Real-time 6K Image Rescaling with Rate-distortion Optimization
Supplementary Material

Summary

This supplementary material is organized as follows.

- Section 1 introduces the implementation of our architecture and training details.
- Section 2 shows more comparison with previous work.
- Section 3 discusses more ablation studies of our designs.

1. Implementation Details

We implement our model in PyTorch and train on a single Nvidia RTX3090 GPU. In this section, we describe the details of our architecture and training settings.

Network Architecture. Table 1 and Table 2 respectively show details of our encoder and decoder. In Table 1, we describe the architecture of our UNet-based encoder [12]. "PixelUnshuffle 4x" stands for the rearrangement of elements [13] which downsamples the HR image by a factor of 4. "3×3, 64, LeakyReLU" denotes a 2d convolution operation of kernel size 3, output channel 64, followed by a LeakyReLU operation. We use the implementation of Residual Dense Block from [15], where "ResidualDenseBlock-32" refers to a Residual Dense Block with minimum channel 32. We save the produced quantization tables and output coefficients into a single JPEG file using TorchJPEG [7]. Building blocks are shown in brackets, with the number of blocks stacked. Downsampling is performed at the beginning of the downsampling block using max pooling of stride 2. After the first convolution in the upsampling block, we upsample the decoder feature with Pixel Shuffling Operation. Then, skip connections with the encoder features are conducted.

In Table 2, we show the details of our efficient decoder, which is developed based on EDSR [10]. We extract features $f(\widehat{C}) \in \mathbb{R}^{24\times\frac{H}{s}\times\frac{W}{s}}$ and concatenate $f(\widehat{C})$ with the RGB image $\hat{y}$. The concatenated features are fed into the decoder to reconstruct the HR image $\hat{x}$:

$$\hat{x} \quad = \quad D(\hat{y} \oplus f(\widehat{C})), \tag{1}$$

where $\oplus$ is the concatenation operator along the channel dimension.

Stage	Building Block	Output Size
Input Downsample	PixelUnshuffle 4x $3\times3, 64,$ LeakyReLU ResidualDenseBlock-32	$H/4 \times W/4 \times 64$
Downsampling Block1	$\begin{bmatrix} 3\times3, 128, \\ \text{LeakyReLU} \end{bmatrix} \times 2$ ResidualDenseBlock-64	$H/8 \times W/8 \times 128$
Downsampling Block2	$\begin{bmatrix} 3\times3, 256, \\ \text{LeakyReLU} \end{bmatrix} \times 2$ ResidualDenseBlock-128	$H/16 \times W/16 \times 256$
Upsampling Block1	$3\times3, 512$ $\begin{bmatrix} 3\times3, 128, \\ \text{LeakyReLU} \end{bmatrix} \times 2$ ResidualDenseBlock-128	$H/16 \times W/16 \times 128$
Upsampling Block2	$3\times3, 256$ $\begin{bmatrix} 3\times3, 64, \\ \text{LeakyReLU} \end{bmatrix} \times 2$ ResidualDenseBlock-64	$H/8 \times W/8 \times 64$
Output layer	$3\times3, 3$	$H/4 \times W/4 \times 3$

Table 1. **Architectures of our encoder.**

Frequency Feature Extractor f	Building Block	Output Size
Input Convolution	$3\times3, 24$	$H/32 \times W/32 \times 24$
Residual Convolution Block	$\begin{bmatrix} 3\times3, 24, \\ \text{ReLU}, \\ 3\times3, 24, \end{bmatrix} \times 16$	$H/32 \times W/32 \times 24$
Output Convolution	$\begin{bmatrix} 3\times3, 96, \\ \text{PixelShuffle 2x} \end{bmatrix} \times 3$	$H/4 \times W/4 \times 24$

Decoder-full	Building Block	Output Size
Input Convolution	$3\times3, 24$	$H/4 \times W/4 \times 24$
RRDB Blocks	$\begin{bmatrix} \text{ResidualDenseBlock-32}, \\ \text{ResidualDenseBlock-32}, \\ \text{ResidualDenseBlock-32}, \end{bmatrix} \times 12$	$H/4 \times W/4 \times 24$
Output Convolution	$\begin{bmatrix} 3\times3, 96, \\ \text{PixelShuffle 2x} \end{bmatrix} \times 2$	
$33, 3,$	$H \times W \times 3$	

Table 2. **Architectures of our efficient decoder.**

Training with Pixel Loss. The model is trained with batch size 16 and patch size 256×256 in each iteration. The initial learning rate is 2e-4. The learning rate is decayed by 0.75 for every $100,000$ iterations.

Test-time Fine-tuning during Downscaling. During downscaling stage, we optimize the pre-trained encoder with a fixed pretrained decoder. In the optimization dur-

Method	Bpp of File Format		Bitrate↓-Distortion↑ Kodak	
Architecture	Bitstream	JPEG	Sum of bpp	LR PSNR / HR PSNR
Hyperprior [5]+JPEG	0.214	0.148	0.51	33.41 / 29.22
HIFIC [11]+JPEG	0.172	0.148	0.32	33.41 / 29.35
Ours	-	0.299	**0.30**	**33.55 / 29.42**

Table 3. Comparison of our Hyperthumbnail framework against learned compression with JPEG thumbnail. In additional baseline, we provide a JPEG thumbnail besides learned compression, and take the sum of bitstream size and JPEG size to calculate the final bpp. Our framework has better rate-distortion performance than "Compression+JPEG" baseline.

ing downscaling, we use full-resolution test images without augmentation as batch size 1 to accelerate optimization. For each image in the test set, we optimize the encoder and QPM for 100 iterations with a learning rate of 2.0×10^{-4}.

2. Additional Comparison Results

2.1. Comparison with Compression+JPEG

The key difference between our rescaling framework with learned image compression [4, 5, 11] is that our hyperthumbnail provides an instant preview that is compatible with existing JPEG codec. However, learned image compression typically requires GPU for decompression using neural networks. For users of learned compression, one practical solution to support instant preview is saving a low-resolution JPEG image as a thumbnail besides compressed bitstream, which we refer to as "Compression+JPEG" framework.

Our rescaling framework has two advantages over the above "Compression+JPEG" solution. (a) First, we embed the high-frequency information into a compact single JPEG file that is easy to deliver. In contrast, the "Compression+JPEG" framework requires two different file formats for preview and compressed bitstreams, which is inconvenient for storage and transmission. (b) Secondly, as evaluated in Table 3, it takes considerable storage for standard JPEG [14] thumbnails to have similar fidelity as our encoded LR thumbnails. We choose Hyperprior [5] and HIFIC [11], two state-of-the-art compression methods with a similar running time as ours to build "Compression+JPEG" baseline. Because of information redundancy in the bitstream and the JPEG file, "Compression+JPEG" framework takes more storage to achieve comparable LR PSNR and HR PSNR with our result. In summary, our Hyperthumbnail provides a compact and succinct representation to support both instant preview and high-frequency reconstruction.

2.2. Quantitative comparison with JPEG

In Figure. 1, we provide an additional comparison of our rate-HR-distortion performance with baselines. Previous

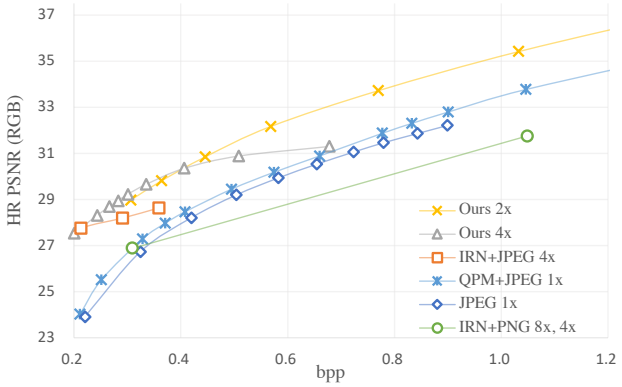

Figure 1. **The rate-HR-distortion curve on Kodak [1] dataset.** Our method ($s = 2, 4$) outperforms JPEG, IRN [16] in the RD performance. For the 'QPM + JPEG' curve, where $s = 1$, we follow the standard JPEG algorithm and adopt QPM module as a plugin for table prediction.

rescaling methods such as IRN [16] in PNG format with different rescaling scale ("IRN+PNG $8\times$,$4\times$") is even worse than "JPEG" [14]. "IRN+JPEG $4\times$" shows that JPEG format with different quality factors boosts the rescaling methods. In contrast, our method is much better than the above three baselines, thanks to our image-specific quantization design.

Another interesting extension of our work is to use QPM as a plug-in to improve the performance of traditional JPEG compression, which is shown by "QPM + JPEG" curve. We set the rescaling factor as $s = 1$, remove our encoder and decoder, and only train our QPM Module as a compression method. An improvement of 0.5dB is observed at most bitrate constraints.

Note that traditional image compression codec, such as JPEG, does not produce an LR embedding as rescaling methods. Thus, their results are only for reference.

2.3. Decoding efficiency comparison with AVIF and JPEGXL.

With no available GPU implementation, we test the decoding efficiency of AVIF **(344.8 ms)** and JPEG-XL **(257.9 ms)** on an Intel Xeon Gold 5218 server CPU at 4K resolution and 0.3 bpp. In comparison, our decoder **(14.1 ms)** is much faster with GPU acceleration. According to a survey [2], the usage statistics of JPEG **(77.8%)** is much higher than AVIF **(0.1%)**. Meanwhile, JPEG-XL will soon be deprecated by Chrome, and some websites (e.g., Twitter and Shopee) use JPEG as the only lossy image file format. Meanwhile, our framework can be integrated into most apps (e.g., Chrome and WhatsApp) without building extra support for transmission and previewing, which is more practical and useful.

Method	Bitrate↓-Distortion↑ Kodak		Time(ms)↓	LR PSNR ↑ / HR PSNR↑		
Architecture	bpp	LR PSNR / HR PSNR	Down / Up	BSD100	Urb100	DIV2K
Ours w/o f	0.30	33.26 / 29.32	86.2 / 32.3	32.56 / 27.57	30.90 / 26.46	33.40 / 30.00
Ours- w/o f-b22	0.30	33.24 / 29.27	86.2 / 38.6	32.51 / 27.63	30.89 / 26.69	33.48 / 30.09
Ours	0.30	33.55 / 29.42	86.2 / 37.8	32.90 / 27.66	31.16 / 26.62	33.62 / 30.15
Ours enc-48	0.30	33.51 / 29.17	63.7 / 37.8	32.88 / 27.58	31.21 / 26.51	33.62 / 30.05
Ours enc-96	0.30	33.52 / 29.29	183.5 / 37.8	32.88 / 27.66	31.22 / 26.66	33.62 / 30.15

Table 4. Ablation study of our encoder-decoder architectures on the downsampling / upsampling time and the PSNR of reconstructed HR image / LR thumbnail.

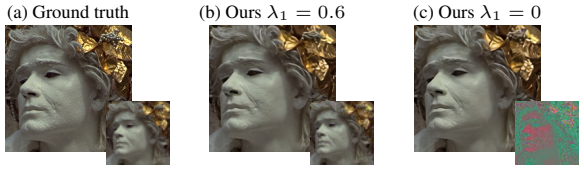

Figure 2. **guidance loss ablation on Kodak [1] image *kodim17*.** We visualize the HR images with their LR counterparts at the bottom-right. (b) (c) are produced by $4\times$ HyperThumbnail models trained with different λ_1 and the bpp is 0.4.

3. Additional Ablation Study

Guidance Loss. In Figure.2, we conduct a qualitative ablation study of guidance loss. It demonstrates that guidance loss is crucial to preserve the quality of LR images, without introducing noticeable degradation to HR images.

Frequency-aware Decoder. Because the efficiency of HR reconstruction is important for a better user experience, our decoder architecture has to be succinct and effective. In Table 4, we study the capacity of our decoder with frequency feature extractor f. Removing f in our framework("Ours-w/of") results in a drop in both the HR and LR RD performance. Based on "Ours-w/of", we increase the residual blocks of the decoder from 16 to 22. "Ours-w/of-b22" takes more upscaling time, but it ends up with a similar HR RD performance with "Ours-w/ f" and a significantly inferior LR RD performance. Since the spatial resolution of quantized coefficients $\hat{C}$ is $\frac{1}{8}$ of the embedding image $\hat{y}$, the running time of frequency feature extractor only accounts for 14.6% of the entire decoder. Thus, our frequency feature extractor f demonstrates a strong advantage with negligible computation cost.

Asymmetric encoder-decoder. We quantitatively evaluate the influence of the encoder capacity on the RD performance in the bottom two rows of Tab. 4. Based on "Ours", We adjust the channel of our encoder from 64 to 48 and 96. The experiment shows that our framework benefits from the larger encoder. Since the 96-channel encoder is $2\times$ slower than 64 channel encoder and the improvement is marginal, we set encoder channel to 64 in most of our experiments to ease training.

Also, larger decoders can be applied to the same HyperThumbnail for better reconstruction quality. As shown in the table below, "Ours-large" decoder outperforms "Ours-full" decoder in the PSNR of HR significantly (Tab. 5) with $4\times$ of parameters, sharing the same HyperThumbnails.

Decoder	Kodak	Set5	Set14	BSD100	Urb100	DIV2K
Ours-full	29.67	30.48	28.21	27.93	27.35	30.49
Ours-large	29.74	30.56	28.39	28.01	27.74	30.61

Table 5. HR reconstruction PSNR with different decoder capacity.

3.1. Additional qualitative results

In this section, we visualize more results on the DIV2K [3] validation dataset and the FiveK [6] dataset. Our model achieves the best balance between the embedding artifacts on LR and the restoration of HR detail. Our approach outperforms baseline methods, especially in texture restoration. All baseline models are trained on the same DIV2K training dataset to fit on guidance LR $\hat{y}$ and target HR $\hat{x}$. The results are cropped from the original image to ease comparison, please refer to Fig. 3. Also, in Fig. 4, we visualize more rescaling results of real world 6K images with our framework.

Acknowledgement

We express our sincere gratitude to our friends and domain experts, Junming Chen, Yue Wu, and Yu Wang for their invaluable contributions in the design of our project. Additionally, we extend our appreciation to Xiaogang Xu and Xilin Zhang for reviewing and revising the writing.

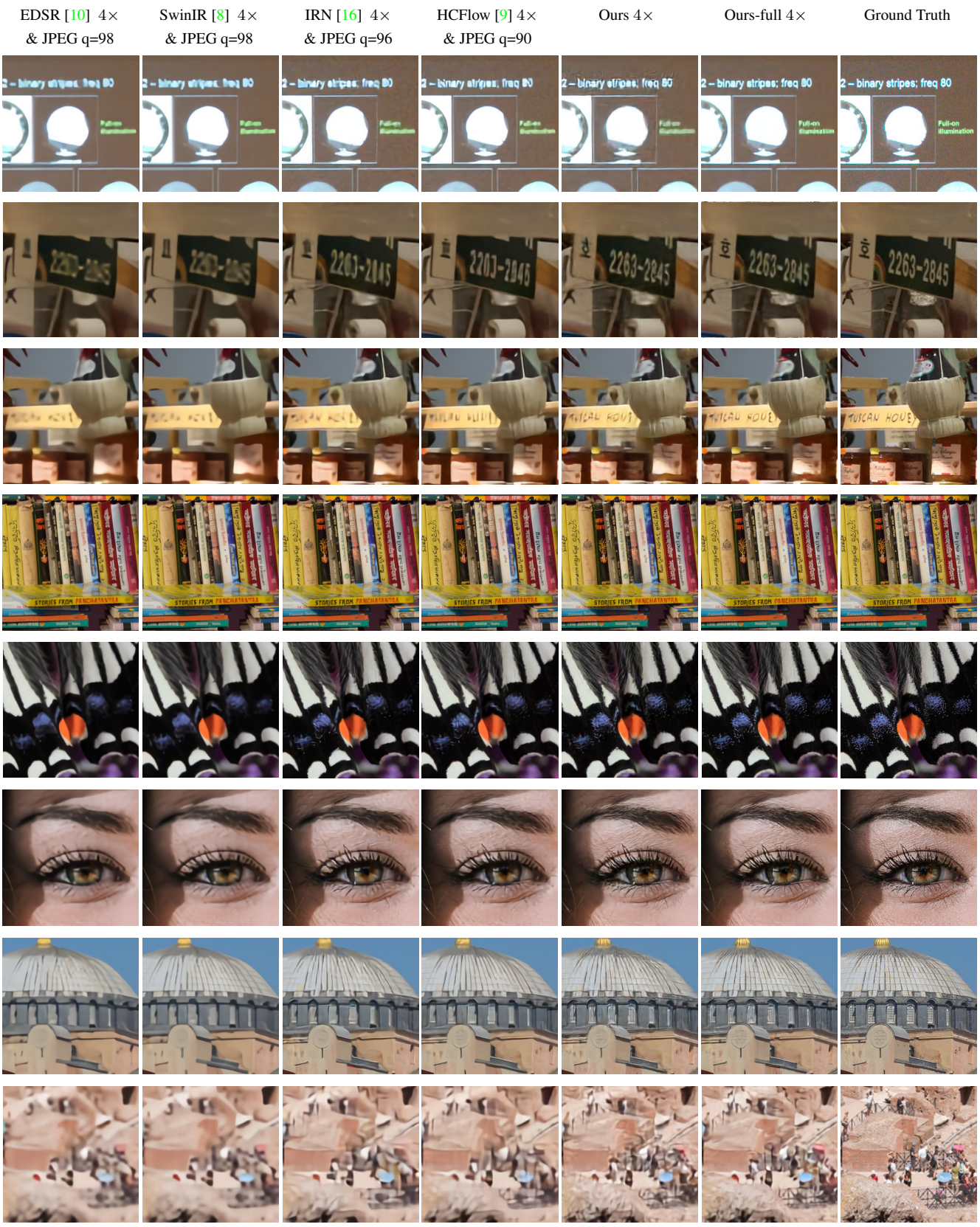

Figure 3. Visual results of performing 4× rescaling on the DIV2K [3] and FiveK [6] datasets with baseline methods and our models. The images are cropped to ease the comparison. Please zoom in for details.

Figure 4. More results of $4\times$ rescaling with our framework on real-world 6K images [6]. Please zoom in for details. Note that the images here are compressed due to the size limit of camera-ready.

References

[1] Kodak lossless true color image suite. http://r0k.us/graphics/kodak/. Accessed: 2022-03-01. 2, 3

[2] Usage statistics of JPEG for websites. https://w3techs.com/technologies/details/im-jpeg. Accessed: 2022-03-01. 2

[3] Eirikur Agustsson and Radu Timofte. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of CVPR Workshops*, 2017. 3, 4

[4] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression. In *Proceedings of ICLR*, 2017. 2

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *Proceedings of ICLR*, 2018. 2

[6] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *Proceedings of CVPR*, 2011. 3, 4, 5

[7] Max Ehrlich, Larry Davis, Ser-Nam Lim, and Abhinav Shrivastava. Quantization guided JPEG artifact correction. In *Proceedings of ECCV*, 2020. 1

[8] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *Proceedings of ICCV Workshops*, 2021. 4

[9] Jingyun Liang, Andreas Lugmayr, Kai Zhang, Martin Danelljan, Luc Van Gool, and Radu Timofte. Hierarchical conditional flow: A unified framework for image super-resolution and image rescaling. In *Proceedings of ICCV*, 2021. 4

[10] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of CVPR Workshops*, 2017. 1, 4

[11] Fabian Mentzer, George Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. In *Advances in NeurIPS*, 2020. 2

[12] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of MICCAI*, 2015. 1

[13] Wenzhe Shi, Jose Caballero, Ferenc Huszar, 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 CVPR*, 2016. 1

[14] Gregory K. Wallace. The JPEG still picture compression standard. *Commun. ACM*, 34(4):30–44, 1991. 2

[15] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. ESRGAN: enhanced super-resolution generative adversarial networks. In *Proceedings of ECCV Workshops*, 2018. 1

[16] Mingqing Xiao, Shuxin Zheng, Chang Liu, Yaolong Wang, Di He, Guolin Ke, Jiang Bian, Zhouchen Lin, and Tie-Yan Liu. Invertible image rescaling. In *Proceedings of ECCV*, 2020. 2, 4