Supplementary Material

Carlos Gomes
ETH Zürich
Zürich, Switzerland
cagomes@ethz.ch

Roberto Azevedo
DisneyResearch|Studios
Zürich, Switzerland
roberto.azevedo@disneyresearch.com

Christopher Schroers
DisneyResearch|Studios
Zürich, Switzerland
christopher.schroers@disneyresearch.com

This document is supplementary material for CVPR 2023 paper #11419: "Video compression with Entropy-constrained Neural Representations". We provide additional architecture and training details (Section 1), and brings more information on how to reproduce the baseline results (Section 3).

1. Architectural and training details

Algorithm 1 presents the pseudo-code for our training step.

Algorithm 1 Training step

```
Require: $\theta$, $\phi$, $\gamma$, $optimizer$, $coord$, $target$, $\lambda$
    $\hat{\theta} \leftarrow Q^{-1}(Q_{ste}(\theta;\gamma);\gamma)$
    $D \leftarrow distortion\_metric(f(coord;\hat{\theta}), target)$
    $\tilde{\theta} \leftarrow Q_{noise}(\theta;\gamma)$
    $R \leftarrow \frac{\sum_{\tilde{\theta}} -\log_2 p_\phi(\tilde{\theta})}{frames \times h \times w}$        ▷ where p is the entropy model
    $Loss \leftarrow D + \lambda \times R$
    $optimizer(\theta, Loss)$        ▷ takes an optimization step
```

Figure 1 illustrates a forward and backward pass through our model.

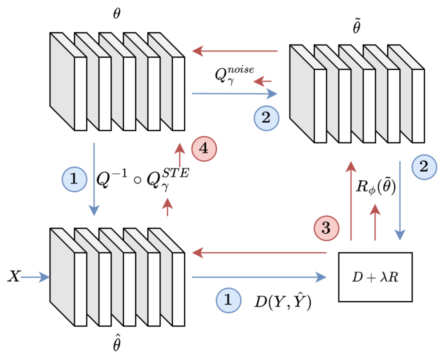

Figure 1. Depiction of a forward and backward pass using quantization-aware training and the R-D loss. **1.** The distortion metric is computed given input coordinates X and weights with quantization error. **2.** To compute the rate term, quantization is replaced with uniform noise. **3.** Gradients flow back to the parameters of the entropy model and the quantization functions of each layer **4.** Gradients flow to θ and the STE approximates the gradients from the quantization operation.

Table 1 shows the hyper-parameter values of our proposal. As discussed in Section 4 of the paper, for our experiments, we first overfit the model without entropy constraint, i.e., $\lambda = 0$, and then for different target bitrates, we fine-tune the model with different λ values.

Hyperparameter	Value
Dimension of Spatial-Temporal Features	$16 \times 9 \times$ (160 if small, 200 if large)
Depth of Convolutional Feature Fusion Layers	200, 200
Upsampling per Convolutional Upscaling block	5, 3, 2, 2, 2 (5, 2, 2, 2, 2 for "BigBuckBunny")
Expansion (Large, Small)	5, 3
Learning Rate	5e-4
Learning Rate Schedule	0.2 * Number of Epochs linear warmup Cosine Learning Rate Decay after
Number of Epochs	1500 (1200 with $\lambda = 0$ and 300 with $\lambda =$ target)
Activation Function	GELU
λ (Small)	0.1, 0.5, 1
λ (Large)	0.01, 0.05, 0.1

Table 1. Parameters for Video Compression for the UVG dataset. Training with no λ for 1200 epochs. LR reset for finetuning with λ

2. Additional results

In the paper we present the aggregated results for the 7 videos from the UVG dataset used in our experiments. In what follows, the reader can also find the individual rate-distortion plots for each video (Figure 2), which allow to further see the strength and limitations of the proposed method. The following results for NeRV were computed by the authors of this paper using the publicly available repository (see Section 2 for more details).

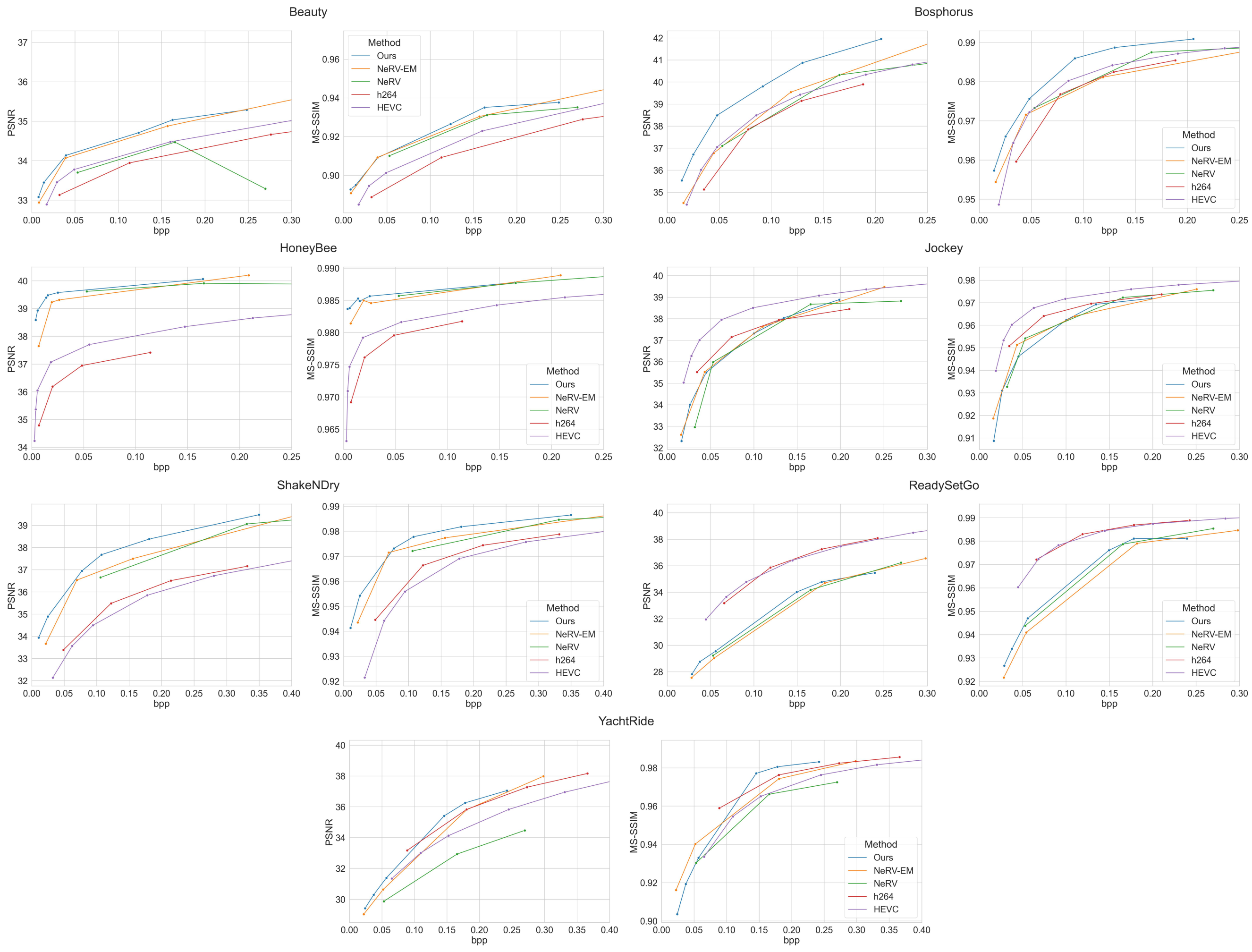

Figure 2. Per-video rate-distortion curves.

3. Baseline reproducibility

3.1. NeRV

To adapt the NeRV architecture to train on individual videos, rather than on a concatenation of videos as proposed in the original paper, we use slightly smaller architectures. The following are the commands we use for training, using the code made available by the authors at https://github.com/haochen-rye/NeRV. To train the models, we use the following command:

```
python train_nerv.py -e 1500 --num-blocks 1 --dataset {dataset} --frame_gap 1 --outf {
    outf} --embed 1.25_80 --stem_dim_num 512_1 --reduction 2 --fc_hw_dim 9_16_{Depth} --
    expansion {Expansion} --single_res --loss Fusion6 --warmup 0.2 --lr_type cosine --
    strides 5 3 2 2 2 --conv_type conv -b 6 --lr 0.0005 --norm none
```

followed by the pruning and fine-tuning with:

```
python train_nerv.py -e 50 --num-blocks 1 --dataset {dataset} --frame_gap 1 --outf {outf}
     --embed 1.25_80 --stem_dim_num 512_1 --reduction 2 --fc_hw_dim 9_16_{Depth} --
    expansion {Expansion} --single_res --loss Fusion6 --warmup 0. --lr_type cosine --
    strides 5 3 2 2 2 --conv_type conv -b 6 --lr 0.0005 --norm none --weight {weight_path
    } --not_resume_epoch --prune_ratio 0.8
```

and finally obtaining the results with:

```
python train_nerv.py -e 50 --num-blocks 1 --dataset {dataset} --frame_gap 1 --outf {outf}
     --embed 1.25_80 --stem_dim_num 512_1 --reduction 2 --fc_hw_dim 9_16_{Depth} --
    expansion {Expansion} --single_res --loss Fusion6 --warmup 0. --lr_type cosine --
    strides 5 3 2 2 2 --conv_type conv -b 1 --lr 0.0005 --norm none --weight {
    pruned_weight_path} --prune_ratio 0.8 --eval_only --quant_bit 8 --quant_axis 1 --
    dump_images
```

.

We use a large, medium and small model size, for which we select a Depth parameter of 128, 128 and 64 and an Expansion parameter of 6, 4 and 4 respectively.

3.2. HEVC

To generate results for HEVC, we first extract frames from the original video file (which can be downloaded at https://ultravideo.fi/#testsequences) with the following commands:

```
mkdir {name_of_file}
ffmpeg -i {name_of_file}.y4m {name_of_file}/frame%05d.png
```

We use these frames for the training of all models. To compress with HEVC, we use:

```
ffmpeg -i {name_of_file}/frame%05d.png -c:v hevc -preset medium -x265-params bframes=0 -crf
    {desired_crf} {name_of_output_video}.mp4
```