NeRV: Neural Representations for Videos

Anonymous Author(s)
Affiliation
Address
email

Abstract

We propose a novel neural representation for videos (NeRV) which encodes videos in neural networks. Unlike conventional representations that treat videos as frame sequences, we represent videos as neural networks taking frame index as input. Given a frame index, NeRV outputs the corresponding RGB image. Video encoding in NeRV is simply fitting a neural network to all video frames and decoding process is a simple feedforward operation. With such a representation, we can treat videos as a neural networks, simplifying several video-related tasks. For example, conventional video compression methods are restricted by a long and complex pipeline, specifically designed for the task. In contrast, with NeRV, we can use any neural network compression method as a proxy for video compression, and achieve comparable performance to traditional frame-based video compression approaches. Besides compression, we demonstrate the generalization of NeRV for other tasks, such as video denoising and video temporal interpolation.

1 Introduction

What is a video? Typically, a video captures a dynamic scene using a sequence of frames. If we consider a frame as a still screenshot of the world as captured by the camera, then a video is a recording of the spatio-temporal state of the camera's world. A schematic interpretation of this is a curve in 2D space, where each point can be characterized with a (x, y) pair representing the spatial state. Figure 1(a) illustrates a line in 2D space, which can be seen as a set of points, where each x value has a corresponding y. Alternatively, a line is a recording of spatial locations, and if we have a model for all (x, y) pairs, then, given any x, we can easily find the corresponding y state. Similarly, we can interpret a video as a recording of the world, where we can find a corresponding RGB state for every single timestamp.

Now, instead of viewing a line as a sequence of points, we can also represent it as a function, $y = f_\theta(x)$, parameterized by a 2D vector $\theta = [m; b]$, in the 2D space; where we can use f_θ to readily compute the the corresponding y for any given x. This encoding considerably reduces the number of parameters needed to describe and store a line. Given the schematic similarities between lines and videos, this work investigates the following question: *can we represent a video as a function of time?*

More formally, can we represent a video V as $V = \{v_t\}_{t=1}^{T}$, where $v_t = f_\theta(t)$, *i.e.*, a frame at timestamp t, v_t, is represented as a function f parameterized by θ. Given their remarkable representational capacity [1], we use deep neural networks the function of choice in this work. Given these intuitions, we propose NeRV, a novel representation that represents videos as implicit functions and encodes them into neural networks. Specifically, with a fairly simple deep neural network, NeRV can reconstruct the corresponding video frames with high quality, given the frame index. Once the video is encoded into a neural network, this network can be used as a proxy for video, where we can directly extract all video information from the representation. Therefore, unlike traditional video representations which treat videos as sequences of frames, our proposed (NeRV) considers a video as a unified neural network with all information embedded within its architecture and parameters.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

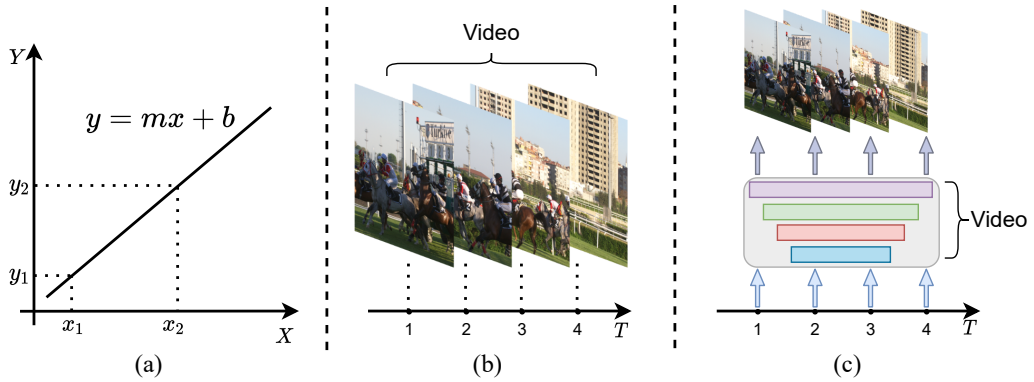


Figure 1: (a) Line in 2D space. (b) Conventional video representation as frame sequences. (c) The proposed **NeRV**, representing video as neural networks. The neural network comprises of different convolutional layers, which take the normalized frame index as the input and output the corresponding RGB frame.

We demonstrate the flexibility of NeRV by exploring several applications it affords. Most notably, we examine the suitability of NeRV for video compression. Traditional video compression frameworks are quite involved. In contrast, given a neural network that encodes a video in NeRV, we can simply cast video compression as a model compression problem, and trivially leverage any well-established or cutting edge model compression algorithm to achieve good compression ratios. In addition, we also explore out-of-the-box application of NeRV to video denoising and temporal interpolation.

Conventional video compression methods (*e.g.*, H.264 [2], HEVC [3]), involve long and burdensome processes, such as specifying key frames and inter frames, estimating the residual information, block-size the video frames, applying discrete cosine transform on the resulting image blocks, entropy coding, and so on. Even with the recently proposed learning-based compression methods [4–6], which use neural networks to improve the encoding/decoding efficiency and quality, the overall framework remains similarly complex. In contrast, in this paper, encoding is training a neural network and decoding is reduced to simply a feedforward inference, allowing us to pose the video compression task as model compression. Note that with this formulation, any advancement in model compression can be readily used for video compression, without any bells and whistles. We conduct extensive experiments on popular video compression datasets, such as UVG [7] and MCL-JCV [8], and show the applicability of model compression techniques on NeRV for video compression. Specifically, we explore a three step model compression pipeline: model pruning, weight quantization, and entropy encoding, and show that each step contributes to the compression task.

Besides video compression, we also explore other out-of-the-box applications of the NeRV representation for video denoising and temporal interpolation. Since NeRV is a learnt implicit function, we can demonstrate its robustness to noise and perturbations. Given a noisy video as input, NeRV generates a high-quality denoised output, without any additional operation, and even outperforms conventional denoising methods. Since NeRV uses a continuous function space for time, we also investigate generating unseen frames when using new timestamps as inputs, observe that it can give reasonable predictions.

The contribution of this paper can be summarized into three parts:

- We propose NeRV, a novel representation for videos, which represents a video as a neural network, converting video encoding to a model fitting process and the decoding to a feedforward operation.
- NeRV allows us to convert the video compression problem to a model compression problem, allowing us to leverage standard model compression tools and reach comparable performance with conventional video compression methods, *e.g.*, H264, HEVC.
- As a general representation for videos, NeRV also shows promising results in other tasks, *e.g.*, video temporal interpolation, video denoising.

2 Related Work

Implicit Neural Representation. Implicit neural representation is a novel way to parameterize a variety of signals. The key idea is to represent an object as a function approximated via a neural network, which maps the coordinate to its corresponding value (*e.g.*, pixel coordinate for an image and RGB value of the pixel). It has been widely applied in many 3D vision tasks, such as 3D shapes [9, 10], 3D scenes [11–14], and appearance of the 3D structure [15–17]. Comparing to explicit 3D representations, such as voxel, point cloud, and mesh, the continuous implicit neural representation can compactly encode high-resolution signals in a memory-efficient way. Most recently, [18] demonstrated the feasibility of using implicit neural representation for image compression tasks. Although it is not yet competitive with the state-of-the-art compression methods, it shows promising and attractive proprieties.

In previous methods, MLPs are often used to approximate the implicit neural representations, which take the spatial or spatio-temporal coordinate as the input and output the signals at that single point (*e.g.*, RGB value, volume density). However, different from that, our proposed NeRV representation, trains a purposefully designed neural network architecture composed of MLPs and convolution layers, and takes the temporal coordinate of a video frame as the input and directly outputs all the RGB values of that frame.

Video compression. As a fundamental task of computer vision and image processing, visual data compression has been studied for several decades. Before the resurgence of deep networks, hand-crafted image compression techniques, like JPEG [19] and JPEG2000 [20], were widely used. Building upon them, many traditional video compression algorithms, such as MPEG [21], H.264 [2], and HEVC [3], have achieved great success. These methods are generally based on transform coding like Discrete Cosine Transform (DCT) [22] or wavelet transform [23], which are well-engineered and tuned to be fast and efficient.

More recently, deep learning-based visual compression approaches have been gaining popularity. For video compression, the most common practice is to utilize neural networks for certain components while using the traditional video compression pipeline. For example, [4] proposed an effective image compression approach and generalized it into video compression by adding interpolation loop modules. Similarly, [24] converted the video compression problem into an image interpolation problem and proposed an interpolation network, resulting in competitive compression quality. Furthermore, [5] generalized optical flow to scale-space flow to better model uncertainty in compression. Later, [6] employed a temporal hierarchical structure, and trained neural networks for most components including key frame compression, motion estimation, motions compression, and residual compression. However, all of these works still follow the overall pipeline of traditional compression, arguably limiting their capabilities.

Model compression. The goal of model compression is to simplify an original model by reducing the number of parameters while maintaining its accuracy. Current research on model compression research can be divided into four groups: parameter pruning and quantization [25–30]; low-rank factorization [31–33]; transferred and compact convolutional filters [34–37]; and knowledge distillation [38–41]. Our proposed NeRV enables us to reformulate the video compression problem into model compression, and utilize standard model compression techniques. Specifically, we use parameter pruning and quantization methods to reduce the model size without significantly deteriorating the performance. Further improvements can be made by adopting more advanced or complicated model compression methods.

3 NeRV: Neural Representations for Videos

We first present the NeRV representation in Section 3.1, including the input, the network architecture, and the training strategy. Then, we present model compression techniques on NeRV in Section 3.2 for video compression.

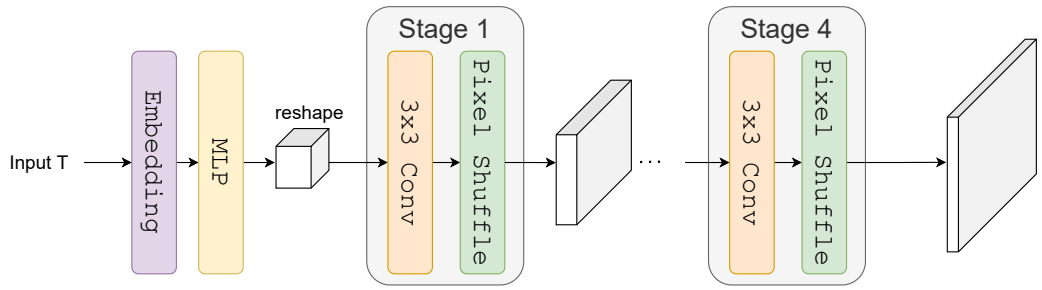


Figure 2: Model Architecture

Table 1: Upscale block architecture

Ind	Modules	Output
1	Convolution	$s^2c \times h \times w$
2	PixelShuffle	$c \times sh \times sw$
3	LeaklyReLU	$c \times sh \times sw$

Table 2: NeRV model architecture. MLP parameters means the hidden layer dimentions, upscale block parameters means the upscale factor used in pixelshuffel

Ind	Modules	Module parameters	Input	Output
1	MLP	(320, 1024, 18432)	(320)	(128, 16, 9)
2	Upscale block	5×	(128, 16, 9)	(512, 80, 45)
3	Upscale block	3×	(512, 80, 45)	(256, 240, 135)
4	Upscale block	2×	(256, 240, 135)	(128, 480, 270)
5	Upscale block	2×	(128, 480, 270)	(64, 960, 540)
6	Upscale block	2×	(64, 960, 540)	(32, 1920, 1080)
7	Convolution	(3, 128, 3, 3)	(32, 1920, 1080)	(3, 1920, 1080)

3.1 NeRV

In NeRV, each video $V = \{v_t\}_{t=1}^{T} \in R^{T \times H \times W \times 3}$ is represented by a function $f_\theta : \mathbb{R} \rightarrow \mathbb{R}^{H \times W \times 3}$, where the input is a frame index t and the output is the corresponding RGB image $v_t \in R^{H \times W \times 3}$. The encoding function is parameterized with a deep neural network (θ), as

$$v_t = f_\theta(t) \tag{1}$$

Therefore, video encoding is done by fitting a neural network f_θ to a given video, such that it can map each input timestamp to the corresponding RGB image.

Input Embedding. Although deep neural networks can be used as universal function approximators [1], directly training the network f_θ with input timestamp t results in poor results, which is also observed by [15, 42]. By mapping the inputs to a high embedding space, the neural network can better fit data with high-frequency variations. Specifically, in NeRV, we use fourier features [43] as our embedding function to encode the input timestamp as

$$\Gamma(t) = (\sin(b^0 \pi t), \cos(b^0 \pi t), ..., \sin(b^{l-1} \pi t), \cos(b^{l-1} \pi t)) \tag{2}$$

where b and l are hyper-parameters of the networks. In our experiments, we use $b = 1.25$ and $l = 160$ as our default setting. More ablation results for b, l are provided in the supplementary material. Given an input timestamp t, normalized between $(0, 1]$, the output of embedding function $\Gamma(\cdot)$ is fed to the neural network.

NeRV Model Architecture. The overall architecture of the NeRV model is illustrated in Figure 2. It takes the time embedding as input and outputs the corresponding RGB image. Leveraging MLPs to directly output the RGB frames can lead to burdensome parameters. Therefore, we stack convolution layers on the MLP base network so that pixels at different locations can share operations and parameters, leading to an efficient network. Similar to super-resolution networks, we design an upscale block using pixel-shuffle [44] for upscaling, illustrated in Table 1. Our NeRV model, illustrated in Table 2, is built by stacking MLP and upscale blocks, and a final 3×3 convolution layer is used for RGB prediction.

Loss Objective. For NeRV, we adopt L2 loss as the loss function for network optimization, which calculates the average of mean squared error over all pixel locations of the predicted image and the

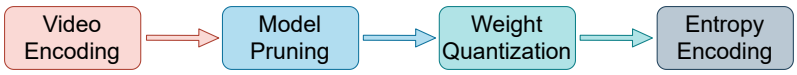


Figure 3: NeRV-based video compression pipeline

Table 3: Comparison of **single** vs. **multiple**. For multiple, we encode multiple videos into one model.

	Video index	1	2	3	4	5	6	7	Mean
Single	PSNR	33.56	34.42	37.92	34.30	28.10	33.22	30.03	33.07
	MS-SSIM	0.896	0.947	0.981	0.927	0.905	0.942	0.909	0.929
Multiple	PSNR	33.89	36.15	37.96	36.96	34.43	34.80	34.53	35.59
	MS-SSIM	0.900	0.959	0.981	0.956	0.971	0.953	0.959	0.954

ground-truth image as

$$L = \frac{1}{n} \sum_{t,x,y} \| f_\theta(t)[x, y] - v_t[x, y] \|^2 \tag{3}$$

where n is the total number of pixels. We also explored other losses (*e.g.*, L1 loss and SSIM loss [45]) and provide those results in the supplementary.

3.2 Model Compression

In this section, we briefly revisit model compression techniques that we use for NeRV to achieve video compression. Our model compression composes of four standard sequential steps: model training, model pruning, weight quantization, and entropy encoding as shown in Figure 4.

Model Pruning. Given a neural network that's fit on a video, we use global unstructured pruning to reduce the model size first. Based on the magnitude of weight values, we set weights below a threshold as zero. After the pruning operation, we fine-tune the model to regain the representation. Through validation experiments, we find that 15%-25% pruning ratio achieves the best trade-off between the PSNR and model size.

Weight Quantization. After model pruning, we apply weight quantization to all neural network parameters. Note that different from many recent works [29, 46–48] that utilize quantization during training, NeRV is only quantized post-hoc (after the training process). Given a FP32 parameter weight for NeRV, we linearly map every channel vector to the closest integer between [0,255]. Finally, we store all these rounded UINT8 tensors together with the scale factors and offsets of the linear mappings. In this step, INT8 quantization can reduce the model size by nearly 75% with minimal loss in performance.

Entropy Encoding. The last step of the model compression we apply to NeRV is entropy encoding. By taking advantage of character frequency, entropy encoding can represent the data with a more efficient codec. Specifically, we employ Huffman Coding [49] after model quantization. Since Huffman Coding is lossless, it is guaranteed that a decent compression can be achieved without any impact on the reconstruction quality. Empirically, the entropy encoding can further reduce the model size by 10%.

4 Experiment

4.1 Datasets and Implementation Details

Datasets. We perform experiments on the UVG [7] dataset and MCL-JCV [8] dataset. These dataset are widely used for video compression evaluation, and contains 7 and 30 videos of 1920x1080 resolution and total number of 3900 and 4115 frames respectively.

Implementation details. In our experiments, we train the network using Adam optimizer [50] with learning rate of 5e-4. We use cosine annealing learning rate schedule [51], batchsize of 1, training

Table 4: Upscale layer ablation

	PSNR	MS-SSIM
Bilinear Pooling	27.90	0.8207
Transpose Conv	31.29	0.8924
PixelShuffle	35.59	0.9542

Table 5: Norm layer ablation

	PSNR	MS-SSIM
BatchNorm	16.90	0.8276
InstanceNorm	33.83	0.9387
None	35.59	0.9542

Table 6: Activation ablation

	PSNR	MS-SSIM
ReLU	35.10	0.9504
Leaky ReLU	35.59	0.9542

Table 7: Impact of input embedding

	PSNR	MS-SSIM
None	23.41	0.7039
Fourier Feature	35.59	0.9542

Table 8: Impact of batchsize and training iterations

Batchsize	Iter	PSNR	MS-SSIM
1	$\times 1$	35.59	0.954
1	$\times 2$	36.48	0.960
1	$\times 4$	37.13	0.965
2	$\times 1$	36.28	0.960
4	$\times 1$	36.89	0.964

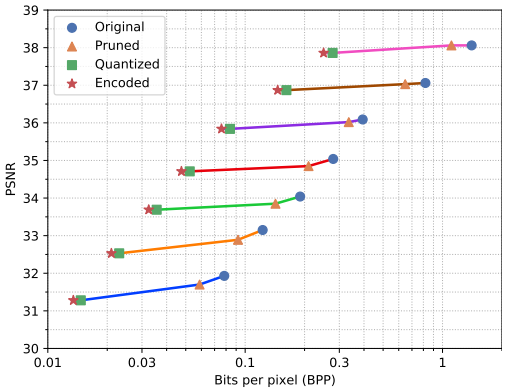


Figure 4: Impact of model compression stages. Each line represents models with different sizes. We show the relationship of PSNR and BPP for each compression steps. Given a trained model (original) and applying pruning, quatization and encoding, it reaches the best PSNR and BPP trad-off finally. Note that these steps are conducted in an accumulating manner.

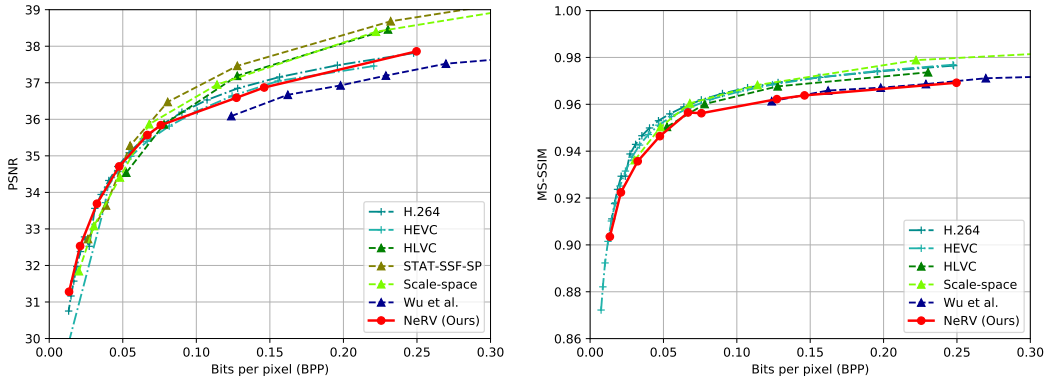
epochs of 150, and warmup epochs of 30 unless otherwise denoted. For the ablation study, we train models on the sub-sampled frames with sampling stride 4 to expedite the training process. There are 5 upscale stages which expand spatial dimension of the feature map by 5, 3, 2, 2, and 2 respectively. By changing the output dimension of MLP output and filter width of upscale blocks, we can build NeRV model with different sizes. We implement our model in PyTorch [52] and train in FP32. Please refer to the supplementary material for more experimental details.

4.2 Video compression.

We evaluate our model on the video compression task and report the performances of two metrics: PSNR and MS-SSIM [53]. Bits-per-pixel (BPP) is adopted to indicate the compression ratio. The process of video compression is as follows. First, we concatenate all the videos into one along the time dimension and training the model on all the frames for each dataset, which we found to be more beneficial than training a single model for each video (see Table 9). After training the network, we apply model pruning, quantization, and entropy encoding as we discussed in Section 3.2.

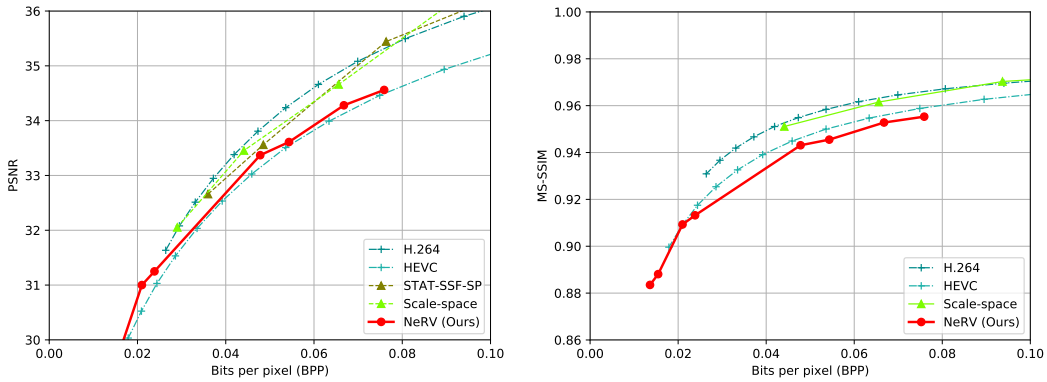
To present how these model compression techniques affect the NeRV's BPP and PSNR, we plot the rate distortion curves for multiple NeRV models with different sizes under different compression steps in Figure 4. Specifically, each plot with the same color is the model the same size. "Original" represent the model after training without employing any compression techniques. It has the highest PSNR performance with the largest BPP value. We can see that the three compression steps on different NeRV models have the similar impact. By applying the pruning, quatization, entropy encoding techniques sequentially, BPP decreases dramatically without deteriorating PSNR much. Among three compression mechanisms, quatization achieves the most significant BPP reduction.

Interestingly, we note that the pruning is the only step that impacts the reconstruction quality and is relevant to the model size of NeRV. The model with larger size tends to have less performance drop due to pruning. This observation suggests that larger models have more redundancy in parameters, and thus can have higher pruning ratios.



(a) PSNR vs. BPP (b) MS-SSIM vs. BPP

Figure 5: Rate distortion plots on the UVG dataset.



(a) PSNR vs. BPP (b) MS-SSIM vs. BPP

Figure 6: Rate distortion plots on the MCL-JCV dataset.

Comparing with Other Video Compression Methods. Figure 5 and Figure 6 show the rate-distortion curves on the UVG and the MCL-JCV datasets. We choose to compare NeRV with H.264 [2], HEVC [3], STAT-SSF-SP [54], HLVC [6], Scale-space [5], and Wu *et al*. [24]. We report both PSNR and MS-SSIM results of these approaches if they are available. H.264 and HEVC are performed with FFMPEG with *medium* preset mode. As the first neural network representation of videos, NeRV generally achieves comparable performance with traditional video compression techniques and other learning-based video compression approaches that employ an encoder-decoder architecture. It worth noting that when BPP is small, NeRV can match the performance of the state-of-the-art method, showing its great potential in high-rate video compression. When BPP becomes large, NeRV's performance drops. But we argue that this is mostly because of the lack of fine-tuning. Due to the limited computing resources we could access, we cannot run large-batch fine-tuning on NeRV models with large size, which has proved to be very helpful for NeRV's performance (see Section 4.3 and Table 8). Therefore, we posit that there is still a lot of room for performance improvement of larger NeRV models.

4.3 Ablation Studies

We conduct all the ablation studies on the UVG dataset. PSNR and MS-SSIM are adopted for evaluation of the reconstructed videos.

Single *vs*. Multiple Video Representations. In Table 9, we compare the performance under two settings: (1) training separate models for each video (single), and (2) training one model to encode multiple videos (multiple). For fair comparison, we adjust the model size to ensure both settings have similar bpp ratios. We first observe that with the identical model architecture and training strategy, the reconstruction performances for each video varies a lot, which is as expected because different videos contain distinct amount of information even if with same duration. Encoding a video is more

Table 9: PSNR results for video denoising. "baseline" refers to the noisy frames before any denoising. See the supplementary for the detailed settings and more visualizations.

noise	white	black	salt&pepper	random (0.3)	random (1.0)	Gaussian
baseline	27.85	28.29	27.95	39.07	30.95	43.53
Gaussian	30.27	30.14	30.23	31.31	30.99	31.35
Uniform	29.11	29.06	29.10	29.82	29.63	29.85
Median	**33.89**	33.84	33.87	33.90	33.89	33.91
Minimum	20.55	16.60	18.09	20.05	18.20	20.35
Maximum	16.16	20.26	17.69	19.79	17.83	20.08
NeRV	33.31	**34.20**	**34.17**	**34.90**	**34.80**	**34.96**

difficult when it contains more information. For example, a video with dynamic frames tends to have much richer information than videos with relatively less movements between frames. Second, when fitting multiple videos into one shared model, the performances of each video is better than training them separately. We conjecture that the convolutional layer can learn better local structures when learning with more data. Specifically, the average PSNR and MS-SSIM results for the multiple setting can outperform the single setting by 2.52 and 0.025 respectively.

Model Architecture Design Ablation. In Table 4, we investigate the effectiveness of different upscale layers adopted in NeRV. We observe that PixelShuffle [44] outperforms the Biliear pooling and Transpose Convolution [55] by a large margin. In addition, in Table 5, we analyse the impact of different normalization layer, we find it interesting that the model without any normalization layer achieves the best result, because the normalization layer reduces the over-fitting capability of the neural network, which is contradictory to our training objective. In Table 6, we also test on different activation functions, and find that leaky ReLU [56] yields higher PSNR and MS-SSIM results than ReLU. We also tested on different embedding types for the input timestamp in Table 7. As we demonstrated earlier, without any input embedding, the model can not learn high-frequency information, resulting in much lower performance.

Batchsize and Training Iterations. Interestingly, as shown in Table 8, by increasing the batchsize and training iterations, the final results can improve consistently over the baseline. As the training objective of NeRV is to overfit on the input video, longer training schedule can result in lower training error and better fit the input video.

4.4 Video Denoising.

We apply several common noise patterns on the original video frames and train the model on the perturbed videos. During training, no masks or locations of noises are provided to the model, *i.e.*, the target of the model is the noisy frames while the model has no extra signal of whether the input is noisy or not. Surprisingly, our model tries to avoid the influence of the noise and regularizes them implicitly with little harm to the compression task simultaneously, which can serve well for most partially distorted videos in practice.

The results are compared with some standard denoising methods including Gaussian, uniform, and median filtering. These can be viewed as the upper bound of any additional compression process. As listed in Table 9, the PSNR of our model's output is usually much higher than the noisy frames although it's trained on the noisy target in fully supervised manner, and has reached an acceptable level for general denoising purpose. Specifically, median filtering has the best performance among the traditional denoising techniques, while our model also outperforms it in most cases or is at least comparable without any extra modules for denoising.

Moreover, while the traditional filters have to be applied through the whole frames, our model learns to concentrate on and fixed the noisy areas, thus avoid changing the entire image. More qualitative comparisons are provided in the supplementary. This implies that our model has learned spatial and temporal relations among neighboring pixels and across consecutive frames, thus preserving reasonable relationships implicitly.

(a) Seen Frame T (b) Interpolated Unseen Frame T+0.5 (c) Seen Frame T+1

Figure 7: Video temporal interpolation results.

4.5 Video Temporal Interpolation

We also explore NeRV for video temporal interpolation task. Specifically, we train our model with a subset of frames sampled from one video, and then use the trained model to infer/predict unseen frames given an unseen interpolated frame index. The video in Figure 7 shows a dog drying itself by shaking its body. Interestingly, compared to the seen frame with time index T and $T+1$, the predicted interpolated unseen frame at time index $T+0.5$, our model can yield the motionless background perfectly with PSNR result of 30.55, however, for the motion area (inside the red rectangle), our model can not make reasonable predictions with PSNR result of 20.33. Improving the architecture can potentially address this issue for interpolation, but is beyond the scope of this work.

5 Discussion

Conclusion. In this work, we present a novel neural representation for videos, NeRV, which encodes videos into neural networks. Our key sight is that by directly training a neural network with video frame index and corresponding RGB image, we can store the weights of the model to represent the videos, which is totally different from conventional representations that treat videos as frame sequences. With such a representation, we show that by simply applying general model compression techniques, our simple representation can match the performances of traditional video compression approaches for the video compression task, without the need to design a long and complex pipeline. We also show that NeRV can outperform standard denoising methods. We hope that this paper can inspire further research works to design novel class of methods for video representations.

Limitations and Future Work. There are some limitations with the proposed NeRV. First, to achieve the comparable PSNR and MS-SSIM performances, the training time of our proposed approach is much longer than the encoding time of traditional video compression methods, 4-5 days *vs*. several minutes. Second, for the video temporal interpolation task, although NeRV is good at predicting the still background, the prediction for the motion area is still far from satisfactory. Third, the architecture design of NeRV is still not optimal, more exploration on the neural architecture might achieve higher performances. For the future work, it will be ideal if we can expedite the training time and enhance the video temporal interpolation performances by designing new neural model architecture and applying other training strategies.

Broader Impact. As the most popular media format nowadays, video can be seen as pixels spreading in spatial and temporal dimensions. Our proposed NeRV is a novel way to represent videos as function of time, which is more efficient and might be used in many video-related tasks, such as video compression, video denoising, and video temporal interpolation and so on. Hopefully, this can potentially save bandwidth, while enriching entertainment potential. Unfortunately, like many advances in deep learning for videos, this approach can be utilized for a variety of purposes beyond our control.

Reproducibility. Code and final models to reproduce the results in our paper will be made public.

References

[1] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 1, 4

[2] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003. 2, 3, 7

[3] 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. 2, 3, 7

[4] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learning image and video compression through spatial-temporal energy compaction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10071–10080, 2019. 2, 3

[5] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8503–8512, 2020. 3, 7

[6] Ren Yang, Fabian Mentzer, Luc Van Gool, and Radu Timofte. Learning for video compression with hierarchical quality and recurrent enhancement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6628–6637, 2020. 2, 3, 7

[7] 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*, pages 297–302, 2020. 2, 5

[8] Haiqiang Wang, Weihao Gan, Sudeng Hu, Joe Yuchieh Lin, Lina Jin, Longguang Song, Ping Wang, Ioannis Katsavounidis, Anne Aaron, and C-C Jay Kuo. Mcl-jcv: a jnd-based h. 264/avc video quality assessment dataset. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1509–1513. IEEE, 2016. 2, 5

[9] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7154–7164, 2019. 3

[10] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas A Funkhouser. Deep structured implicit functions. 2019. 3

[11] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *arXiv preprint arXiv:1906.01618*, 2019. 3

[12] Chiyu Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, Thomas Funkhouser, et al. Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010, 2020.

[13] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. *arXiv preprint arXiv:2003.04618*, 2, 2020.

[14] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *European Conference on Computer Vision*, pages 608–625. Springer, 2020. 3

[15] 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. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020. 3, 4

[16] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[17] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540, 2019. 3

[18] Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 3

[19] Gregory K Wallace. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992. 3

[20] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001. 3

[21] Didier Le Gall. Mpeg: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991. 3

[22] Nasir Ahmed, T_ Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE transactions on Computers*, 100(1):90–93, 1974. 3

[23] Marc Antonini, Michel Barlaud, Pierre Mathieu, and Ingrid Daubechies. Image coding using wavelet transform. *IEEE Transactions on image processing*, 1(2):205–220, 1992. 3

[24] Chao-Yuan Wu, Nayan Singhal, and Philipp Krahenbuhl. Video compression through image interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 416–431, 2018. 3, 7

[25] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. 2011. 3

[26] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.

[27] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[28] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *arXiv preprint arXiv:1608.03665*, 2016.

[29] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 5

[30] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. 3

[31] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2754–2761, 2013. 3

[32] Emily Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *arXiv preprint arXiv:1404.0736*, 2014.

[33] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 3

[34] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016. 3

[35] Shuangfei Zhai, Yu Cheng, Weining Lu, and Zhongfei Zhang. Doubly convolutional neural networks. *arXiv preprint arXiv:1610.09716*, 2016.

[36] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225. PMLR, 2016.

[37] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. In *International conference on machine learning*, pages 1889–1898. PMLR, 2016. 3

[38] Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013. 3

[39] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[40] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 742–751, 2017.

[41] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018. 3

[42] 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 *International Conference on Machine Learning*, pages 5301–5310. PMLR, 2019. 4

[43] Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020. 4

[44] 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. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1874–1883, 2016. 4, 8

[45] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 5

[46] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. Scalable methods for 8-bit training of neural networks. *arXiv preprint arXiv:1805.11046*, 2018. 5

[47] Fartash Faghri, Iman Tabrizian, Ilia Markov, Dan Alistarh, Daniel Roy, and Ali Ramezani-Kebrya. Adaptive gradient quantization for data-parallel sgd. *arXiv preprint arXiv:2010.12460*, 2020.

[48] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *arXiv preprint arXiv:1812.08011*, 2018. 5

[49] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. 5

[50] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[51] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 5

[52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 6

[53] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003. 6

[54] Ruihan Yang, Yibo Yang, Joseph Marino, and Stephan Mandt. Hierarchical autoregressive modeling for neural video compression. *arXiv preprint arXiv:2010.10258*, 2020. 7

[55] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 8

[56] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. 8

A Appendix

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 4.
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4.1.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Given the resource constraint, we are unable to provide the error bars for all possible experiments, however, we did test the variance of some experiment settings, which can be negligible.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]