HNeRV: A Hybrid Neural Representation for Videos

Supplementary Material

A. Ablation study

We show the effectiveness of even-distributed parameters in Table 9 and Table 10 by increasing kernel size and channel width of later layers. For the NeRV block, it uses fixed $K = 3$, and channel reduction factor $r = 2$. We also show an embeddings ablation study, for spatial size ($h \times w$) in Table 11 and embedding dimensions (d) in Table 12.

Table 9. **Kernel size** (K_{min}, K_{max}) ablation, (with r=1.2)

K	PSNR	MS-SSIM
1,3	35.02	0.9752
1,5	**35.57**	**0.9773**
1,7	35.07	0.9757
3,3	33.09	0.9587

Table 10. **Channel reduction** r ablation, (with K=1,5)

r	PSNR	MS-SSIM
1	34.96	0.9745
1.2	**35.57**	**0.9773**
1.5	34.98	0.9762
2	34.32	0.9715

Table 11. **Embedding spatial size** ablation

$h \times w$	PSNR	MS-SSIM
1×2	34.79	0.9735
2×4	**35.57**	**0.9773**
4×8	35.12	0.9761

Table 12. **Embedding dimension** ablation

d	PSNR	MS-SSIM
8	35.13	0.9770
16	**35.57**	**0.9773**
32	35.08	0.9758

B. Video decoding

We firstly show command to evaluate decoding speed of H.264 and H.265:

```
ffmpeg -threads ThreadsNum -i Video -preset medium -f null -benchmark -
```

And we also show quantitative decoding results in Table 13, 14, and Table 15. In Table 15, we can further increase video decoding speed with a smaller channel width (*i.e.* a big reduction factor $r = 2$).

C. Video compression

Then we show the details for downstream tasks of video compression, which can be divided into three steps: global unstructure pruning, quantization, and entropy encoding.

1) Model Pruning. Given a pre-trained model, we use global unstructured pruning to reduce the model size, where parameters below a threshold are pruned and set as zero. For a model parameter θ_i, $\theta_i = \begin{cases} \theta_i, & \text{if } \theta_i \geq \theta_q \\ 0, & \text{otherwise,} \end{cases}$ where θ_q is the q percentile value for all model parameters θ. As a normal practice, we fine-tune the model to regain the representation after pruning.

Table 13. Decoding FPS ↑

PSNR	32	35	37
H.264	279.7	240.9	192.7
H.265	211.9	163.2	132.5
DCVC	4.7	4.6	4.5
HNeRV	**395.9**	**332.7**	**224.8**

Table 14. Decoding time (s) ↓

# Frames	100%	50%	25%
H.264	0.548	0.480	0.343
H.265	0.809	0.708	0.506
DCVC	27.913	24.424	17.446
HNeRV	**0.397**	**0.198**	**0.099**

Table 15. HNeRV Decoding FPS

PSNR	32	35	37
r=1.5	395.9	332.7	224.8
r=1.75	397.4	373.8	320.7
r=2	**405.5**	**383.3**	**350.5**

2) Model and embedding quantization. Model quantization and embedding quantization follow the same scheme. Given an vector μ, we linearly map every element to the closest integer,

$$\mu_i = \text{Round}\left(\frac{\mu_i - \mu_{\text{min}}}{\text{scale}}\right) * \text{scale} + \mu_{\text{min}}, \text{where}$$
$$\text{scale} = \frac{\mu_{\text{max}} - \mu_{\text{min}}}{2^{\text{b}} - 1} \tag{3}$$

μ_i is one vector element, 'Round' is a function that rounds to the closest integer, 'b' is the bit length for quantization, μ_{max} and μ_{min} are the max and min value of vector μ, and 'scale' is the scaling factor. For scaling factor and zero points at this step, we can also try other methods instead of current min-max one, like choosing 2^b evenly-distributed values to minimum the mean square error.

3) Entropy encoding. Finally, we use entropy encoding to further reduce the size. Specifically, we leverage Huffman coding [15] for quantized weights and get lossless compression.

D. Weight Pruning for Model Compression.

We appreciate this concern, which has been unresolved since the original NeRV paper. By applying entropy encoding (assigning fewer bits for frequent symbols), we can store pruned weights with limited bits, since all pruned weights share a frequent symbol: 0. We provide corrected model compression results in Tab. 16, and will update the paper accordingly. We use 2 baselines (models with no pruning) – one where the model is only quantized, and another where we apply entropy encoding after quantization. As we prune more parameters, entropy encoding enables us to use fewer bits to store the sparse model weights.

Table 16. Compression results. "Size ratio" compares to model with quant. only, and "Sparsity" indicates amount of weights pruned.

Compression	Quant	Prune + Quant + Entropy coding				
Sparsity	0%	0%	10%	15%	20%	25%
PSNR	37.61	37.56	37.51	37.32	37.02	36.61
Size (bits)	11.54M	10.94M	10.41M	10.09M	9.77M	9.36M
Size ratio	100%	94.8%	90.2%	87.4%	84.7%	81.1%

E. HNeRV architecture details

We also provide architecture details for HNeRV models in various tasks and datasets in Table 17, with total size, strides list, encoder dimension $c1$, embedding dimension d, channel width of decoder input $c2$, channel reduction r, lowest channel width Ch_{min}, min and max kernel size K_{min}, K_{max} .

Table 17. HNeRV architecture details

Video size	size	strides	c1	d	c2	r	Ch_{min}	K_{min}, K_{max}
640×1280	0.35	5,4,4,2,2	64	16	32	1.2	12	1,5
640×1280	0.75	5,4,4,2,2	64	16	48	1.2	12	1,5
640×1280	1.5	5,4,4,2,2	64	16	68	1.2	12	1,5
640×1280	3	5,4,4,2,2	64	16	97	1.2	12	1,5
480×960	3	5,4,3,2,2	64	16	110	1.2	12	1,5
960×1920	3	5,4,4,3,2	64	16	92	1.2	12	1,5

F. Per-video compression results

We also show video compression results for **UVG** videos in Figure 10.

G. More visualizations

We show more visualizations for video regression (Figure 11), video interpolation (Figure 12), and video inpainting (Figure 13).

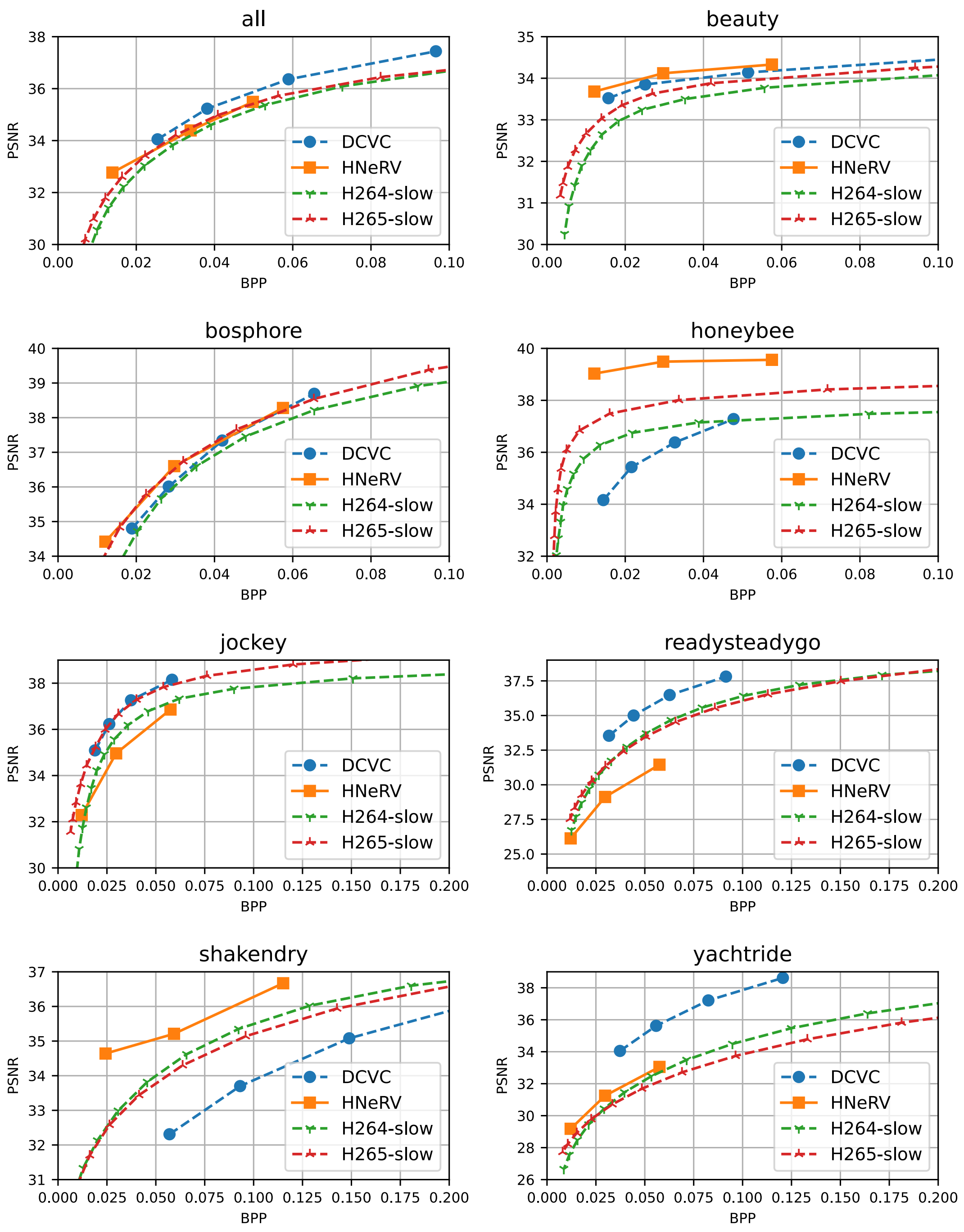

Figure 10. **Compression** results averaged across all **UVG** videos, and for each specific videos.

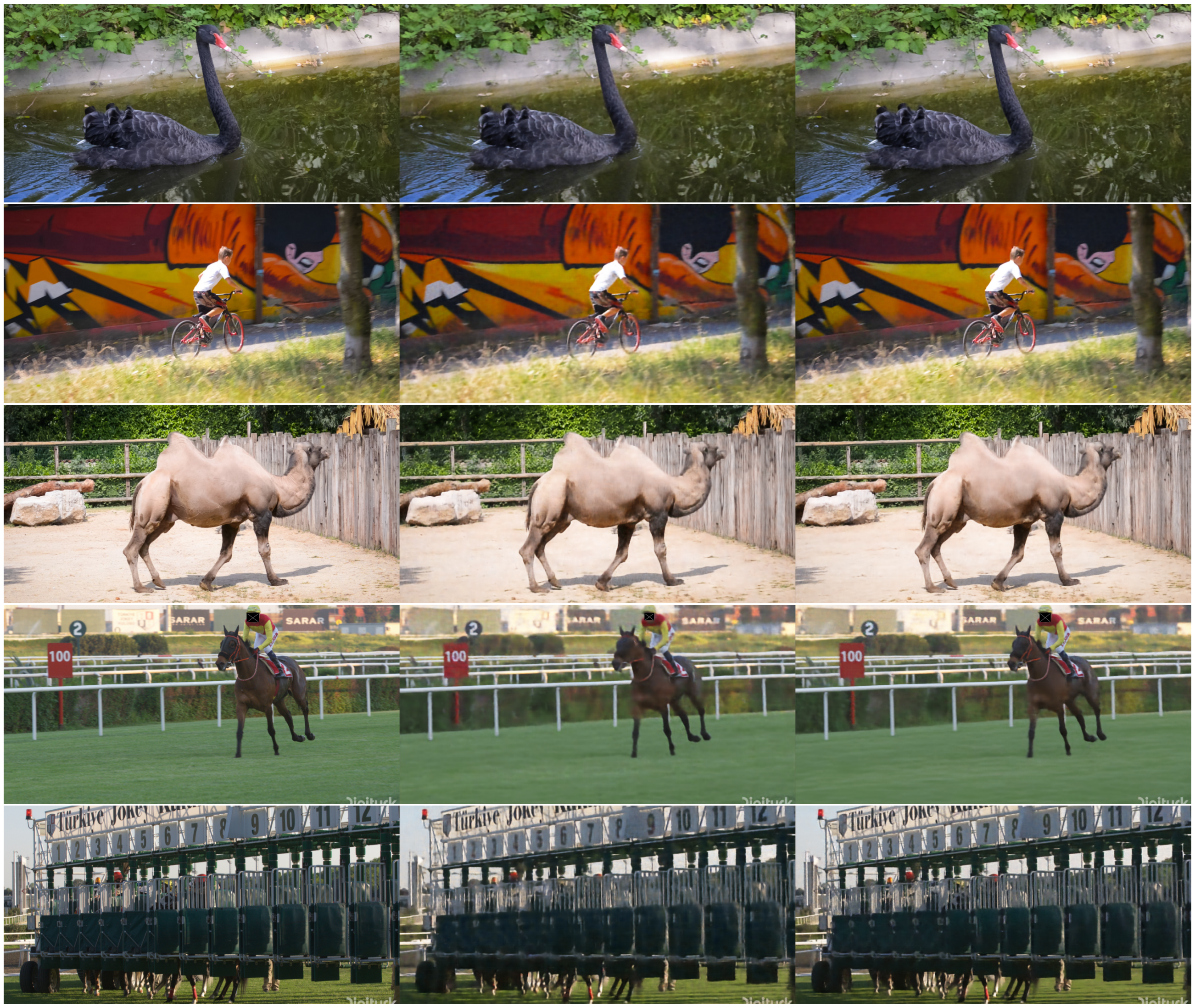
Figure 11. **Video regression** results. **Left)** ground truth. **Middle)** NeRV output. **Right)** HNeRV output.

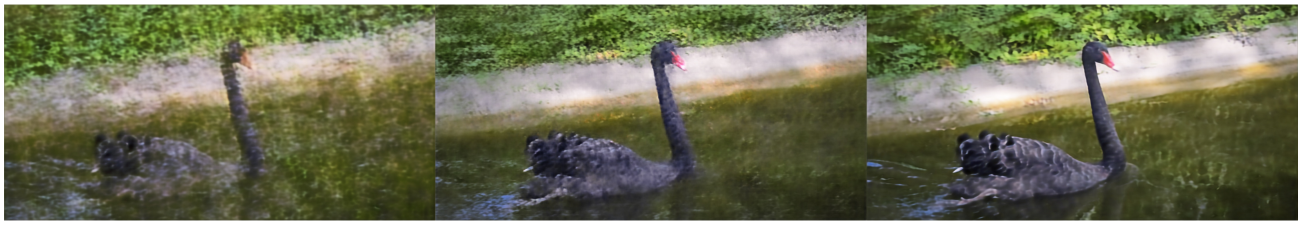
Figure 12. **Interpolation** results.

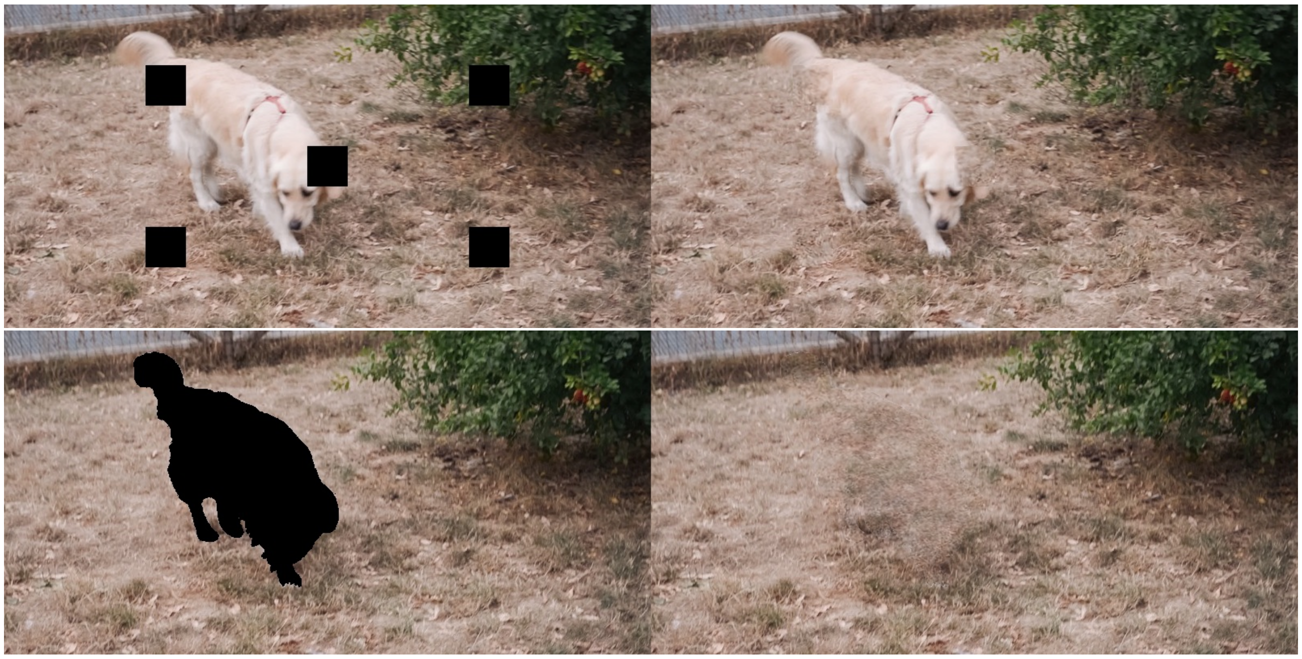

Figure 13. **Inpainting** results.