sample an image $\mathbf{x}^i$ from the data distribution. We then define a coordinate vector $\mathbf{p}$ with a coordinate grid of the same resolution as the image $\mathbf{x}^i$. We initialize the inner loop parameters ϕ_0^i to the current outer loop parameters θ. For $i = 1$ this is just random initialization, afterwards these are the meta-learned parameters. We start the inner loop on line 9. For k iterations we compute the MSE loss between the image $\mathbf{x}^i$ and the output of the INR parameterized by the inner loop parameters ϕ_j^i. On line 13 we perform a gradient update of the inner loop parameters. α contains the learning rates for the inner loop gradient update. A simple choice is using the same static learning rate for all parameters in ϕ_j^i. The power of meta-learning is that we can also meta-learn the learning rate of the inner loop, thus α is an optimization variable just like θ. We can take this even a step further by meta-learning a learning rate for every individual inner loop parameter and every step j. This variant is referred to by the authors [48] as a *per parameter per step* learning rate type [5]. We are effectively learning k times as many learning rates as model parameters. We use the Hadamard product on line 13 to denote that the product between the learning rates in α and the gradient is performed componentwise. The subscript j of α denotes that we have different learning rates in each step j.

After k iterations of the inner loop, we recompute the loss for the inner loop parameters of the last step. On line 15 we perform a gradient update of the meta-learned model parameters and the learning rates α. This is a gradient of

[5] https://github.com/vsitzmann/metasdf

Algorithm 1 MetaSiren (modified version of MetaSDF[48] Algorthim 1)

Required Inputs:
  $\mathcal{D}$: dataset for meta-learning
  α_{init}: initial learning rates for inner loop
procedure TRAININITIALIZATION($\mathcal{D}, \alpha_{init}$)
    Randomly initialize θ
    $\alpha \leftarrow \alpha_{init}$
    for $i \in [1, n]$ **do**
      Sample training image $\mathbf{x}^i \sim \mathcal{D}$
      Get coordinates $\mathbf{p} = \text{coord}(\mathbf{x}^i) \in [-1, 1]^{W \times H}$
      Initialize $\phi_0^i \leftarrow \theta$, $\mathcal{L} \leftarrow 0$
      for $j \in [0, k-1]$ **do**
        $\mathcal{L} \leftarrow \text{MSE}(f_{\phi_j^i}(\mathbf{p}), \mathbf{x}^i)$
        $\phi_{j+1}^i \leftarrow \phi_j^i - \alpha_j \odot \nabla_{\phi_j^i} \mathcal{L}$
      $\mathcal{L} \leftarrow \text{MSE}(f_{\phi_k^i}(\mathbf{p}), \mathbf{x}^i)$
      $\theta, \alpha \leftarrow (\theta, \alpha) - \beta \nabla_{(\theta,\alpha)} \mathcal{L}$
    return θ, α

Algorithm 2 Overfit INR starting from a meta-learned initialization

Required Inputs:
  $\mathbf{x}$: the image to overfit
  $\mathbf{p}$: coordinate grid at desired resolution
  θ_0: meta-learned initialization
  α: meta-learned learning rates
procedure OVERFITMETA($\mathbf{x}, \mathbf{p}, \theta_0, \alpha$)
    for $j \in [0, k-1]$ **do**
      $\theta_{j+1} \leftarrow \theta_j - \alpha_j \odot \nabla_{\theta_j} \text{MSE}(f_{\theta_j}(\mathbf{p}), \mathbf{x})$
    $\theta \leftarrow \theta_k$
    $\theta^\star \leftarrow \arg\min_\theta \mathcal{L}(\mathbf{x}, f_\theta(\mathbf{p}))$
    return $\theta^\star$

gradients: backpropagation is applied to the computation graph of the inner loop that itself contains the inner loop gradient updates. We continue on line 8 by sampling the next image and repeat the process. After all outer loops have finished, we return the meta-learned weights θ and learning rates α.

7.2 Overfitting from Meta-Learned Initializations

At the beginning of the overfitting phase, we make use of parameter-wise learning rates obtained from meta-learning. Basically, we just run the inner loop once which gets us already close to the final image in just $k = 3$ steps as shown in Algorithm 2. We then continue optimizing with Adam. The momentum terms of the Adam optimizer are uninitialized at this point. We have found that linearly increasing the learning rate in a warmup phase of 100 epochs prevents an initial

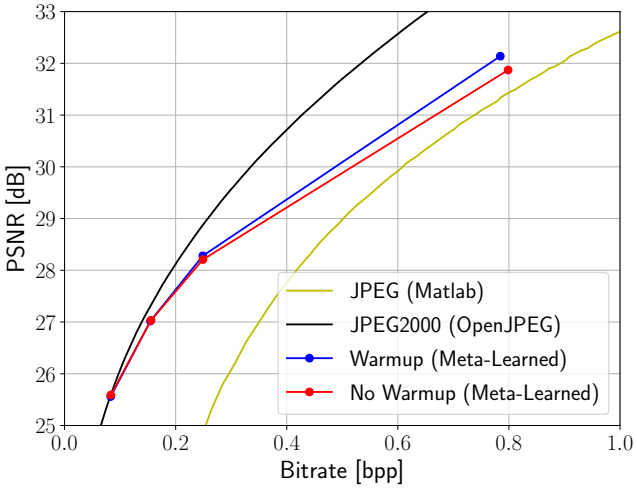


Fig. 10: Comparing the meta-learned approach evaluated on Kodak with and without a warmup phase in the beginning of the training. We achieve better performance at higher bitrates when using a warmup phase.

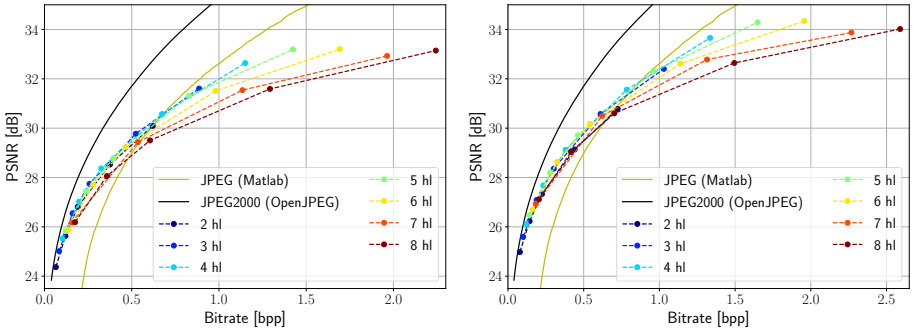
degradation of reconstruction quality and even improves final performance at higher bitrates (see Fig. 10).

7.3 Number of Layers and Hidden Dimension

Important architecture choices when using MLP based networks, are the number of hidden layers and the number of hidden units. Given an MLP, *depth* or *width* both directly influence the number of parameters and indirectly impact bitrate. In other words, there are two ways of scaling up the network. We examine the rate-distortion performance for various combinations of hidden units ($M \in \{32, 48, 64, 96, 128\}$) and hidden layers ($\{2, \dots, 8\}$) in Fig. 11 using our basic approach and $\lambda = 10^{-6}$. We can see from both plots that increasing the number of layers eventually leads to diminishing returns: The bitrate keeps increasing while the gain in PSNR is small. The flattening for higher numbers of hidden layers is even more pronounced at the lower bitwidth $b = 7$. The quantization noise is stronger here and with increasing depth the noise might get amplified and limit the performance. We conclude that rate-distortion performance scales more gracefully with respect to the width of the model. We do however notice as well that the lowest setting of 2 hidden layers is typically outperformed by a network with fewer hidden units and more layers.

7.4 Impact of L_1 Regularization.

In this experiment we try to verify whether L_1 regularization has a beneficial effect on performance. We train with the default parameters starting from ran-



(a) Bitwidth $b = 7$ (b) Bitwidth $b = 8$

Fig. 11: Comparing compression performance of models with the number of hidden layers (hl) varying between 2-8 and quantization bitwidths of $b = 7$ or $b = 8$ bits.

dom initializations and vary λ within $[0, 10^{-4}]$.
In Fig. 12 we observe that a value of $\lambda = 10^{-5}$ has better performance at higher bitrates than lower choices for λ. The performance improvement shows as a reduction in bitrate which supports the claim that the L_1 regularization can lead to a reduction in entropy. Increasing, the regularization strength to $\lambda = 10^{-4}$ restricts the weights too much, resulting in worse performance than $\lambda = 10^{-5}$. Thus, L_1 regularization can help to reduce entropy, but needs to be combined with a modification in architecture size to achieve a good rate-distortion tradeoff.

7.5 Post-Quantization Optimization.

We compare our meta-learned approach for different post-quantization optimization settings. Fig. 13 shows the performance difference evaluated on Kodak. We see that AdaRound and retraining applied on their own lead to a consistent improvement. The best choice throughout the bitrate range is however to apply the methods in conjunction.

7.6 Influence of Quantization Bitwidth

We show the influence of bitwidth on the rate-distortion performance in Fig. 14 for the meta-learned approach and in Fig. 15 for the basic approach. For the meta-learned approach 7-bit is the best choice for both datasets. For the basic approach however, 8-bit quantization outperforms lower bitwidths. On the Kodak dataset the difference between 7- and 8-bit quantization is quite small nevertheless. We also show the unquantized performance of the 4 MLPs with variying number of hidden units as dashed horizontal lines. We see that for a bitwidth of 8 we can almost reach unquantized performance for the majority of configurations.

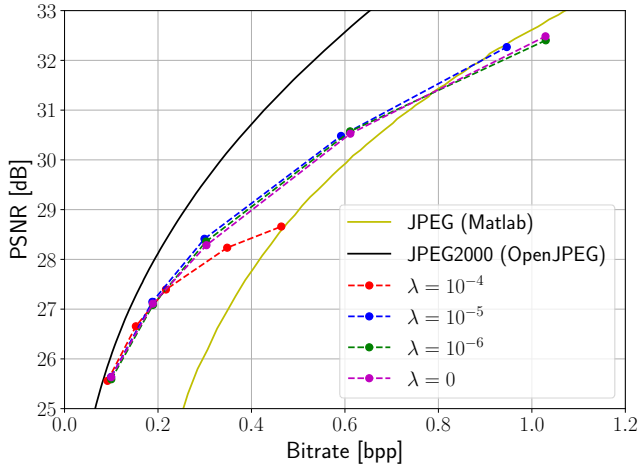


Fig. 12: Rate-distortion performance for different $L1$ regularization parameters λ evaluated on the Kodak dataset.

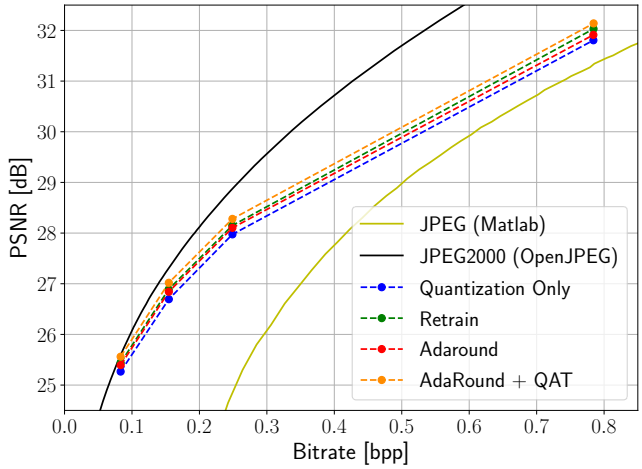
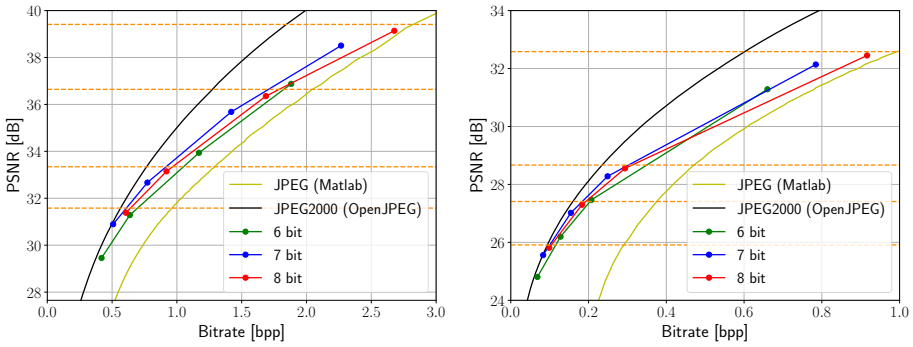


Fig. 13: Comparison of QAT, AdaRound and the combination of both to basic quantization on the Kodak dataset.

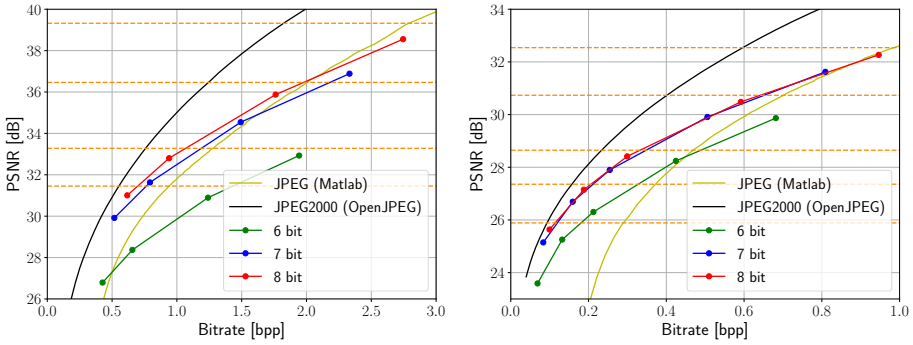
7.7 Generalization of Meta-Learned Initializations

We want to show that the meta-learned initializations are able to generalize to out-of-distribution images, even if the meta-learning dataset contains only similar images. To this end, we minimally crop and resize Kodak images to the same resolution and aspect ratio as CelebA (178×218) and then compress them using meta-learned initializations obtained from CelebA. In Fig. 16 we show that the meta-learned approach still outperforms the basic approach.



(a) CelebA (b) Kodak

Fig. 14: Comparision of different quantization bitwidths for the *meta-learned* approach. The PSNR achieved by the unquantized models is shown by the dashed horizontal lines. Note that these are not rate-distortion curves and are only supposed to show the distortion introduced by quantization.



(a) CelebA (b) Kodak

Fig. 15: Comparision of different quantization bitwidths for the *basic* approach. The PSNR achieved by the unquantized models is shown by the dashed horizontal lines. Note that these are not rate-distortion curves and are only supposed to show the distortion introduced by quantization.

7.8 Further Runtime Comparison

In this paper we typically train the models until full convergence, hence we optimize for the best rate-distortion performance. To show that our method can be tuned for fast runtime, we compare the encoding/decoding runtime (in [s]) with JPEG and Xie et al [58] in Tab. 1, under the constraint that the rate-distortion performance at least matches JPEG. Thus, we trade-off performance with runtime.

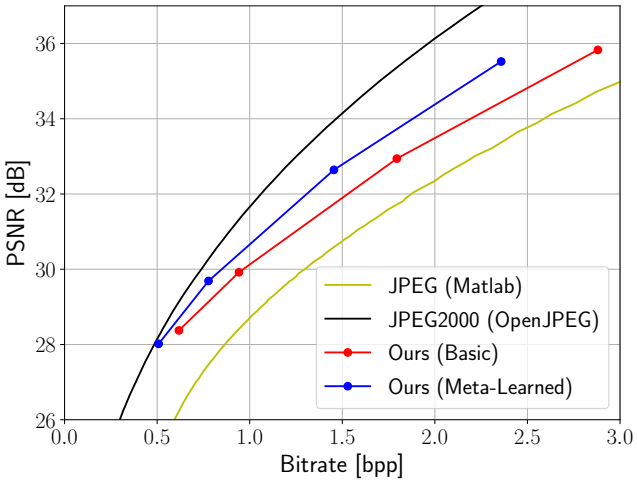


Fig. 16: Generalization experiment: Using meta-learned initializations trained on CelebA evaluated on (cropped and resized) Kodak images. The meta-learned initializations provide enough generalization capability to improve compression performance also on out-of-distribution images.

7.9 Detailed Training Hyperparameter Overview

In this section we provide a complete overview of all hyperparameters. We also mention details specific to the training procedure in the respective subsection.

Image Compression Hyperparameters

Architecture. We summarize the default architecture hyperparameters in Table 2. We typically evaluate our method for several choices of the hidden dimension M. For Kodak and CelebA images at full resolution we use $M \in \{32, 48, 64, 128\}$ and $M \in \{24, 32, 48, 64\}$ respectively. For Kodak images at half resolution (2x scale) we use $M \in \{8, 16, 32, 48\}$ and also reduce the number of input frequencies to $L = 12$. For Kodak images at quarter resolution (4x scale) we use $M \in \{4, 8, 16, 32\}$ and further reduce the number of input frequencies to $L = 10$.

Meta-Learning the Initializations. In Table 3 we list the default values of the hyperparameters for meta-learning the initializations. We use a learning rate schedule that halves the learning rate when no improvement has been made in the last (10 = patience) validations. For validation we use a subset of 100 images sampled from the validation set of the respective dataset, in our case CelebA or DIV2K. We compute the validation loss by running the inner loop optimization for each image in the validation subset. The fact that computing the validation loss involves inner loop training is the reason why we limit the

	JPEG	Xie [58]	Ours	Ours*
Encoding	0.0061	2.22	4.49	0.542
Decoding	0.0023	5.69	0.664	0.722

Table 1: Comparison of average encoding and decoding speeds in [s] on Kodak. For INRs we show the time at 0.175 bpp where JPEG is outperformed in terms of PSNR in all cases. All methods use the same hardware. INR encoding runs on GPU. Everything else, including decoding with INRs runs on CPU. Xie [58] only supports CPU inference for encoding and decoding according to the authors. Note that half of our decoding time is entropy coding. Ours* (w. meta-learning) reaches the performance of JPEG approximately 10x faster than Ours (no meta-learning). JPEG is clearly the fastest method, but we can outperform Xie [58], given that the encoding device has a GPU. We deliberately show decoding time for CPU because GPUs are less common on the end-user device and it makes the timings more comparable to Xie [58] and JPEG.

Architecture Hyperparameters	
Description	Value
Hidden layers N	3
Activation function	$\sin(30x)$
Input encoding	Positional with $\sigma = 1.4$
Input frequencies L	16 (Kodak), 12 (CelebA)

Table 2: Default architecture hyperparameters.

Meta-Learning Hyperparameters	
Description	Value
Outer loop initial learning rate β	$5 \cdot 10^{-5}$
Outer loop batch size	1
Outer loop optimizer	Adam [32]
Epochs	30 (DIV2K), 1 (CelebA)
Inner loop initial learning rate α_{init}	10^{-5}
Learning rate type	*per parameter per step*
Inner loop steps k	3
Steps until validation	500
LR schedule patience	10
LR schedule factor	0.5

Table 3: Default hyperparameters for learning the initializations.

validation set size to 100. When we train the initializations on CelebA we train for 1 epoch. When using the DIV2K dataset we train for 30 epochs because it is a significantly smaller dataset. We finally, save the initialization that achieved the lowest validation loss overall.

Overfitting. The default hyperparameters for the overfitting stage are shown in Table 4. If not mentioned otherwise we use these hyperparameters for all

Overfitting Hyperparameters	
Description	Value
Initial learning rate	$5 \cdot 10^{-4}$
Optimizer	Adam [32]
Epochs	25000
L_1 regularization λ	10^{-5}
Steps until validation	1
LR schedule patience	500
LR schedule factor	0.5
Early stopping epochs	5000

Table 4: Default hyperparameters for overfitting the INR.

Quantization Hyperparameters	
Description	Value
Bitwidth	7 (Meta-Learned), 8 (Basic)
Retraining epochs	300
Optimizer	Adam [32]
Retraining learning rate	10^{-6}
AdaRound iterations	1000
AdaRound regularizer	10^{-4}
Bitstream coding	arithmetic coding

Table 5: Default hyperparameters for quantization, post-quantization optimization and bitstream coding.

experiments. In particular, we do not use L_1 regularization for experiments at reduced resolution. Since we are overfitting, we validate on the training image itself. We reduce the learning rate by a factor of 0.5 if the loss has not improved during the last 500 epochs. Note that 1 epoch equals 1 optimizer step, in other words, one training batch contains all pixels of the image we overfit. We train for 25000 epochs to make sure every architecture and configuration we test has the chance to reach convergence. As to be expected, smaller models typically reach peak performance faster. We stop training if the loss has not improved in the last 5000 epochs to prevent unnecessary computation resource use. In the end, we return the parameters of the model that has achieved the lowest loss during overfitting.

Quantization, Post-Quantization Optimization & Entropy Coding. In Table 5 we show the default values for the quantization and bitstream coding related stages. We emphasize that per default we use the combination of AdaRound and QAT.

3D Shape Compression Hyperparameters We use a very similar training procedure for the 3D shape compression as for image compression. For images we simply use the pixels of one image as the batch used for overfitting. For 3D

however, we need to choose a number of 3D point samples, in our case 100000, for which we first compute the ground truth distance to the surface and then use them to fit our model. We use a subset of 10000 points as a batch for training. Overall, we train for 500 epochs with an initial learning rate of $5 \cdot 10^{-5}$ and the same learning rate schedule as for images. For 3D shape compression we do not use L_1 regularization. We use the same architectures as for Kodak, namely $M \in \{32, 48, 64, 128\}$ and $L = 16$ input frequencies. The overfitted models are quantized to 8 bit and we then optimize the weights using 2000 iterations of AdaRound and 50 epochs of retraining with a learning rate of 10^{-7}.
For the Draco baseline we call the encoder with a certain bitwith to quantize the mesh, in particular, the "qp" flag. We vary the bithwidth within $[5, \dots, 12]$. We use the highest compression quality setting "cl" of 10. To make sure we only encode a raw mesh, we set the skip flag for TEXTURE, NORMAL and GENERIC.

7.10 Additional Image Examples

We show additional compression examples to compare our method to the codecs JPEG and JPEG2000. We first show more images at the lowest bitrate, *i.e.*, using the lowest hidden dimension, where our method is most competitive to JPEG2000: In Fig. 17 we evaluate on KODAK using a model with $M = 32$ and in Fig. 18 we evaluate on CelebA using a model with $M = 24$. We visually confirm a clear advantage over JPEG for all images and similar performance as JPEG2000 at this bitrate. Moreover, we show examples at higher bitrates, *i.e.*, using larger hidden dimensions, in Fig. 19 (Kodak) and Fig. 20 (CelebA). While our method maintains an advantage over JPEG, JPEG2000 outperforms our approach with an increasing advantage towards higher bitrates, where the difference is most apparent in the rendering of fine details.

7.11 Visualization of compressed 3D shapes

We demonstrate the effectiveness of INRs for 3D shape compression by visual comparison in Fig. 21. The SDFs learned by the INRs in general render a much smoother surface than the mesh compression algorithm Draco, while being very storage efficient. Draco introduces significant surface noise making the compressed shape very rough. The reduction of information for INRs is more faithful in that the encoded shape is a simplified version of the original: The smaller model with $M = 64$, that requires only roughly a quarter of the storage of the model with $M = 128$, still looks very much like the original with certain details smoothed out. This makes it visually much more pleasing than the rough looking Draco compression, where the details are lost through surface noise.

Fig. 17: Performance comparison on Kodak using a hidden dimension of $M = 32$ for all models.

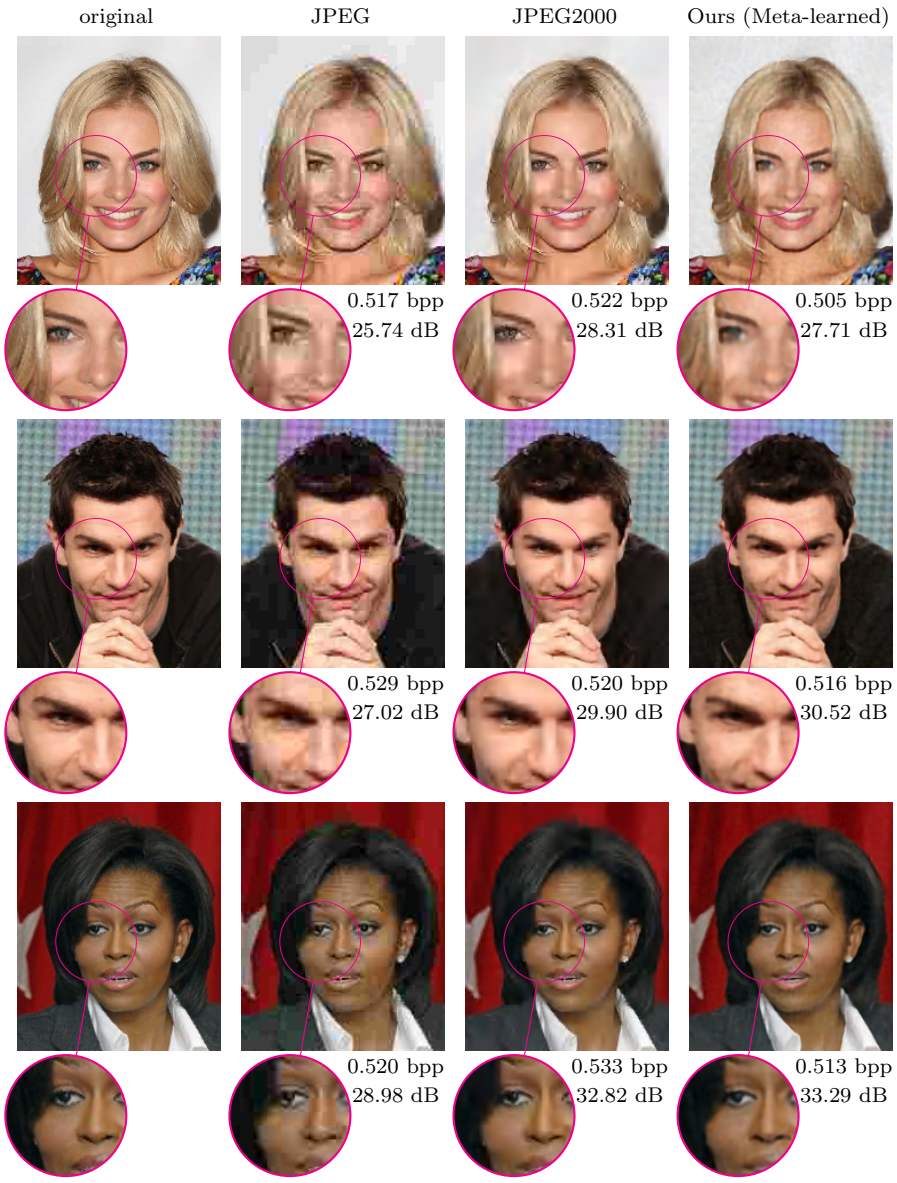


Fig. 18: Performance comparison on CelebA using a hidden dimension of $M = 24$ for all models.

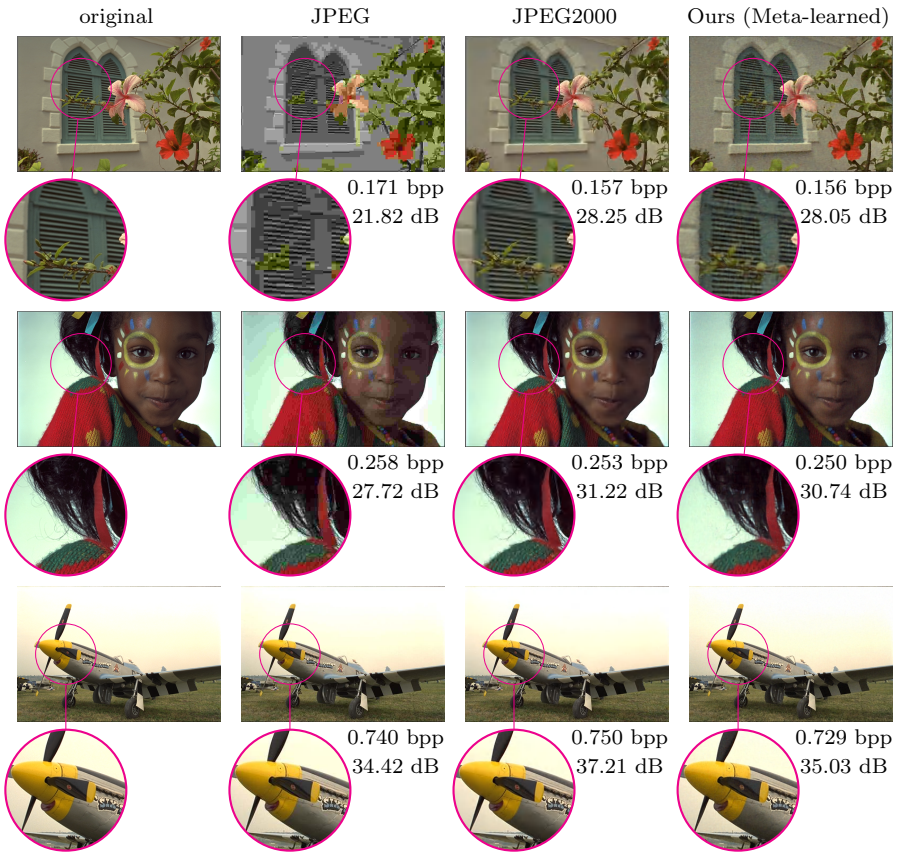


Fig. 19: Performance comparison on Kodak using a hidden dimensions of $M = 48$ (top), $M = 64$ (middle), $M = 128$ (bottom).

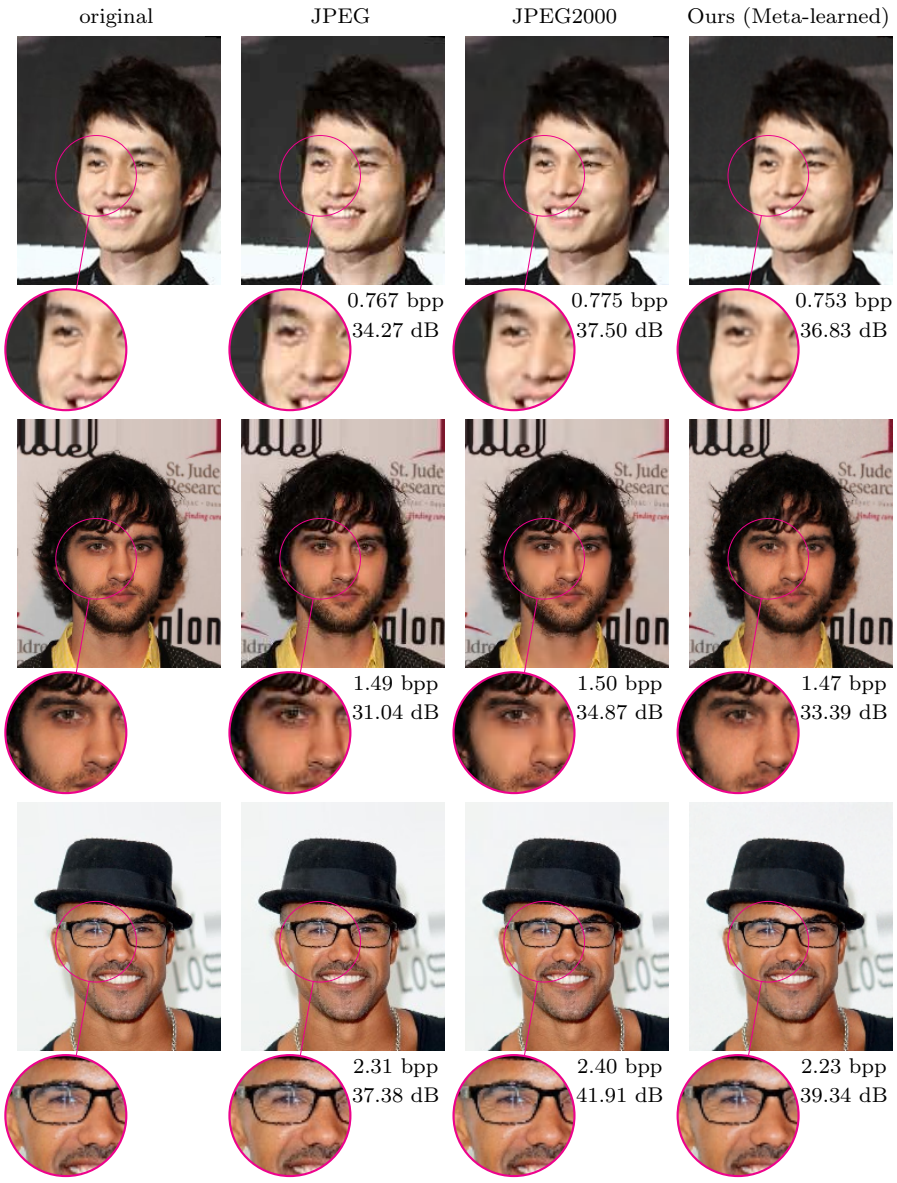


Fig. 20: Performance comparison on CelebA using a hidden dimensions of $M = 32$ (top), $M = 48$ (middle), $M = 64$ (bottom).

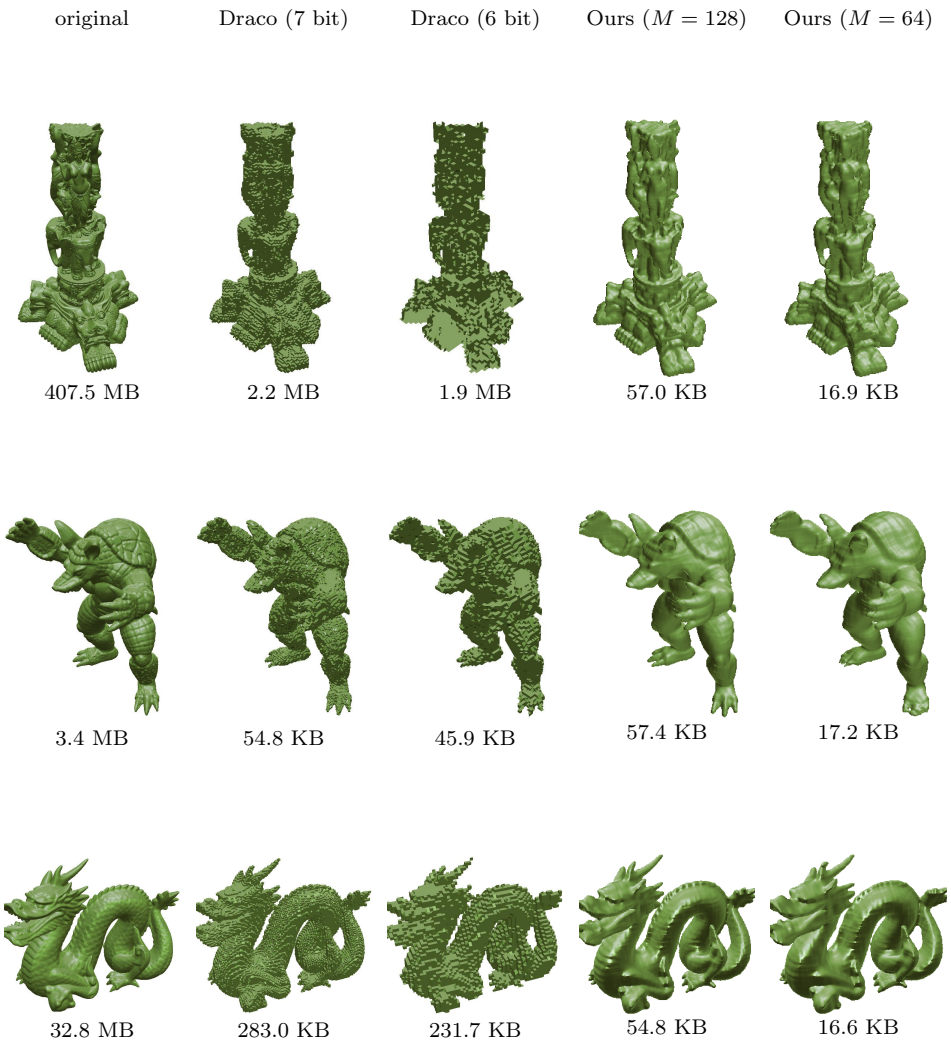


Fig. 21: Visual comparsion of the mesh compression algorithm Draco compared to our method applied to 3D shape compression. We compare 2 quantization setting for Draco, namely 6 and 7 bit, and two hidden dimensions $M = 64, 128$ using our method. Our method shows a significantly smoother surface reconstruction and better detail at similar or lower storage.