Video Compression with Entropy-Constrained Neural Representations

Carlos Gomes
ETH Zürich
Zürich, Switzerland
carlosmiguel.gomes@live.com.pt

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

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

Abstract

Encoding videos as neural networks is a recently proposed approach that allows new forms of video processing. However, traditional techniques still outperform such neural video representation (NVR) methods for the task of video compression. This performance gap can be explained by the fact that current NVR methods: i) use architectures that do not efficiently obtain a compact representation of temporal and spatial information; and ii) minimize rate and distortion disjointly (first overfitting a network on a video and then using heuristic techniques such as post-training quantization or weight pruning to compress the model). We propose a novel convolutional architecture for video representation that better represents spatio-temporal information and a training strategy capable of jointly optimizing rate and distortion. All network and quantization parameters are jointly learned end-to-end, and the post-training operations used in previous works are unnecessary. We evaluate our method on the UVG dataset, achieving new state-of-the-art results for video compression with NVRs. Moreover, we deliver the first NVR-based video compression method that improves over the typically adopted HEVC benchmark (x265, disabled b-frames, "medium" preset), closing the gap to autoencoder-based video compression techniques.

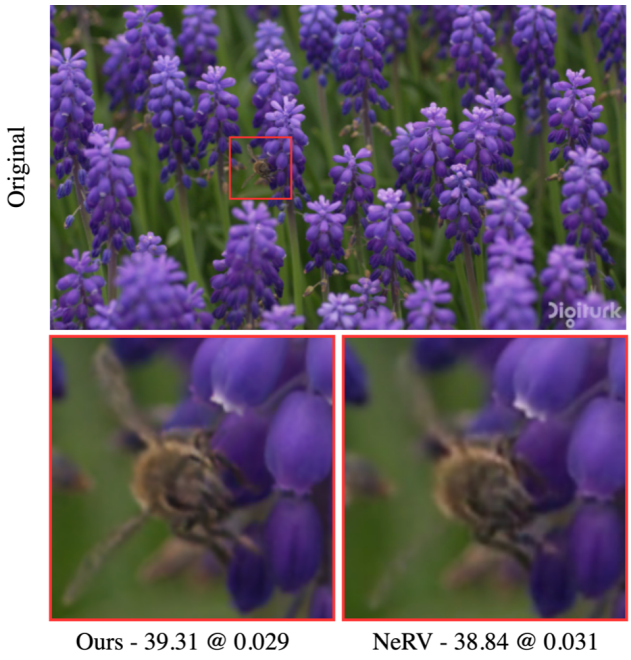

Figure 1. Compared to the previous state-of-the-art method, our approach produces sharper frames at lower bpp. Images are labeled with PSNR @ bpp.

1. Introduction

Lossy video compression is a Rate-Distortion (R-D) optimization problem of the form $\min D+\lambda R$. Given a video, the encoder's task is to minimize the number of bits required to represent it, R (Rate), while also minimizing any distortion brought about by the compression, D (Distortion). λ controls the trade-off between both and defines a Pareto frontier, where improvements in the distortion term come at an increased cost in the rate term. Traditionally, heuristic-based engineered video codecs are used to encode videos in efficient representations. Such codecs have gradually developed into complex pipelines and have been established as powerful standards such as H.264 [26] and HEVC [30].

Inspired by the success of deep learning in many image processing tasks, over the past few years, video compression using neural networks has been the target of much research [8, 18]. Most of the proposed methods are based

on Encoder-Decoder neural network pairs, capable of transforming Groups of Frames (GoF) into latent representations and subsequently recovering them. These representations are quantized, entropy coded, and then stored/transmitted. Commonly, the loss function optimized by these methods is theoretically grounded in information theory and structured as an R-D problem. D is some measure of the difference between the original and decoded frames, and R is given by an estimate of the lower bound of the length of a bit sequence representing the latent vector.

More recently, Implicit Neural Representations (INRs) have emerged as alternatives to dense grids for representing continuous signals. They have seen remarkable success in 3D scene reconstruction and shape representation [22, 24], and have also been used for video representation [3, 7, 33]. In such a case, a video is interpreted as a function $f(x, y, t) = (R, G, B)$ which is approximated by fitting a neural network to a set of samples $S = \{(x, y, t), (R, G, B)\}$. The video is then effectively stored in the neural network's parameters and can be recovered by performing forward passes. Interestingly, by using INRs, video compression can be framed as a neural network compression problem.

Previous efforts in using INRs for video compression, however, have mostly treated the problems of representing the input signal with high fidelity and compressing it as mostly disjoint tasks. A network is first trained to minimize a distortion loss and is then put through some procedure to reduce its size, e.g., storing its weights in a 16-bit floating point format, quantization, or pruning [7, 10, 29]. Furthermore, current architectures are not designed with parameter efficiency as a priority, suffering from an inefficient allocation of parameters, which can be prohibitive for the task of video compression [7, 16].

Tackling the above issues, we propose a new compact convolutional architecture for video representation that provides better R-D performance than previous works (see Figure 1). In addition, drawing on information theory, we propose a theoretically motivated R-D formulation for video compression with INRs that jointly minimizes rate and distortion. We build on the work of Oktay *et al*. [23] and model the entropy of the neural network weights, allowing us to minimize it jointly with the distortion. Thus, our method learns weights that simultaneously provide high-fidelity representations of the original video and have low entropy. Applying entropy coding methods to the final weights produces a compressed video representation.

In summary, our main contributions are:

1. we propose a novel compact convolutional architecture for neural video representation, which results in better representation capacity than NeRV [7] and faster encoding and decoding than E-NeRV [16];

2. we formally define signal compression with INRs as an R-D problem by modeling the entropy of the weights and using quantization-aware training (allowing end-to-end training and eliminating the need for post-training techniques such as pruning);

3. we show that such an entropy modeling can also improve other methods, e.g., NeRV;

4. we evaluate our method on the UVG [21] dataset, improving on the state-of-the-art results for video compression with INRs and outperforming DVC [17], a well-established neural video compression method.

2. Related Work

Neural Image Compression State-of-the-art approaches to video compression can be traced back to the task of image compression. Balle *et al*. [4] use non-linear transform coding, where an image, represented as a vector of pixel intensities $\mathbf{x}$, is transformed through some function g and then quantized, introducing some error. This quantized latent representation is then passed through a Shannon-style encoder that losslessly compresses it. $\mathbf{x}$ can then be approximately recovered using another transform $g\prime$. g and g' are parameterized as neural networks, and the entropy of the latent vector is modeled, allowing for the loss to be formulated as an R-D problem. Targeting low-bitrate compression, generative adversarial networks (GANs) have also been proposed to hallucinate details during inference [2,20].

Neural Video Compression Neural video compression extends the ideas of neural image compression to sequences of frames. Typically, an Encoder-Decoder pair of models are learned over a large dataset of videos. The encoder produces latent representations of the sequence of frames that are quantized and entropy coded. The decoder is then able to approximately reconstruct the original sequence. Approaches of this type commonly embed temporal consistency priors into the model by electing a key-frame within a group of frames (GoP) and transmitting a latent representation of it. The information for reconstructing the remaining frames in the GoP, e.g., optical flow and residuals, is then encoded into separate vectors. Lu *et al*. [17] propose DVC, an end-to-end approach to neural video compression based on the pipeline of traditional video codecs, but leveraging learned neural networks instead of hand-crafted algorithms. Several works build on this approach, proposing different forms of representation for the key and predicted frames [1,9,19]. The common pillar of these methods is the definition of the loss to be optimized as a Rate-Distortion problem, theoretically grounded in concepts from information theory. As demonstrated by Mentzer *et al*. [19], accurately modeling the entropy in this problem may even preclude the need for the injection of biases and priors through

complex encoding and decoding pipelines.

Implicit Neural Representations (INRs) INRs are learned, parametrized functions that can be used as continuous representations of signals, usually in the form of neural networks. They can be trained directly on samples from the signal to be fit (e.g., $(x, y) \rightarrow (R, G, B)$ pairs for images) or indirectly on outputs produced from the signal by a known differentiable process (e.g., differentiable rendering in the case of NeRF [22]) to fit said signal. One issue with these networks is the spectral bias problem [25], where they struggle to represent high-frequency details. Sitzmann *et al*. [29] introduce SIREN, using periodic activation functions to combat this phenomenon, and demonstrate their unequivocal benefits in representing images and videos when compared to traditional activation functions. Mildenhall *et al*. [22] use a different strategy to overcome the spectral bias. Before passing the input coordinates to the network, an extra step, called positional encoding, is applied. This step maps the coordinates to a higher dimensional space obtained by applying sinusoidal functions with a range of frequencies to each coordinate and concatenating the results in a vector. As in previous works in INR-based video compression, we also adopt positional encoding.

INRs for compression Dupont *et al*. [10] suggest using INRs for image compression by fitting a network on the image and storing its weights in a 16-bit floating point format. They show superior results compared to JPEG in low bit-rate regimes, encouraging more investigation into INRs as an alternative to traditional neural image compression. Further approaches [11, 29] introduce explicit quantization methods and meta-learning. In these works, a single INR is learned over a dataset of images. To encode an image, only small modulations to this network must be learned and transmitted. Strümpler *et al*. [29] achieve state-of-the-art results for image compression using INRs by then quantizing these modulations and fine-tuning them to recover performance. They additionally introduce the use of L1 regularization, arguing that it approximately induces lower entropy in the weights. When varying the strength of this regularization, however, their results reveal only a minor effect in the compressed representation, suggesting L1 regularization may not be a good surrogate for entropy minimization.

Video compression with INRs NeRV [7] proposes to represent video by learning an INR for all pixel values jointly. They depart from traditional INRs by making use of convolutional layers, learning a mapping from a time coordinate, representing the frame number, to a full frame, decoded from this frame number: $f : \mathbb{R} \rightarrow \mathbb{R}^{x \times y}$. Although continuity is lost in the spatial domain, it allows the network to leverage convolutional layers, more commonly used in image processing applications than fully connected layers. Their results show that image-based neural video representations can provide superior performance than (continuous) pixel-based representations for the task of video compression. Additionally, training and inference are significantly faster. Li *et al*. [16] propose E-NeRV, an optimization to the NeRV architecture, by disentangling the spatial and temporal components of the input. This allows them to eliminate the large fully connected layers required by NeRV, resulting in a more efficient allocation of network parameters. Both NeRV and E-NeRV require the training of networks with different architectures to achieve different R-D tradeoffs and applies post-training quantization and weight pruning to achieve compression. While this process achieves smaller networks, the compression and representation components are essentially disjoint. Our proposed architecture is more streamlined while achieving comparable quality to E-NeRV and resulting in faster encoding and decoding times. Moreover, while E-NeRV focuses mainly on the representation capacity of the network, we propose to use an end-to-end training method based on entropy minimization for lossy video compression, which removes the need for post-training quantization and pruning.

Entropy Minimization for Model Compression Oktay *et al*. [23] adapted entropy minimization techniques [4, 5] for neural network compression. They introduce a reparametrization for neural network weights, which can be seen as performing a similar role to the transforms in Balle *et al*. [4] and model these reparameterized weights in a similar way as Balle *et al*. [5] models image latents. Any neural network can then be trained on its original loss function with an additional entropy penalty term. They perform comparisons with state-of-the-art neural network compression techniques, demonstrating that entropy minimization can offer a simpler and more elegant solution while achieving similar or better results. Part of our work is an extension of this idea, adapting entropy minimization techniques to INRs for video compression.

3. Method

Our work consists of two proposals for the improvement of neural representations for video compression: a more efficient neural network architecture for video compression (Section 3.1) and the formalization of the task as an R-D problem (Section 3.2).

3.1. Architecture

Frame-based INRs, introduced by Chen *et al*. [7], have significant advantages over pixel-based representations in terms of computational efficiency and R-D performance for video compression. However, the initially proposed NeRV architecture suffers from inefficient use of parameters, mainly due to its reliance on fully connected layers to produce a tensor of spatio-temporal features from the scalar

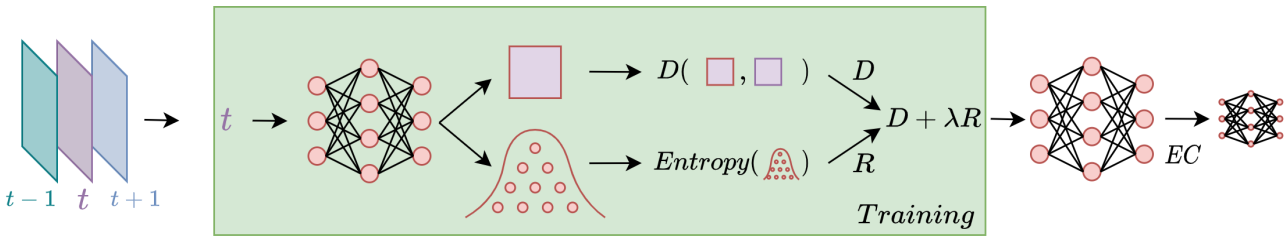

Figure 2. Illustration of our proposed method. We use an INR to fit a sequence of frames. During the training process, we minimize some distortion metric together with the entropy of the weights. After the training process, any entropy coding method can be used to obtain a compressed representation of the video sequence.

time input. [16] These layers require a very large number of weights to produce outputs whose size is suitable for being reshaped and fed into convolutional layers.

Motivated by such issues, we propose a fully convolutional design. The single input to the network is the frame number normalized to be in the range $[0, 1)$. Using this input, we build a matrix $T \in \mathbb{R}^{1\times h\times w}$, where every element is set to t. This is concatenated with a fixed coordinate grid $M \in \mathbb{R}^{2\times h\times w}$ where $M[0, h\prime, w\prime] = \frac{w\prime}{W}$ and $M[1, h\prime, w\prime] = \frac{h\prime}{H}$, for a target video with a resolution of $W \times H$. Positional encoding [22], as seen in Equation (1), is then applied to each element of the resulting tensor, followed by two convolutional layers with 3x3 kernels and 160 channels. We thus obtain a tensor of spatio-temporal features that is passed to the upscaling portion of the network.

$$\gamma(x) = \left(\sin\left(1.25^0\pi x\right), \cos\left(1.25^0\pi x\right), \cdots, \sin\left(1.25^{L-1}\pi x\right), \cos\left(1.25^{L-1}\pi x\right)\right) \quad (1)$$

As in NeRV, this is made up of a series of upscaling blocks, each consisting of a convolutional layer and a PixelShuffle module [28]. We follow Li *et al.* [16] and introduce an AdaIN [14] module at the beginning of each block. For each block, there is also a single fully connected layer that processes the temporal input coordinate to produce the inputs for each AdaIN module. While this means our model technically contains non-convolutional layers, these make up a very small part of the total number of parameters of the model ($\approx 2\%$ in our smallest model and $\approx 0.6\%$ in our largest). As our distortion objective, we adopt the loss used in NeRV, shown in Equation (2). While any reasonable distortion metric can be used with our method, we make this choice to enable a more direct comparison with NeRV in the context of this work. This is a mixture of L1 and SSIM, where x is the original frame and x' is the network output.

$$D(x, x') = 0.7\times\|x-x'\|_1+0.3\times(1-SSIM(x, x')) \quad (2)$$

3.2. Entropy Minimization

We first approach the problem of video compression with neural representations from the perspective of compressing any signal in general. Let us consider compactly representing a signal $s : \mathbb{R}^I \to \mathbb{R}^O$, with I the dimension of the input coordinates and O the dimension of the signal, using an INR. We do this given access only to a set of samples from the signal $S = \{(x_1, y_1), ..., (x_n, y_n) \mid x_i \in \mathbb{R}^I, y_i \in \mathbb{R}^O\}$ consisting of input coordinates x_i and target values y_i. We use an INR parametrized by θ, $f_\theta : \mathbb{R}^I \to \mathbb{R}^O$, to approximate s, taking as input any coordinate x and producing an approximation to the target value y. We can then recover s by densely sampling from f_θ, with the signal s effectively becoming stored in the parameters θ. To achieve compactness, we frame this as a Rate-Distortion problem.

In such a problem, we seek to find θ that minimizes the quantity $D + \lambda R$, where R represents the cost of storing θ, D represents the distortion between f_θ and s, and λ establishes the trade-off between the two. As a surrogate for s, we minimize this over the dataset S using gradient descent. A larger value of λ will give more weight to R in the optimization, resulting in a more compact signal representation, potentially at the cost of some added distortion. A smaller value of λ will have the opposite effect.

Signal Compression with Neural Representations We build on the work of Oktay *et al.* [23] to train an INR following this framework. Concretely, D can be defined as any reasonable metric that captures the signal's distortion and that we wish to optimize for. R is defined as the amount of information encoded in the parameters θ, and is given by Shannon's source coding theorem [27] as $-\log_2 p(\theta)$, with p being the probability over the set of all weights. This can also be interpreted as a tight lower bound on the number of bits occupied by the entropy-coded parameters. We minimize both these quantities during training jointly. Once the final weights are obtained, any form of entropy coding can be used to encode them, achieving a compact representation of the signal that approaches this lower bound. Figure 2 il-

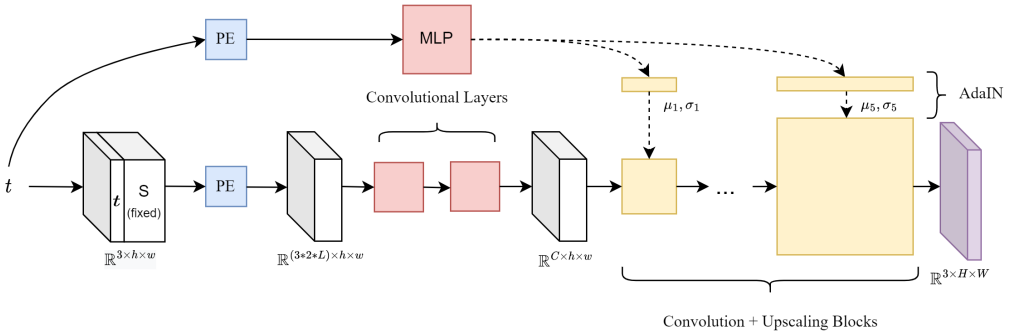

Figure 3. The frame index is taken as input and expanded into a matrix. It is then appended to a fixed coordinate grid, which is fed into our network after a positional encoding step. Dotted lines show the AdaIN module, present in each upscaling block.

lustrates the full approach.

To make use of Shannon's source coding theorem, we must work with a discrete set of symbols. However, for optimization, we require continuous weights. To fit this paradigm, we define: i) a quantization function Q_γ, with learnable parameters γ, mapping continuous weights to discrete symbols, and ii) a dequantization function Q_γ^{-1}, mapping the symbols to the values at the center of their respective quantization bins. Q_γ^{-1} is evidently not an exact inverse of Q_γ, and thus the operation $Q_\gamma^{-1}(Q_\gamma(x))$ incurs an error in recovering x unless the value of x is precisely one of the centers of the quantization bins.

We optimize over the continuous parameters θ, using the symbols $\hat{\theta} = Q_\gamma(\theta)$ to perform calculations for the rate and the weights with quantization error $Q_\gamma^{-1}(\hat{\theta})$ to perform the forward pass with the network and obtain an approximation of the signal. We further make the simplifying assumption that $\hat{\theta}$ consists of symbols produced by a memoryless source. The optimization problem thus becomes

$$\min_{\theta} \sum_{(x,y)\in S} D(f(x; Q_\gamma^{-1}(\hat{\theta})), y) + \lambda \sum_{i=0}^{|\hat{\theta}|} -\log_2 \hat{p}(\hat{\theta}_i), \quad (3)$$

where $\hat{p}$ is the pmf of $\hat{\theta}$, which can be readily computed. To optimize this loss, the process minimizes the distortion by learning parameters θ that can appropriately represent the signal, and γ that provide a small enough quantization error. Simultaneously, the distribution of $Q_\gamma(\theta)$ must also have a sufficiently small entropy, to minimize the R.

From the above, we identify the two error sources introduced in this process. The first is the error introduced in approximating s with f_θ, which can be minimized by increasing the number of parameters used to model s, making better choices in the architecture of the INR, among others. The second is the quantization error introduced by Q_γ, which can be minimized by shifting the centers of quantization bins appropriately or using more bins of smaller widths at an increased cost in the entropy of the distribution.

Quantization We now define the function Q_γ. Following [12, 23], we use scalar quantization, taking the integers as our discrete set of symbols and defining Q_γ as an affine transform with scale and shift parameters α and β respectively, followed by rounding to the nearest integer

$$Q_\gamma : \mathbb{R} \to \mathbb{Z}, Q_\gamma(x) = \left\lfloor \frac{x+\beta}{\alpha} \right\rceil, \gamma = \{\alpha, \beta\}. \quad (4)$$

Q_γ^{-1} is then naturally defined as

$$Q_\gamma^{-1}(x) = x \times \alpha - \beta. \quad (5)$$

As in [12, 23], each layer of the neural network is quantized separately and has its own parameters α and β, which are themselves learned. This allows for some level of granularity in varying the quantization of different parameters, while not incurring too large of an overhead in the number of scale and shift parameters, which must also be stored.

One issue with this process is the non-differentiability of the rounding operation. There are two main approaches to this problem in the literature. The first is the replacement of the rounding operation with uniform noise of the same scale as the quantization bins. This is frequently used as a replacement for quantization [13]. The second is the use of the Straight Through Estimator (STE) when computing the gradient for the rounding operation. We define these as two functions, Q_{noise} and Q_{ste}. As in Balle *et al.* [4], we obtain the best results using Q_{ste} for calculating the distortion metric, as it avoids the introduction of random noise, and Q_{noise} for calculating the entropy term.

Entropy modeling Given $\hat{\theta}$, we can calculate the minimum bit length to encode all the weights in the network exactly as follows:

$$\sum_{w\in\hat{\theta}} -\log_2 \hat{p}(w) \quad (6)$$

where

$$\hat{p}(w) = \frac{1}{|\hat{\theta}|} \sum_{w' \in \hat{\theta}} \mathbb{1}_{w'=w}. \tag{7}$$

The problem with this approach lies in the non-differentiable operator $\mathbb{1}$. To train a network with gradient descent, we need to find a differentiable approximation to the discrete distribution of the weights. We follow Balle *et al*. [4] and replace the discrete rate term with a differential entropy by replacing Q with Q_{noise}. We then seek a parametrized function p_ϕ that approximates the probability density function of the parameters perturbed by uniform noise $\tilde{\theta}$.

We can elegantly fit the parameters of this approximation jointly with the parameters of the INR using the same loss function presented above [4]. Additionally, we convolve the approximation p_ϕ with the standard uniform density. Balle *et al*. [4] argue that this enables a better approximation of the underlying distribution, which we also empirically observe.

Given p_ϕ, our complete loss is defined as

$$\min_{\theta,\phi,\boldsymbol{\gamma}} \sum_{(x,y)\in S} D(f(x; Q^{-1}(Q_{ste}(\theta;\boldsymbol{\gamma});\boldsymbol{\gamma})), y) + \lambda \frac{\sum_{l\in layers} -\log_2 p_\phi(Q_{noise}(\theta_l;\gamma_l))}{frames \times h \times w}$$

where $\boldsymbol{\gamma}$ collects all α and β from each layer. The left term computes the distortion metric over the dataset using the quantized weights, which are computed using each layer's respective α and β. The right term approximates the minimum bit length to encode the approximately quantized parameters using p_ϕ. This rate term is divided by the total number of pixels, making λ invariant to the video's resolution and number of frames.

To model the weights of the neural network, we interpret weights in each layer as being generated from an independent source. Within each layer, weights are taken as i.i.d. samples. We experiment with different modalities, such as modeling all weights in the network jointly, but find that we obtain better results by allowing each layer to be modeled independently. Following Balle *et al*. [5], we fit a small neural network to the distribution of weights in each layer. The details of this process can be consulted in [5, Appendix 6.1]. At the end of the training process, we use the context-adaptive binary arithmetic coder (CABAC) for entropy coding, adapting the implementation provided by DeepCABAC [32] to perform vanilla CABAC encoding on the weights of the neural network.

4. Experiments

Representational Capacity To validate our architecture, we perform a comparison with NeRV [7] and E-NeRV [16] with no compression (no entropy modeling or quantization). We use models with a similar number of parameters and train them on the "BigBuckBunny" sequence for 300 epochs using the Adam optimizer with a learning rate of 5e-4 and a cosine learning rate schedule. We set the dimensions of the spatio-temporal representation to $9 \times 16 \times 200$ for our model. Table 1 shows the results, in which our simpler model is able to retain the improvements of E-NeRV over NeRV with a significantly smaller performance penalty.

Model	Parameters	PSNR	Time	Time Δ	FPS
NeRV	12.5M	41.1	21m	—	88
E-NeRV	12.5M	42.8	53m	152%	45.3
Ours	12.5M	42.7	32m	52%	56

Table 1. Representational capacity results. For NeRV, we use the architecture the authors employ for the UVG dataset, adapted for the smaller resolution. Time increase is shown relative to NeRV. For decoding, we used a batch size of 20. GPU: RTX 3090

Video Compression We evaluate our results on the UVG dataset, featuring 7 videos with a resolution of 1920×1080. We compare our approach with: i) HEVC, a traditional video codec; ii) NeRV; iii) NeRV-EM, NeRV extended with our entropy minimization proposal; iv) DVC [17], a classic neural video compression method; and v) Scale-Space [1], a neural video compression method achieving results close to the state-of-the-art. As our metrics for perceptual similarity, we employ PSNR and MS-SSIM.

Entropy Minimization as Fine-Tuning To reduce training time, we leverage the flexibility of our method, treating entropy minimization as a fine-tuning process given a pre-trained INR for the video to be compressed. For the desired number of total epochs e, we train the INR for $0.8 \times e$ epochs with $\lambda = 0$, a faster process since it does not require the calculation of the R term in the loss. The remaining epochs are trained with the target λ. This allows us to reuse the same model to encode videos of different quality, resulting in large resource savings. Using this method, we observe only a very minor degradation in performance compared to training a model from scratch for each desired λ. We use a learning rate of 5e-4 with the Adam [15] optimizer, adopting a cosine learning rate schedule after a linear warm-up period of $0.2 \times$ the number of epochs. This learning rate schedule does not apply to the parameters of the entropy models. Finally, we set $L = 80$ for the positional encoding step.

While in theory we can achieve different compression rates while changing only λ, we find that, as λ gets large, it is helpful to reduce the size of the network. We use a larger (50M params.) and a smaller (14M params.) archi-

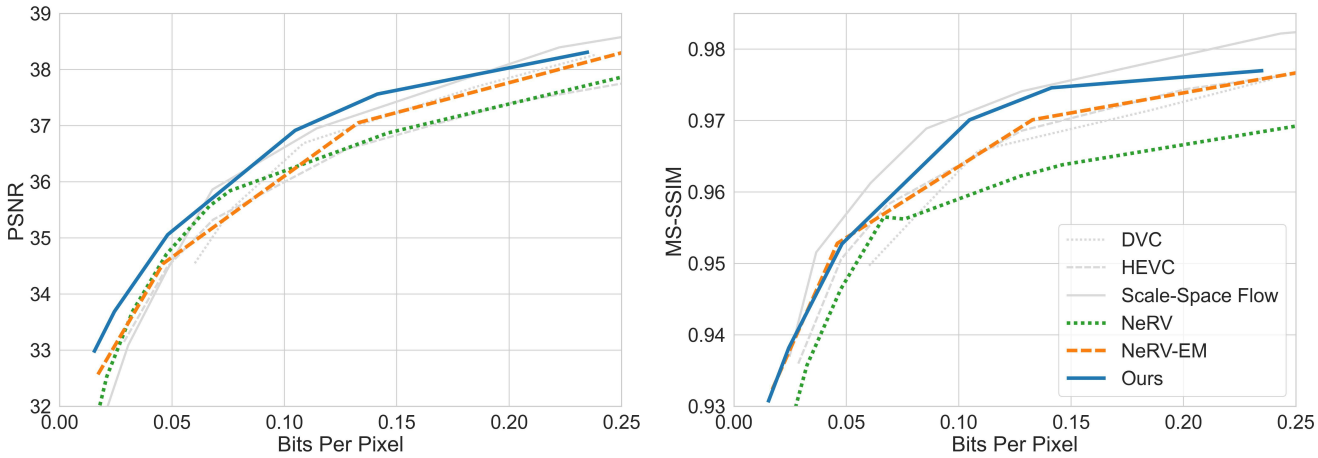

Figure 4. R-D performance on the UVG dataset. Our method outperforms the previous state-of-the-art using INRs. The individual contributions of the joint R-D optimization and the novel architecture are evident in the plot.

tecture to cover different ranges of the bpp scale. Each architecture makes use of the fine-tuning procedure defined above separately. Thus, we compromise between a different architecture for each bit-rate and a single architecture for all bit-rates. Architectural details can be consulted in the Supplementary Materials.

For each video, we train the small network on $\lambda \in \{0.1, 0.5, 1\}$ and the large network on $\lambda \in \{0.01, 0.05, 0.1\}$. We then average these points across all videos, grouping them by architecture and λ, and concatenate the resulting points into a curve. Figure 4 shows the R-D curves for our models compared to the baseline methods for the UVG dataset.

Evaluation Our method outperforms NeRV on both evaluated metrics, exhibiting the advantages of the streamlined architecture and the end-to-end training procedure, and establishing new state-of-the-art results for INR-based video compression. Furthermore, our method is the first INR-based video compression method to surpass the typically employed HEVC benchmark (x265, "medium" preset, disabled b-frames) [1, 19, 29] in R-D performance for the UVG dataset in both SSIM and PSNR. We also close the gap to modern autoencoder-based neural video compression methods, outperforming DVC across the whole bpp spectrum for both metrics. The results in the plot for NeRV, DVC, and Scale-Space Flow were gracefully provided by the authors.

A qualitative comparison can be seen in Figures 1 and 5. Results for HEVC were produced using ffmpeg [31] using the configurations described previously. Frames for NeRV were obtained using the published code. The code and trained models for Scale-Space Flow are not openly available, so frames for this model were obtained using the implementation from the CompressAI library, which is trained on the Vimeo90K dataset [6].

In all cases, our method produces visually sharper videos than NeRV for equal or smaller bitrates. Videos such as "ShakeNDry" often pose a problem to compression algorithms that rely on motion estimation, such as HEVC or Scale-Space Flow. In these scenarios, optical flow is a poor way to compress the video, as a large number particles are moving independently in different directions. Models which rely on this architectural prior introduce visible artifacts at high compression rates, and are outperformed by INR-based methods, which avoid this problem. On the other hand, for "YachtRide", motion-based descriptors compress the scene well, as most motion vectors are parallel. While our method clearly outperforms NeRV in this scene, HEVC shows the best performance, while SSF introduces some visible artifacts. Additional qualitative examples are provided in the supplementary material.

Encoding/Decoding time Tables 2 and 3 show the training and inference time for our method. We reinforce that the initial training time can be amortized over all the different compression levels. While these training times may seem too long, it is important to remember that most videos transmitted over the internet are encoded once and transmitted many times. For instance, in the case of large video hosting platforms, a video must only be compressed once, in an environment with access to high-performance computing machines, so that it can be stored in a server where it will remain for a long period of time. Given sufficient GPU RAM, it is also possible to reduce training time significantly by increasing the batch size.

As reported by Chen *et al*. [7], INR-based video compression methods show great advantages in decoding time compared to other models. Since the decoding process re-

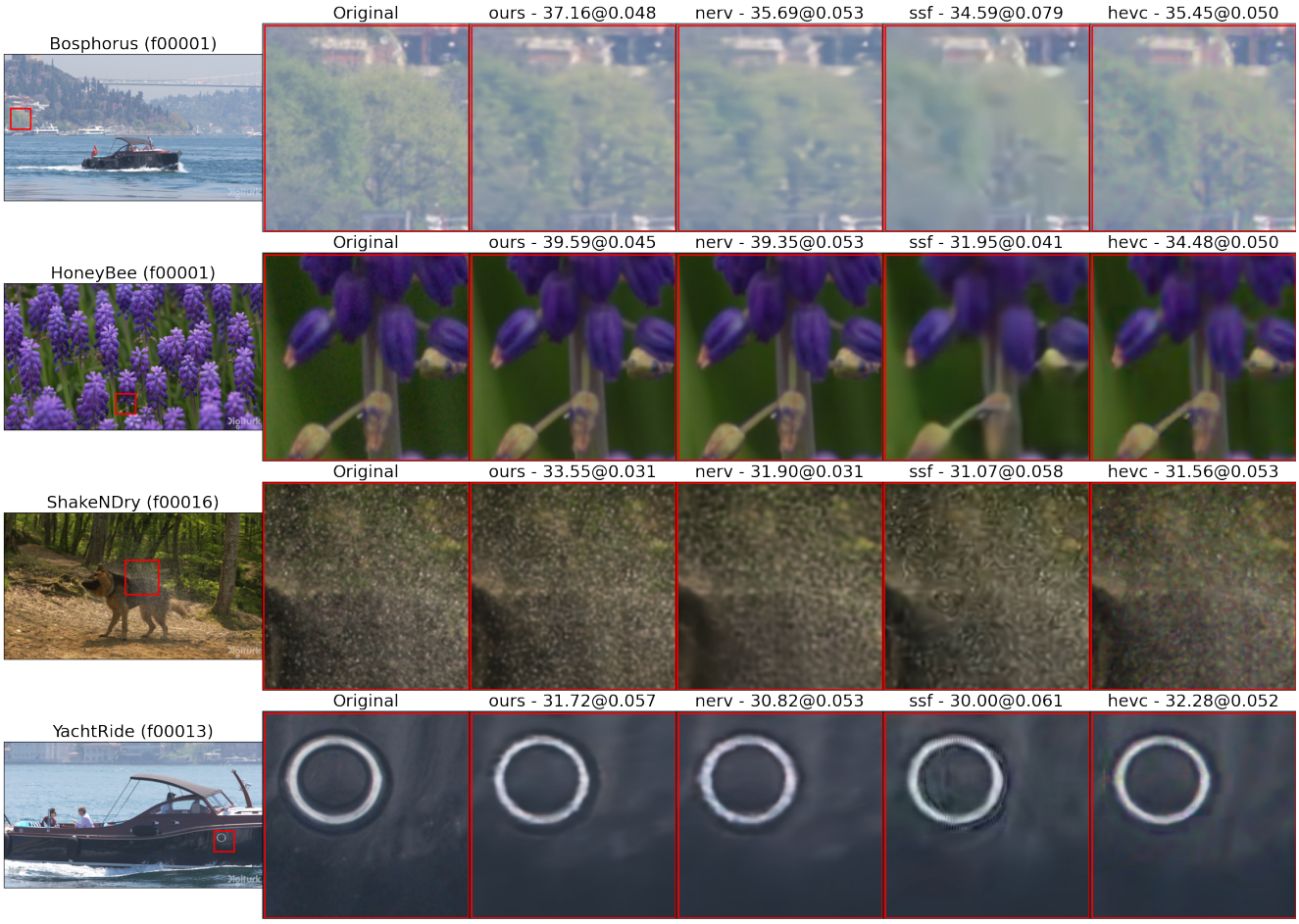

Figure 5. Qualitative comparison between our method, NeRV, SSF, and HEVC. Images are labeled with PSNR@bpp.

Architecture	Initial Training	Fine-tuning
Large	30h35m	14h09m
Small	14h55m	6h03m

Table 2. Training time for videos with 600 frames. It is important to note that only slightly inferior results can be obtained with drastically fewer training epochs. GPU: NVIDIA A6000.

Model	FPS (GPU)	FPS (CPU)
Ours (Large)	21.44	0.5
Ours (Small)	61.88	1.5
Scale-Space	1.25	0.15

Table 3. Decoding performance (1920x1080). Scale-Space using quality level 5/8 (from CompressAI [6]). GPU: RTX 3090

quires only a forward pass, and each frame can be computed independently, INR-based methods additionally offer good opportunities for parallelization when decoding.

5. Conclusion

We present a novel convolutional architecture for video representation capable of achieving higher fidelity encoding. We couple this with an end-to-end entropy-based neural network compression method to achieve video compression, bridging the gap to Neural Video Compression by formalizing the loss as a rate-distortion problem. We demonstrate the effectiveness of our method by testing on the UVG dataset, producing state-of-the-art results in video compression with INRs. While the presented results are promising, the adoption of neural representations for video compression as viable encoders still requires future research into less cost-prohibitive entropy-modeling methods in order to speed up encoding time, as well as further architecture improvements, seeking the optimal distribution of parameters in a compact network.

References

[1] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Balle, Sung Jin Hwang, and George Toderici. Scale-Space

Flow for End-to-End Optimized Video Compression. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8500–8509, Seattle, WA, USA, June 2020. IEEE. 2, 6, 7

[2] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Extreme learned image compression with gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2587–2590, 2018. 2

[3] Yunpeng Bai, Chao Dong, and Cairong Wang. Ps-nerv: Patch-wise stylized neural representations for videos, 2022. 2

[4] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end Optimized Image Compression, Mar. 2017. arXiv:1611.01704 [cs, math]. 2, 3, 5, 6

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *arXiv:1802.01436 [cs, eess, math]*, May 2018. arXiv: 1802.01436. 3, 6

[6] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. Compressai: a pytorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. 7, 8

[7] Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. NeRV: Neural Representations for Videos. *arXiv:2110.13903 [cs, eess]*, Oct. 2021. arXiv: 2110.13903. 2, 3, 6, 7

[8] Dandan Ding, Zhan Ma, Di Chen, Qingshuang Chen, Zoe Liu, and Fengqing Zhu. Advances in video compression system using deep neural network: A review and case studies. *Proceedings of the IEEE*, 109(9):1494–1520, 2021. 1

[9] Abdelaziz Djelouah, Joaquim Campos, Simone Schaub-Meyer, and Christopher Schroers. Neural Inter-Frame Compression for Video Coding. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6420–6428, Seoul, Korea (South), Oct. 2019. IEEE. 2

[10] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. COIN: COmpression with Implicit Neural representations. *arXiv:2103.03123 [cs, eess]*, Apr. 2021. arXiv: 2103.03123. 2, 3

[11] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. COIN++: Neural Compression Across Modalities, June 2022. arXiv:2201.12904 [cs, eess, stat]. 3

[12] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned Step Size Quantization. *arXiv:1902.08153 [cs, stat]*, May 2020. arXiv: 1902.08153. 5

[13] R.M. Gray and D.L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44(6):2325–2383, 1998. 5

[14] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 4

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[16] Zizhang Li, Mengmeng Wang, Huaijin Pi, Kechun Xu, Jianbiao Mei, and Yong Liu. E-nerv: Expedite neural video representation with disentangled spatial-temporal context, 2022. 2, 3, 4, 6

[17] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An End-to-end Deep Video Compression Framework, Apr. 2019. arXiv:1812.00101 [cs, eess]. 2, 6

[18] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and video compression with neural networks: A review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020. 1

[19] Fabian Mentzer, George Toderici, David Minnen, Sung-Jin Hwang, Sergi Caelles, Mario Lucic, and Eirikur Agustsson. VCT: A Video Compression Transformer, June 2022. Number: arXiv:2206.07307 arXiv:2206.07307 [cs, eess] version: 1. 2, 7

[20] Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[21] 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*, MMSys '20, page 297–302, New York, NY, USA, 2020. Association for Computing Machinery. 2

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *arXiv:2003.08934 [cs]*, Aug. 2020. arXiv: 2003.08934. 2, 3, 4

[23] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. Scalable Model Compression by Entropy Penalized Reparameterization. *arXiv:1906.06624 [cs, stat]*, Feb. 2020. arXiv: 1906.06624. 2, 3, 4, 5

[24] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. *arXiv:1901.05103 [cs]*, Jan. 2019. arXiv: 1901.05103. 2

[25] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019. 3

[26] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley Publishing, 2nd edition, 2010. 1

[27] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. 4

[28] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network, Sept. 2016. arXiv:1609.05158 [cs, stat] version: 2. 4

[29] Yannick Strümpler, Janis Postels, Ren Yang, Luc van Gool, and Federico Tombari. Implicit neural representations for image compression, 2021. 2, 3, 7

[30] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012. 1

[31] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006. 7

[32] Simon Wiedemann, Heiner Kirchhoffer, Stefan Matlage, Paul Haase, Arturo Marban, Talmaj Marinč, David Neumann, Tung Nguyen, Heiko Schwarz, Thomas Wiegand, Detlev Marpe, and Wojciech Samek. Deepcabac: A universal compression algorithm for deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 14(4):700–714, 2020. 6

[33] Yunfan Zhang, Ties van Rozendaal, Johann Brehmer, Markus Nagel, and Taco Cohen. Implicit Neural Video Compression. *arXiv:2112.11312 [cs]*, Dec. 2021. arXiv: 2112.11312. 2