OVERFITTING FOR FUN AND PROFIT: INSTANCE-ADAPTIVE DATA COMPRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural image and video compression has been shown to outperform classical methods in terms of rate-distortion (RD) performance, with results still improving rapidly. At a high level, these methods are based on an autoencoder that tries to reconstruct the input instance from a (quantized) latent representation, coupled with a prior that is used to losslessly compress these latents. Due to limitations on model capacity and imperfect optimization and generalization, such models will not in general perform optimal compression on test data. However, one of the great strengths of learned compression is that if the test-time data distribution is known and relatively low-entropy (e.g. a camera watching a static scene, a dash cam in an autonomous car, etc.), the model can easily be finetuned or *adapted* to this distribution, leading to improved RD performance. In this paper we take this concept to the extreme, adapting the *full* model to a *single* video, and sending model updates (quantized and compressed using a parameter-space prior) along with the latent representation. Unlike previous work, we finetune not only the encoder/latents but the entire model, and take into account both the effect of model quantization and the additional costs incurred by sending the model updates during finetuning. We evaluate an I-frame model on the UVG video dataset, and demonstrate that our method is able to improve RD performance for every video tested, sometimes by a large margin.

1 INTRODUCTION

The most common approach to neural lossy compression is to train a variational autoencoder (VAE)-like model on a training dataset to minimize the *expected* RD cost $D + \beta R$ (Theis et al., 2017; Kingma & Welling, 2013). Although this approach has proven to be very successful (Ballé et al., 2018), a model trained to minimize expected RD cost over a dataset is unlikely to be optimal for every test instance, because the model has limited capacity, and both optimization and generalization will be imperfect. The problem of generalization will be especially significant when the testing distribution is different from the training distribution, as is likely to be the case in practice.

Suboptimality of the encoder has been studied extensively under the term *inference suboptimality* (Cremer et al., 2018), and it has been shown that finetuning the encoder or latents for a particular instance can lead to improved compression performance (Lu et al., 2020; Campos et al., 2019; Yang et al., 2020; Guo et al., 2020). This approach is appealing as no additional information needs to be added to the bitstream, and nothing changes on the receiver side. Performance gains however are limited, because the prior and decoder can not be adapted.

In this paper we present a method for *full-model instance-adaptive compression*, i.e. adapting the *entire* model to a *single* data instance. Unlike previous work, our method takes into account the costs for sending not only the latent prior, but also the decoder model updates, as well as quantization of these updates. This is achieved by extending the typical RD loss with an additional model-rate term M that measures the number of bits required to send the model updates under a newly introduced *model prior*, resulting in a combined RDM loss.

As an initial proof of concept, we show that this approach can lead to very substantial gains in RD performance (~ 1 dB PSNR gain at the same bit rate in the low bit rate region) on the problem of all-intra video coding, where a video is coded frame-by-frame using an I-frame (image compression) model. Additionally, we show how the model rate bits are distributed across the model.

The rest of this paper is structured as follows. In Section 2 we discuss the basics of neural compression and related work on adaptive compression. Section 3 presents our method, including details on the RDM loss, the choice of the model prior, network quantization, and the encoding/decoding procedure. In Sections 4 and 5 we present our experiments and results, followed by a discussion in Section 6.

2 Preliminaries and Related Work

2.1 Neural data compression

The standard approach to neural compression can be understood as a particular kind of VAE (Kingma & Welling, 2013). In the compression literature the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is typically defined by a neural network with either deterministic output (so $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is one-hot) (Habibian et al., 2019) or with fixed uniform $[0, 1]$ noise on the outputs (Ballé et al., 2018). In both cases, sampling $z \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is used during training while quantization is used at test time.

The latent $\boldsymbol{z}$ is encoded to the bitstream using arithmetic coding in conjunction with a latent prior $p_\theta(\boldsymbol{z})$, so that coding $\boldsymbol{z}$ takes about $-\log p_\theta(\boldsymbol{z})$ bits (up to discretization). On the receiving side, the arithmetic decoder is used with the same prior $p_\theta(\boldsymbol{z})$ to decode $\boldsymbol{z}$ and then reconstruct $\hat{\boldsymbol{x}}$ using the decoder network $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ (note that we use the same symbol θ to denote the parameters of the prior and decoder jointly, as in our method both will have to be coded to the bitstream).

From these considerations it is clear that the rate R and distortion D can be measured by the two terms in the following loss:

$$\mathcal{L}_{\text{RD}}(\phi, \theta) = \beta \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[-\log p_\theta(\boldsymbol{z})\right]}_{R} + \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[-\log p_\theta(\boldsymbol{x}|\boldsymbol{z})\right]}_{D}. \tag{1}$$

As it turns out, this loss is equal (up to the tradeoff parameter β and an additive constant) to the standard evidence lower bound (ELBO) used in VAE training. In general the rate term of the ELBO is written as a KL divergence between encoder and prior, but since $D_{\text{KL}}(q, p) = R - H[q]$ and the encoder entropy $H[q]$ is constant in our case, the KL loss is equivalent to the rate loss.

2.2 Adaptive Compression

A compression model is trained on a dataset $\mathcal{D}$ with the aim of achieving optimal RD performance on test data. However, because of limited model capacity, optimization difficulties or insufficient data (resulting in poor generalization), the model will in general not achieve this goal. When the test data distribution differs from that of the training data, generalization will not be guaranteed even in the limit of infinite data and model capacity, and perfect optimization.

A convenient feature of neural compression however is that a model can easily be finetuned on new data or data from a specific domain. A model can for instance (further) be trained after deployment, and Habibian et al. (2019) showed improved RD gains after finetuning a video compression model on footage from a dash cam, an approach dubbed *adaptive compression* (Habibian et al., 2019).

In adaptive compression, decoding requires access to the adapted decoder and prior models. These models (or their delta/update relative to a pretrained shared model) thus need to be added to the bitstream. When the amount of data coded with the adapted model is large, the cost of signaling the model update will be negligible as it is amortized. However, a trade-off exists, as the smaller the domain of adaptation, the more we can expect to gain from adaptation (e.g. an image compared to a video or collection of videos). In this paper we consider the extreme case where the domain of adaptation is a single instance, resulting in costs for sending model updates which become very relevant.

2.3 Closing the amortization gap

Coding model updates can easily become prohibitively expensive when the model is adapted for every instance. However, if we only adapt the encoder or latents, no model update needs to be added to the bitstream as the encoder is not needed for decoding as the latents are sent anyway. We can

thus close, or at least reduce, the *amortization gap* (the difference between $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and the optimal encoder; Cremer et al. (2018)) without paying any bits for model updates. Various authors have investigated this approach: Aytekin et al. (2018); Lu et al. (2020) adapt the encoder, while Campos et al. (2019); Yang et al. (2020); Guo et al. (2020) adapt the latents directly. This simple approach was shown to provide a modest boost in RD performance.

2.4 Encoding model updates

As mentioned, when adapting (parts of) the decoder or prior to an instance, model updates have to be added to the bitstream in order to enable decoding. Recent works have proposed ways to finetune parts of the model, while keeping the bitrate overhead small. For instance Klopp et al. (2020) train a reconstruction error predicting network at encoding time, quantize its parameters, and add them to the bitstream. Similarly (Lam et al., 2019; 2020) propose to finetune all parameters or only the convolutional biases, respectively, of an artifact removal filter that operates after decoding. A sparsity-enforcing and magnitude-suppressing penalty is leveraged, and additional thresholding is applied to even more strictly enforce sparsity. The update vector is thereafter quantized using k-means clustering. Finally, (Zou et al., 2020) finetunes the latents in a meta-learning framework, in addition to updating the decoder convolutional biasses, which are quantized by k-means and thereafter transmitted.

All these methods however perform quantization post-training (leading to a potentially unbounded reduction in performance). Also, albeit the use of regularizing loss terms, no valid proxy for the actual cost of sending model updates is adopted. Finally, none of these methods performs adaptation of the full model.

3 Full-Model Instance-Adaptive Compression

In this section we present our framework, in which we finetune an entire compression model for one instance, while taking into account both model quantization and the true costs for sending model updates during training. The main idea is described in Section 3.1, after which quantization-specific details are given in Section 3.2 and the algorithm is described in Section 3.3.

3.1 Finetuning at inference time

Full-model instance-adaptive compression starts with a set of *global model* parameters $\theta_\mathcal{D}$ (obtained by training on dataset $\mathcal{D}$) which are finetuned along with encoder parameters $\phi_\mathcal{D}$ to fit a single instance $\boldsymbol{x}$. This results in new parameters θ, ϕ, of which only θ has to be signalled in the bitstream. In practice we learn and encode only the *model updates* $\delta = \theta - \theta_\mathcal{D}$. In order to encode δ, we introduce a continuous model prior $p(\delta)$ to regularize these updates, and adopt the quantized counterpart $p[\bar{\delta}]$ for entropy (de)coding them (more on quantization in Section 3.2).

The overhead for sending quantized model update $\bar{\delta}$ is given by model rate $\overline{M} = -\log p[\bar{\delta}]$, and can be approximated by its continuous counterpart $M = -\log p(\delta)$ (see Appendix A for justification).

Adding this term to the standard RD loss using the same RD tradeoff parameter β, we obtain the instance-adaptive compression objective:

$$\mathcal{L}_{\text{RDM}}(\phi, \delta) = \mathcal{L}_{\text{RD}}(\phi, \theta_\mathcal{D} + \bar{\delta}) - \beta \underbrace{\log p(\delta)}_{M}. \tag{2}$$

At inference time, this objective can be minimized directly to find the optimal model parameters for transmitting datapoint $\boldsymbol{x}$. It takes into account the additional costs for encoding model updates (M term), and incorporates model quantization during finetuning ($\mathcal{L}_{\text{RD}}$ evaluated at $\bar{\theta} = \theta_\mathcal{D} + \bar{\delta}$).

For simplicity we model parameter update $\delta = \theta - \theta_\mathcal{D}$ as a zero-centered Gaussian with zero covariance and a single shared (hyperparameter) σ, denoting the standard deviation:

$$p(\delta) = \mathcal{N}(\delta \,|\, \mathbf{0}, \sigma \boldsymbol{I}). \tag{3}$$

Note that this is equivalent to modeling θ by $p(\theta) = \mathcal{N}(\theta \,|\, \theta_\mathcal{D}, \sigma \boldsymbol{I})$.

When entropy (de)coding $\bar{\delta}$ under $p[\bar{\delta}]$ we must realize that even the zero-update, i.e. $\delta = \mathbf{0}$, is not for free. We define these initial static costs as $\overline{M}_0 = -\log p[\bar{\delta} = \mathbf{0}]$. Because the mode of our prior is zero, these initial costs $\overline{M}_0$ equal the minimum costs. Minimization of eq. (2) thus ensures that – after overcoming these static costs – any extra bit spent on model updates will be accompanied by a commensurate improvement in RD performance.

3.2 QUANTIZATION

In order to quantize a scalar δ_i (denoted by δ in this section to avoid clutter), we use N equispaced bins of width t, defined by the quantization function:

$$\bar{\delta} = Q_t(\delta) = \text{clip}\left(\left\lfloor \frac{\delta}{t} \right\rceil \cdot t, \min = -\frac{(N-1)t}{2}, \max = \frac{(N-1)t}{2}\right). \tag{4}$$

As rounding is non-differentiable, the gradient of Q_t is approximated using the Straight-Through estimator (STE), proposed by Bengio et al. (2013). That is, we assume $\partial Q_t(\delta)/\partial\delta = 1$.

The bins are intervals $Q_t^{-1}(\bar{\delta}) = [\bar{\delta}-t/2, \bar{\delta}+t/2]$. We view t as a hyperparameter and define N to be the smallest integer such that the region covered by bins (i.e. the interval $[-(N-1)t/2, (N-1)t/2]$), covers at least $1 - 2^{-8}$ of the probability mass of $p(\delta)$. Indeed the number of bins N is proportional to the ratio of the width of $p(\delta)$ to the width of the bins: $N \propto \sigma/t$. The number of bins presents a tradeoff between finetuning flexibility and model rate costs $\overline{M}$, so the σ/t-ratio is an important hyperparameter. The higher N, the higher these costs due to finer quantization, but simultaneously the lower the quantization gap, enabling more flexible finetuning.

Since $\bar{\delta} = Q_t(\delta)$, the discrete model prior $p[\bar{\delta}]$ is the pushforward of $p(\delta)$ through Q_t:

$$p[\bar{\delta}] = \int_{Q^{-1}(\bar{\delta})} p(\delta)d\delta = \int_{\bar{\delta}-t/2}^{\bar{\delta}+t/2} p(\delta)d\delta = P(\delta < \bar{\delta} + t/2) - P(\delta < \bar{\delta} - t/2). \tag{5}$$

That is, $p[\bar{\delta}]$ equals the mass of $p(\delta)$ in the bin of $\bar{\delta}$, which can be computed as the difference of the CDF of $p(\delta)$ evaluated at the edges of that bin.

3.3 ENTROPY CODING AND DECODING

After finetuning of the compression model on instance $\boldsymbol{x}$ by minimizing the loss in eq. (2), both the latents and the model updates are entropy coded (under their respective priors $p_{\bar{\theta}}(\boldsymbol{z})$ and $p[\bar{\delta}]$) into a bitstream $\boldsymbol{b}$. Decoding starts by decoding $\bar{\delta}$ using $p[\bar{\delta}]$, followed by decoding $\boldsymbol{z}$ using $p_{\bar{\theta}}(\boldsymbol{z})$ (where $\bar{\theta} = \bar{\delta} + \theta_{\mathcal{D}}$), and finally reconstructing $\hat{x}$ using $p_{\bar{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. The whole process is shown in Figure 1 and defined formally in Algorithms 1 and 2.

4 EXPERIMENTAL SETUP

4.1 DATASETS

The experiments in this paper use images and videos from the following two datasets:

CLIC19 [1] The CLIC19 dataset contains a collection of natural high resolution images. It is conventionally divided into a professional set, and a set of mobile phone images. Here we merge the existing training folds of both sets and use the resulting dataset $\mathcal{D}$ to train our I-frame (i.e. image) model. The corresponding validation folds are used to validate the global model performance.

UVG [2] The Ultra Video Group dataset (Mercat et al., 2020) consists of RAW 1080p and 4k videos. This dataset has been extended after its first publication. We use the videos of the original publication, which are listed in Appendix D. The 1k 8bit versions are converted to RBG png using ffmpeg, and the original frame size is center-cropped to facilitate down-sampling by a factor 64, required by the adopted model architecture (described in Appendix B). The resulting videos are of size $(W \times H \times T) = 1920 \times 1024 \times 600$ pixels[3]. We use this dataset to validate our instance-adaptive data compression framework.

[1] `https://www.compression.cc/2019/challenge/`

[2] `http://ultravideo.cs.tut.fi/`

[3] The video `ShakeNDry 1920x1080 8bit YUV Y4M` only contains 300 frames.

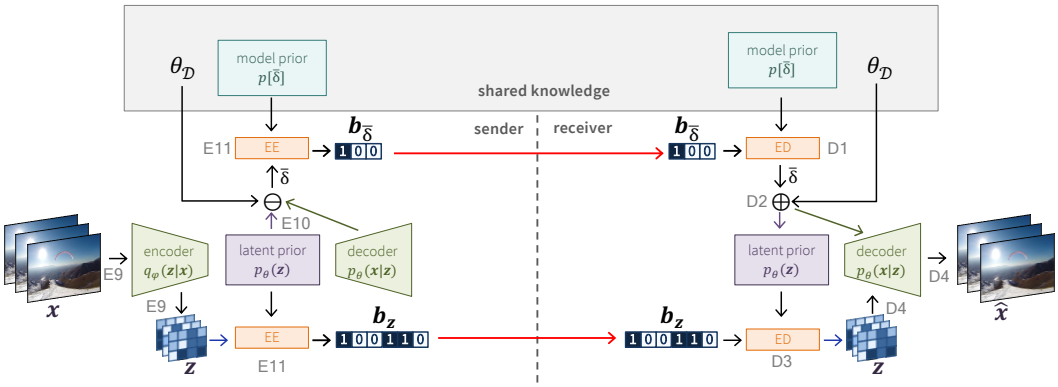


Figure 1: Visualization of encoding (Algorithm 1) and decoding (Algorithm 2) of our full-model instance-adaptive method. Each step is denoted with a code, e.g. *E9*, which refers to line 9 of the encoding algorithm. EE and ED denote entropy encoding and decoding, respectively. Both the compact latent space $\boldsymbol{z}$ and the parameter updates in $\bar{\delta}$ are encoded in their respective bitstreams $\boldsymbol{b}_{\boldsymbol{z}}$ and $\boldsymbol{b}_{\bar{\delta}}$. Model prior $p[\bar{\delta}]$ entropy decodes $\boldsymbol{b}_{\bar{\delta}}$, after which the latent prior can decode $\boldsymbol{b}_{\boldsymbol{z}}$.

Algorithm 1 Encoding of $\boldsymbol{x}$

Input: Model parameters $\{\phi_{\mathcal{D}}, \theta_{\mathcal{D}}\}$ pretrained on training set $\mathcal{D}$, model parameter quantizer Q_t, model prior $p[\bar{\delta}]$, and datapoint to be compressed $\boldsymbol{x}$.
Output: Compressed bitstream $\boldsymbol{b}$.

1: Initialize model parameters: $\phi = \phi_{\mathcal{D}}$, and $\theta = \theta_{\mathcal{D}}$
2: **for** step in MAX STEPS **do**
3: Sample one frame: $\boldsymbol{x}_f \sim \boldsymbol{x}$
4: Quantize transmittable parameters: $\bar{\theta} \leftarrow Q_t[\delta] + \theta_{\mathcal{D}}$, with $\delta = \theta - \theta_{\mathcal{D}}$
5: Forward pass: $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \,|\, \boldsymbol{x}_f)$ and evaluate $p_{\bar{\theta}}(\boldsymbol{x}_f \,|\, \boldsymbol{z})$ and $p_{\bar{\theta}}(\boldsymbol{z})$.
6: Compute loss $\mathcal{L}_{RDM}(\phi, \delta)$ on $\boldsymbol{x}_f$ according to eq. (2)
7: Backpropagate using straight-through gradient estimator for Q_t, then update ϕ, θ using gradients $\dfrac{\partial \mathcal{L}_{RDM}}{\partial \phi}$ and $\dfrac{\partial \mathcal{L}_{RDM}}{\partial \theta}$
8: **end for**
9: Compress $\boldsymbol{x}$ to $\boldsymbol{z} \sim q_\phi(\boldsymbol{x})$
10: Compute quantized model parameters: $\bar{\theta} = \theta_{\mathcal{D}} + \bar{\delta}$, with $\bar{\delta} = Q_t[\theta - \theta_{\mathcal{D}}]$
11: Entropy encode: $\boldsymbol{b}_{\bar{\delta}} = \mathrm{enc}(\bar{\delta}; p[\bar{\delta}])$ and $\boldsymbol{b}_{\boldsymbol{z}} = \mathrm{enc}(\boldsymbol{z}; p_{\bar{\theta}}(\boldsymbol{z}))$
12: Concatenate bitstreams: $\boldsymbol{b} = (\boldsymbol{b}_{\bar{\delta}}, \boldsymbol{b}_{\boldsymbol{z}})$

Algorithm 2 Decoding of $\boldsymbol{x}$

Input: Model parameter prior $p[\bar{\delta}]$, bitstream $\boldsymbol{b} = (\boldsymbol{b}_{\bar{\delta}}, \boldsymbol{b}_{\boldsymbol{z}})$, model parameters $\theta_{\mathcal{D}}$ pretrained on training set $\mathcal{D}$.
Output: Decoded datapoint $\hat{\boldsymbol{x}}$

1: Entropy decode: $\bar{\delta} = \mathrm{dec}(\boldsymbol{b}_{\bar{\delta}}; p[\bar{\delta}])$
2: Compute updated parameters: $\bar{\theta} = \theta_{\mathcal{D}} + \bar{\delta}$
3: Entropy decode latent under finetuned prior: $\boldsymbol{z} = \mathrm{dec}(\boldsymbol{b}_{\boldsymbol{z}}; p_{\bar{\theta}}(\boldsymbol{z}))$
4: Decode image as the mean of the finetuned decoder: $p_{\bar{\theta}}(\boldsymbol{x}|\boldsymbol{z})$

4.2 MODEL ARCHITECTURE AND TRAINING ON FULL DATASET

Instance-adaptive compression is model-agnostic. As a proof of concept, we choose to finetune an image compression (I-frame) model on a video sequence as it has a smaller number of parameters than typical video compression models. Specifically, we adopt the hyperprior-model proposed by Ballé et al. (2018), including the mean-scale prior from Minnen et al. (2018) (without context).

This global model is trained on the training fold of the **CLIC19** dataset, using the RD objective given in eq. (1). Appendix B provides more details on both its architecture and the adopted training procedure.

4.3 INSTANCE-ADAPTIVE FINETUNING

We separately finetune the global model for each video in the **UVG** dataset. As a benchmark, we first finetune the *encoder only*, i.e. only parameters ϕ, as their updates do not have to be added to the bitstream after finetuning. Global models trained with RD tradeoff parameter $\beta \in \{3e{-}3, 1e{-}3, 2.5e{-}4\}$ are used to initialize the models that are then being finetuned with the corresonding value for β. The training objective is given by the RD-loss in eq. (1), and optimized with a constant learning rate of $1e{-}6$.

Secondly, we validate our instance-adaptive *full-model*-finetuning framework by jointly updating ϕ and θ using the RDM loss from eq. (2). In this case we finetune the global model that is trained with $\beta = 0.001$, independent of the value of β during finetuning. Empirically, this resulted in negligible difference in performance compared to using the global model of the corresponding finetuning β, while it alleviated memory constraints thanks to the smaller size of this low bit rate model architecture (see Appendix B).

The training objective in eq. (2) is expressed in bits per pixel and optimized using a fixed learning rate of $1e{-}4$. The quantization bin width t and standard deviation σ of $p[\bar{\delta}]$ were set to 0.005 and 0.05, respectively.

All finetuning experiments (both for *encoder-only* and *full-model*) ran for 50k steps, each containing one minibatch of a single, full resolution, frame from the video. For optimization, we used the Adam optimizer with its default settings (Kingma & Ba, 2014).

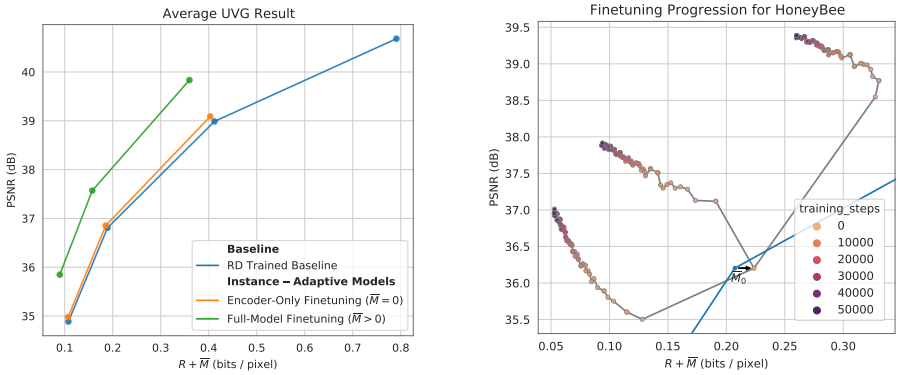
5 RESULTS

5.1 RATE-DISTORTION GAINS

Figure 2(a) shows the RD performance for encoder-only and full-model finetuning, averaged over all videos in the UVG dataset for different rate-distortion tradeoffs, i.e. different values for β. Finetuning of the entire parameter-space results in much higher RD gains (approximately 1 dB for the same bit rate) compared to only finetuning the encoding parameters ϕ. Figure 6 in Appendix C shows this plot for each video separately.

For this same experiment, Table 1 provides insight in the distribution of bits over latent rate R and model rate $\overline{M}$. Interestingly, for the three rate-distortion tradeoffs a similar number of bits is spent on updating the model, while bits for latent encoding differ per β. This could be caused by the small magnitude of $\overline{M}$ compared to R.

Figure 2(b) shows for one video how finetuning progressed over training steps. One can clearly see that the initial model rate costs $\overline{M}_0$ are quickly overcome by the large RD gains achieved at the start of finetuning. This effect was visible for all tested videos (Appendix C shows this plot for each video separately).



(a) (b)

Figure 2: RD performance for three different rate-distortion tradeoffs with $\beta = 3e{-}3$, $1e{-}3$, and $2.5e{-}4$ (from left to right). (a) Averaged performance over all videos of the UVG dataset. We compare *full-model* finetuning to *encoder-only* finetuning and the global model performance. Finetuning the entire model results in approximately 1 dB gain for the same rate. (b) Finetuning progression of the `HoneyBee` video over time. Between each dot, 500 training steps are taken, showing that already at the start of finetuning large RD gains are achieved, and the initial costs M_0 are quickly defeated. The RD performance continues to improve during finetuning.

β	PSNR (dB)	$R + \overline{M}$	R	$\overline{M}$	$\overline{M}_0$	$\overline{M} - \overline{M}_0$	$\overline{M}$ (bits/parameter)
		(bits/pixel)					
0.00300	35.914	0.088	0.068	0.020	0.018	0.0021	4.899
0.00100	37.643	0.154	0.134	0.020	0.018	0.0020	4.890
0.00025	39.930	0.351	0.332	0.020	0.018	0.0019	4.877

Table 1: Distribution of bit rate for different rate-distortion tradeoffs β, averaged over the videos in the UVG datset. The number of bits (per pixel or per parameter) are split over the latent rate R and the model rate $\overline{M}$, which is computed using the quantized model prior $p[\overline{\delta}]$.

5.2 Distribution of bits

Figure 3 shows for different (mutually exclusive) parameter groups the distribution of the model updates δ (top row). Interestingly one can see how for all, but the *Codec Decoder IGDN* group, the updates are being clipped by our earlier defined quantizer Q_t. This suggests the need for large, expensive updates in these parameter groups, for which the additional RD gain thus appears to outweigh the extra costs. It also suggests that using a slightly larger value for σ might be useful, this however comes with a higher number of quantization bins, and thus increased costs $\overline{M}$.

Figure 3 also provides the corresponding number of bits for all updates split per parameter group. Note that the scale of this bits-axis differs per group. In the *Hyperprior* plot (left column), one can clearly see that – thanks to entropy encoding – the (relatively large number of) small updates are encoded using a smaller number of bits than the large updates. Looking a the model rate $\overline{M}$ per parameter group, we see that most bits are "spent" for the convolutional weights in the decoder of both the *Codec* and the *HyperCodec*.

6 Discussion

In this work we presented instance-adaptive neural compression, the first method that enables finetuning of a full compression model on a single video, while restricting the additional bits for encoding the model updates. To this end, we proposed an extension of the typical rate-distortion loss by incorporating both model quantization and the additional model rate costs during finetunig. Under

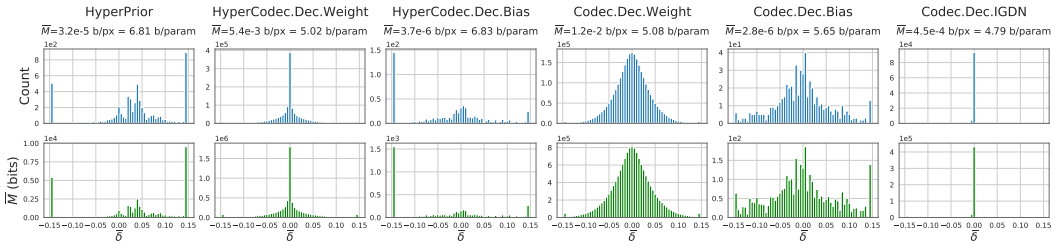


Figure 3: Empirical distribution of parameter updates for the model finetuned on the `HoneyBee` video with $\beta = 2.5e{-}4$. Columns denote parameter groups. Top: Histograms of the model updates $\overline{\delta}$. Bottom: Bit allocation as a function of $\overline{\delta}$. Subtitles indicate the total number of bits for each parameter group, both expressed in bits per pixel (b/px) and bits per parameter (b/param).

perfect optimization assumption, this loss guarantees pure RD performance gains after overcoming a small initial cost for encoding the zero-update. We showed improved RD performance on all videos in the UVG dataset, with an average distortion improvement of approximately 1 dB for the same bit rate.

Between videos, we found a difference in achieved finetuning RD gain (see Appendix C). Possible causes can be two-fold. First, the performance of the global model for each video differs, which can influence the potential finetuning gain. Second, some videos contain more motion and zooming than others, which might better be captured using a neural video compression model rather than our adopted I-frame model. The model rate costs (which directly scale with the number of decoding parameters of the model) were not found to be a bottleneck in this research, suggesting that even larger models, e.g. neural video compression models, could be used for instance-adaptive full-model finetuning in future research. Note that the size of the finetuning instance yields an important tradeoff as the model rate cost decreases with the size of the finetuning instance while the finetuning potential increases.

On the other hand, a much smaller decoding model architecture could be chosen, therewith moving the computational complexity of data compression from the receiver side to the sender side by heavily finetuning this small model on each datapoint. We foresee the convenience of this computational shift in applications where bandwidth and receiver compute power is scarce, but encoding compute time is not so important. Such a use case in practice for example happens for (non-live) video streaming services to low-power edge devices.

We foresee even higher gains for instance-adaptive full-model finetuning, by more exhaustively researching the settings for hyperparameters such as the learning rate (including a schedule), the standard deviation of the model prior σ and its quantization bin width t. The most natural extension would be to adopt (learned) layer- or parameter-dependent values for σ and t. In fact, not only the parameter settings, but also the form of the prior itself is subject to improvement, as from Table 1, it can be seen that the initial costs $\overline{M}_0$ cover the largest part of the total model rate $\overline{M}$ under the current Gaussian model prior. Choosing a more peaky distribution, e.g. a Laplace, lowers the initial bits $\overline{M}_0$, however this also comes with higher additional costs $\overline{M} - \overline{M}_0$. In the ideal case it not only shifts the partition in initial costs and additional model rate costs, but also lowers to total model rate for the same distortion. Lastly, more flexible model priors could be used, e.g. mixtures of Gaussians, or layer- or parameter-conditional priors. A hypernetwork (such as the hyperprior from Ballé et al. (2018)) could also predict the optimal parameters of the chosen prior. We leave extensions with more flexible priors for future research.

REFERENCES

Caglar Aytekin, Xingyang Ni, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska M Hannuksela. Block-optimized variable bit rate neural image compression. In *CVPR Workshops*, pp. 2551–2554, 2018.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Joaquim Campos, Meierhans Simon, Abdelaziz Djelouah, and Christopher Schroers. Content adaptive optimization for neural image compression. *arXiv preprint arXiv:1906.01223*, 2019.

Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. January 2018.

Tiansheng Guo, Jing Wang, Ze Cui, Yihui Feng, Yunying Ge, and Bo Bai. Variable rate image compression with content adaptive optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 122–123, 2020.

Amirhossein Habibian, Ties van Rozendaal, Jakub M Tomczak, and Taco S Cohen. Video compression with rate-distortion autoencoders. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 7033–7042. openaccess.thecvf.com, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2014.

Diederik P Kingma and Max Welling. Auto-Encoding variational bayes. December 2013.

Jan P Klopp, Liang-Gee Chen, and Shao-Yi Chien. Utilising low complexity cnns to lift non-local redundancies in video coding. *IEEE Transactions on Image Processing*, 2020.

Yat-Hong Lam, Alireza Zare, Caglar Aytekin, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska Hannuksela. Compressing weight-updates for image artifacts removal neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Yat-Hong Lam, Alireza Zare, Francesco Cricri, Jani Lainema, and Miska Hannuksela. Efficient adaptation of neural network filter for video compression. *arXiv preprint arXiv:2007.14267*, 2020.

Guo Lu, Chunlei Cai, Xiaoyun Zhang, Li Chen, Wanli Ouyang, Dong Xu, and Zhiyong Gao. Content adaptive and error propagation aware deep video compression. *arXiv preprint arXiv:2003.11282*, 2020.

Alexandre Mercat, Marko Viitanen, and Jarno Vanne. Uvg dataset: 50/120fps 4k sequences for video codec analysis and development. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pp. 297–302, 2020.

David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 10771–10780. Curran Associates, Inc., 2018.

L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/pdf?id=rJiNwv9gg`.

Yibo Yang, Robert Bamler, and Stephan Mandt. Improving inference for neural image compression. *arXiv preprint arXiv:2006.04240*, 2020.

Nannan Zou, Honglei Zhang, Francesco Cricri, Hamed R Tavakoli, Jani Lainema, Miska Hannuksela, Emre Aksu, and Esa Rahtu. L2 c–learning to learn to compress. *arXiv preprint arXiv:2007.16054*, 2020.

A QUANTIZATION

Figure 4 shows model update quantization (left) and its corresponding gradient based on the Straight-Through gradient estimator Bengio et al. (2013). Also the discrete (true) model rate $\bar{M}$ (middle) and its continuous analogy M (right) with their corresponding gradients are shown.

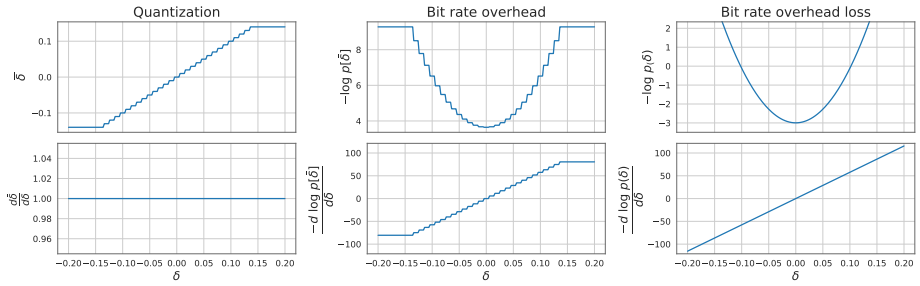


Figure 4: (Left) Illustrative example of the quantization effect and the corresponding gradient (using the Straight-Through estimator) for parameter update δ. (Middle) The bit rate overhead and its gradient with respect to δ. (Right) The continuous counterpart of the bit rate overhead used in the RDM loss function. It can be seen how the gradient of the continuous bit rate overhead is an unbiased, low variance estimator of the gradient of the discrete bit rate overhead.

During training, the continuous model rate term M is used as it has a smoother gradient. The proof below shows that $\nabla_\delta \bar{M}$ is an unbiased first-order approximation of $\nabla_\delta M$, validating its use during training.

The gradient of the continuous model rate loss M toward δ is defined as:

$$\nabla_\delta M = -\nabla_\delta \log p(\delta) = -\frac{\nabla_\delta p(\delta)}{p(\delta)} \tag{6}$$

The gradient of the non-differentiable discrete model rate loss $\bar{M}$ towards δ can be defined by exploiting the Straight-Through gradient estimator Bengio et al. (2013): $\nabla_\delta \bar{\delta} = 1$.

As such[4]:

$$\begin{aligned}\nabla_\delta \bar{M} = -\nabla_\delta \log p[\bar{\delta}] &= -\nabla_\delta \log[CDF(\bar{\delta} + \frac{t}{2}) - CDF(\bar{\delta} - \frac{t}{2})] \\ &= -\frac{p(\bar{\delta} + \frac{t}{2}) - p(\bar{\delta} - \frac{t}{2})}{CDF(\bar{\delta} + \frac{t}{2}) - CDF(\bar{\delta} - \frac{t}{2})}\end{aligned} \tag{7}$$

By first-order approximation we can write:

$$\nabla_\delta p(\bar{\delta}) \approx \frac{p(\bar{\delta} + \frac{t}{2}) - p(\bar{\delta} - \frac{t}{2})}{t} \tag{8}$$

and

$$CDF(\bar{\delta} + \frac{t}{2}) - CDF(\bar{\delta} - \frac{t}{2}) \approx p(\bar{\delta})t. \tag{9}$$

Using eq. (8) and eq. (9), we can finalize the gradient of the discrete model rate costs $\nabla_\delta \bar{M}$ as:

$$\nabla_\delta \bar{M} \approx -\frac{\nabla_\delta p(\bar{\delta})t}{p(\bar{\delta})t} = -\frac{\nabla_\delta p(\bar{\delta})}{p(\bar{\delta})}, \tag{10}$$

which results to be a first-order approximation of $\nabla_\delta M$.

[4]Due to cutting (a maximum) of 2^{-8} mass from the tails of $p(\delta)$ to faciliate quantization, the difference in cumulative masses in eq. (7) should be renormalized by $1 - 2^{-8}$. As this is a constant division inside a $\log$, it results in a subtraction of $\nabla_\delta \log 1 - 2^{-8} = 0$. As this normalization does not influence the gradient, we omitted it for the sake of clarity.

B GLOBAL MODEL ARCHITECTURE AND TRAINING

For our neural compression model, we adopt the architecture proposed by Ballé et al. (2018), including the Mean-Scale prior from Minnen et al. (2018). We use a shared hyperdecoder to predict the mean and scale parameters, resulting in a double number of feature maps in the last layer of the hyperdecoder. Like (Ballé et al., 2018) we use a model architecture with fewer parameters for the low bit rate regime ($\beta \geq 1e{-}3$). Table 2 indicates both the model architecture and the number of parameters, grouped per sub-model. The upper row in this table links the terminology proposed by Ballé et al. (2018) to conventional VAE terminology which we follow in this work.

Figure 5 provides a visual overview of the model architecture, where we use $\boldsymbol{z}_2$ and $\boldsymbol{z}_1$ to indicate the latent and hyper-latent space respectively (referred to as $\boldsymbol{y}$ and $\boldsymbol{z}$ in the original paper of Ballé et al. (2018)). Note that even though we adopt a hierarchical latent variable model, we simplify the notation by defining a single latent space $\boldsymbol{z} = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$ throughout the rest of this work.

For training we adopt a mixed quantization strategy, where we use the quantized latent (with a Straight-Trough gradient estimator) $\boldsymbol{z}_2$ to calculate the distortion loss, while we use noisy samples for $\boldsymbol{z}_1, \boldsymbol{z}_2$ when computing the rate loss. Optimization of eq. (1) on the training fold of the **CLIC19** dataset, was done using the Adam optimizer with default settings (Kingma & Ba, 2014), and took 1.5M steps for the low bitrate models, and 2.0M steps for the high bitrate models. Each step contained 8 random crops of 256×256 pixels, and the initial learning rate was set to $1e{-}4$ and lowered to $1e-5$ at 90% of training.

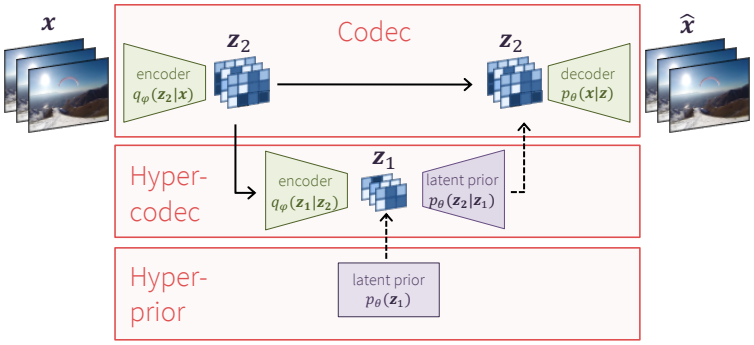


Figure 5: The mean-and-scale hyperprior model architecture visualized as a VAE framework.

Table 2: Model architecture and parameter count for the adopted hyperprior model. As suggested by Ballé et al. (2018), a distinct architecture is used for the low bit rates. We report the number of output channels per layer, for the exact model architecture we refer to Ballé et al. (2018). Note that the last column refers to the total number of parameters that need to be known at the receiver side.

	Transmitter		**Receiver**			
	Encoder $q(\boldsymbol{z}_2\|\boldsymbol{x})$	Hyper Encoder $q(\boldsymbol{z}_1\|\boldsymbol{z}_2)$	Hyperprior $p(\boldsymbol{z}_1)$	Hyper Decoder $p(\boldsymbol{z}_2\|\boldsymbol{z}_1)$	Decoder $p(\boldsymbol{x}\|\boldsymbol{z}_2)$	Total
Low bit rate model						
Layers x Output channels	4x192	3x128	3x3	2x128 + 1x256	3x192 + 1x3	-
Parameter count	2.89M	1.04M	2.89M	5.50k	1.26M	4.16M
High bit rate model						
Layers x Output channels	4x320	3x192	3x3	2x192 + 1x384	3x320 + 1x3	-
Parameter count	8.01M	2.40M	8.01M	8.26k	2.95M	10.97M

C RATE-DISTORTION FINETUNING PERFORMANCE PER VIDEO

Figure 6 shows the RD plots for the different videos after finetuning for 50k steps. Note that the blue lines, indicating global model performance, differ per video. All videos outperform both the global model and *encoder-only* finetuning, after finetuning the entire model. Note however that some videos seem more difficult to be finetuned than others, e.g. `Jockey` versus `HoneyBee`.

Figure 7 shows for each of these videos the finetuning progression split over training steps. It can be seen that for all videos, the initial cost $\overline{M}_0$ are mostly quickly overcome. Only in the low bit rate regime it sometimes takes more training steps to outperform the global model, e.g. for Beauty. Differences in performance are visible among videos. The videos that most easily finetune, e.g. `HoneyBee`, show quicker performance improvent after the start of training, but also continue to improve more over time.

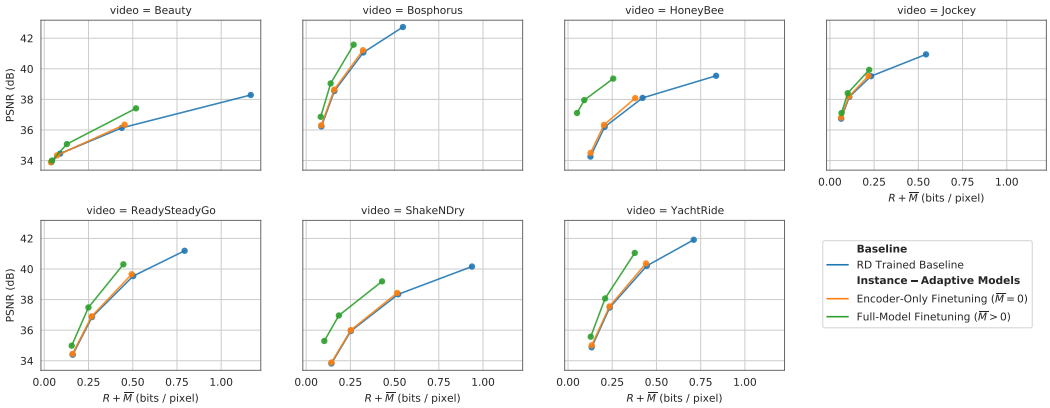


Figure 6: RD performance for instance-adaptive encoder-only and full-model finetuning, compared with the performance of the global model. Each plot is evaluated on a different video, and the instance-adaptive models used to create these graphs are finetuned on this corresponding single video.

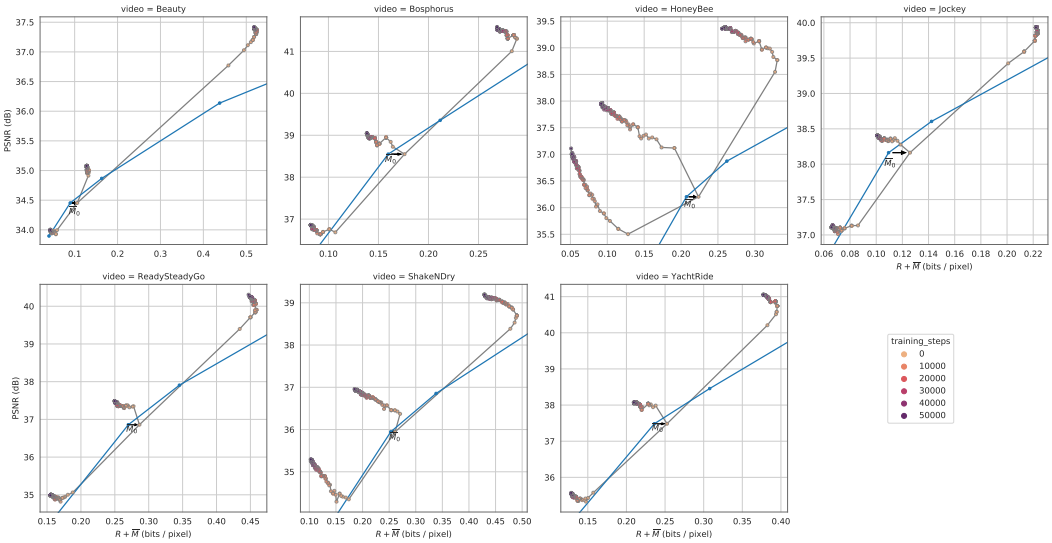


Figure 7: Progression of finetuning over time for all videos from the UVG dataset.

D LIST OF UVG VIDEOS

- `Beauty 1920x1080 8bit YUV Y4M`
- `Bosphorus 1920x1080 8bit YUV Y4M`
- `HoneyBee 1920x1080 8bit YUV Y4M`
- `Jockey 1920x1080 8bit YUV Y4M`
- `ReadySetGo 1920x1080 8bit YUV Y4M`
- `ShakeNDry 1920x1080 8bit YUV Y4M`
- `YachtRide 1920x1080 8bit YUV Y4M`