Signal Processing: Image Communication 128 (2024) 117172

Contents lists available at ScienceDirect

Signal Processing: Image Communication

journal homepage: www.elsevier.com/locate/image

Learned fractional downsampling network for adaptive video streaming

Li-Heng Chen [a,b,*], Christos G. Bampis [b], Zhi Li [b], Joel Sole [b], Chao Chen [c], Alan C. Bovik [a]

[a] Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA
[b] Netflix, Inc, Los Gatos, CA, USA
[c] Discord, Inc, San Francisco, CA, USA

ARTICLE INFO

Keywords:
Downsampling
Convolutional neural networks
Adaptive video streaming
Perceptual video quality

ABSTRACT

Given increasing demand for very large format contents and displays, spatial resolution changes have become an important part of video streaming. In particular, video downscaling is a key ingredient that streaming providers implement in their encoding pipeline as part of video quality optimization workflows. Here, we propose a downsampling network architecture that progressively reconstructs residuals at different scales. Since the layers of convolutional neural networks (CNNs) can only be used to alter the resolutions of their inputs by integer scale factors, we seek new ways to achieve fractional scaling, which is crucial in many video processing applications. More concretely, we utilize an alternative building block, formulated as a conventional convolutional layer followed by a differentiable resizer. To validate the efficacy of our proposed downsampling network, we integrated it into a modern video encoding system for adaptive streaming. We extensively evaluated our method using a variety of different video codecs and upsampling algorithms to show its generality. The experimental results show that improvements in coding efficiency over the conventional Lanczos algorithm and state-of-the-art methods are attained, in terms of PSNR, SSIM, and VMAF, when tested on high-resolution test videos. In addition to quantitative experiments, we also carried out a subjective quality study, validating that the proposed downsampling model yields favorable results.

1. Introduction

Digital videos have become the largest portion of Internet traffic, driven by the evolution of consumer electronics and the tremendous popularity of video sharing and streaming platforms. Recently, the demand for online videos has surged even further during the global pandemic. In a streaming video workflow, each component plays an important role in achieving end-to-end efficiency. For example, raw video sources determine the base video quality, while the selection of encoding parameters affects the rate–distortion tradeoffs of video compression. An important recent advance is adaptive streaming framework, a technique that is widely used by video streaming services like Netflix, Youtube, and Facebook to improve the quality of experience of viewers [1–3].

In a general workflow for adaptive video streaming like that illustrated in Fig. 1, a high-resolution source video segment (typically a scene) is spatially downscaled into multiple resolutions using a set of scale factors $M \in \mathbb{Q}_+$. The videos at each resolution are then encoded using different quantization parameters (QPs), yielding a variety of rate-quality tradeoffs. From amongst the generated resolution-bitrate representations, a "best" video chunk is determined (typically by perceptual optimization using an objective video quality model like SSIM [4], VIF [5], or VMAF [6].), then streamed. On the client side, the bitstream is decoded and scaled back to the device resolution prior to display. It is important to understand that the scale factor $M \geq 1$ may be any reasonable rational number. Despite not being explicitly defined in the adaptive streaming protocols, some specific encoding resolutions that require fractional downsampling are essential to enable adaptation to device capability. For instance, streaming a 1080p source at 720p resolution ($M = 1.5$) is one of the most common choices of streaming services. A system lacking such resolutions may deliver suboptimal rate–distortion performance. The upscaling algorithms implemented on different display devices generally vary, and are not often known by the stream providers. Hence, from a providers perspective, the encoding pipeline cannot be optimized for a specific upsampling algorithm.

Over the past few years, deep neural networks have been applied to solve a wide diversity of video processing problems [7–9], including architectures designed for image resizing. Unlike legacy resizing algorithms, which rely heavily on conventional signal processing concepts, here we will optimize the parameters of a learned resizer in an end-to-end manner. In this way, we seek to improve adaptive streaming pipelines using CNN-based downsampling protocols. From a practical perspective, several design facets/challenges need consideration:

* Corresponding author at: Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA.
E-mail address: lhchen@utexas.edu (L.-H. Chen).

https://doi.org/10.1016/j.image.2024.117172
Received 25 October 2021; Received in revised form 5 August 2022; Accepted 15 July 2024
Available online 22 July 2024

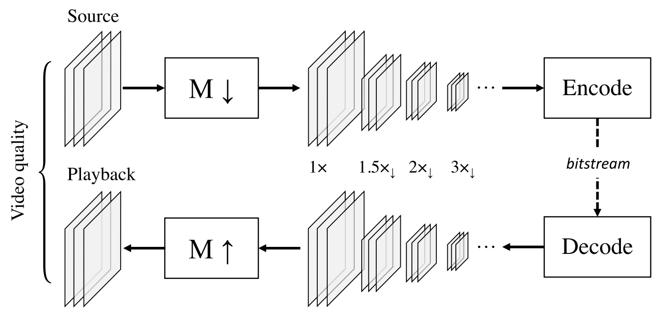


Fig. 1. A general encoding/decoding pipeline for adaptive streaming. Downsampling (represented by M↓) and upsampling (represented by M↑) are an integral part of the pipeline. The parameter $M \geq 1$ denotes the current scale factor, which may be varied. The end-to-end video quality can be used to determine encoding recipe.

1. **Model generality.** As previously stated, the upscaling algorithm at the receiver is unknown at the encoder. Hence, downsampling models should generalize well to different upsampling algorithms. It would also be beneficial if the models are agnostic to encoding algorithms, since video contents may be streamed in the format of multiple codecs.
2. **Arbitrary scale factors.** When designing video resizers, allowing for non-integer scale factors is an important, but often neglected, feature of compression workflows. However, currently available convolutional layers can only resize their inputs by integer factors.
3. **Model complexity.** A CNN-based model used for scaling should not be too computationally expensive. Since modern video encoding pipelines often process contents with resolution as large as 4K, deep models with many parameters may fail to meet memory efficiency or speed constraints.

Towards addressing these challenges, we conducted an online subjective video quality study that we use to analyze and compare our resizing model to justify the perceptual relevance of our results. We use this tool to support the main contributions of this work, which we summarize as follows:

- We show a way to re-design network architectures to learn residuals *prior to* scaling. This provides significant improvement over our early results [10].
- Our new model allows for fractional resizing factors.
- As a demonstration of the value of the approach, we integrate the learned downsampling models with a realistic video encoding pipeline for adaptive video streaming, to achieve improved coding efficiency.
- We conduct extensive experiments to validate both the objective and subjective quality improvements delivered by our proposed approach. Our model can generalize well across different video codecs and upsamplers.

The rest of this paper is organized as follows. Section 2 reviews the related literature, while Section 3 presents details of our new proposed network architecture which allows non-integer resizing factors. Experiments and analysis are presented in Section 4. Finally, Section 5 concludes with a discussion of future directions of research.

2. Background

We begin with a background review of studies related to encoding for adaptive video streaming and objective video quality. Following that, we discuss recent research work on conventional/machine learning based video scaling algorithms.

2.1. Brief overview of adaptive video streaming

Many early ‘fixed bitrate ladder’ approaches determine a resolution for each of a set of specified bitrates. These utilized look-up tables that were designed empirically [11], or that can be adapted to video content [12]. Studies [13–15] have shown that encoding videos at lower resolutions generally results in better quality, when videos are compressed to low bitrates. These schemes balance distortions produced by scaling against those by video encoding.

A more sophisticated method is to encode each source video at multiple combinations of resolution and compression levels, yielding multiple rate–distortion (R–D) curves. Optimal encoding recipes can then be selected on the convex hull of these R–D curves. This concept has been employed by Netflix [12] and other streaming companies [1,3]. More sophisticated resolution adaptation approaches have also been investigated [16–18], involving super-resolution (SR) [19], statistical modeling [20], or classification [21]. Two recently proposed frameworks, ViSTRA [22] and ViSTRA2 [23], utilize spatio-temporal (and bitdepth) resolution adaptation and a CNN-based super-resolution model to obtain significant improvements in coding efficiency.

2.2. Objective video quality assessment algorithms

Another topic closely relevant to adaptive video streaming is the prediction of perceptual video quality. We will require the use of full-reference video quality assessment (VQA) models, given that high-quality reference data is usually available at the encoder (provider) side of a video streaming pipeline. Rather than using the PSNR, which correlates poorly with visual perception [24], we can select from a variety of powerful perception-based VQA models, such as VIF [5], MOVIE [25], ST-MAD [26], VQM-VFD [27–29], and so on [30–33]. The most successful of these are the SSIM family [4,34] of computationally simple models, and VMAF [6]. These are currently widely deployed to perceptually optimize a large fraction of compressed Internet video traffic, including downsampling conducted as part of compression.

2.3. Resizing algorithms

Beyond early, simple approaches to video resizing or super-resolution, such as bilinear, bicubic [35], and Lanczos interpolation, a variety of more sophisticated models have been proposed. For example, patch-based methods have recently been proposed that exploit intra [36,37] or inter [38,39] similarities, and data-driven models [40–43] have produced excellent results. However, these kinds of upscaling models are expensive for consumer devices, and simple interpolation filters are still used by most of web browsers, video players, and handheld devices. Importantly, upscaling algorithms are usually implemented on the device (client) side, and they are generally not known during encoding.

As compared to the SR problem, optimization of resolution reduction has been much less extensively studied, but there has been some recent work. Recent approaches include aligning local image features by optimizing a joint bilateral filter [44], and preserving perceptually important details [45–47]. Deep learning based downsampling models include CNN-CR [48], which applies a 10-layer CNN to learn the residuals of bicubic downsampled images. Kim et al. jointly trained a downscaling network and an upscaling network to address task-related scenarios [49]. The content adaptive resampler (CAR) model [50] also estimates both downsampling and upsampling kernels by training an existing SR network. We are aware of only a few studies [48,51] that develop learned downsampling for compression. While these methodologies can produce significant improvements in compression efficiency, non-integer scale factors were not considered. Another recent solution [52] treats non-integer scaling as a special case. This method, which is the most related to our work, operates by resizing on input feature map to a desired resolution *before* feeding into a CNN, but only when the scale factor is not an integer. Otherwise, a convolutional layer with integer stride is used. Our proposed elementary block, which we introduce in Section 3.1, takes a simpler form, and can be used for scaling by *arbitrary* factors. Moreover, the method in [52] presumes the use of bilinear upsampling on any device being fed downstream. As we will demonstrate in Section 4.5, training downsampling networks against subsequent bilinear upsampling can produce results that "overshot", if they are then employed in systems using more sophisticated upsamplers.

3. Proposed method

Next we describe the design methodology behind our proposed downsampling models, including the network architecture, training framework, the loss functions that we employ, and details regarding the implementation.

3.1. Dealing with fractional scale factors

Towards understanding how we can resolve the integer limitation on downsample scaling that is imposed by CNNs, we begin by exploring two different methods:

3.1.1. "Digital signal processing" approach

Changing the sampling density of digital signals by a non-integer factor can be achieved by applying successive expansion and decimation operations, given a rational scale factor

$$M = \frac{L}{N}, \tag{1}$$

where L and N are integers. Inspired by the classic fractional rate conversion structure, we can design an upsampling network $f_{\uparrow L}$, followed by a downsampling network $f_{\downarrow N}$, using integer scale factors L and N, respectively. As illustrated in Fig. 2, let $x \in \mathbb{R}^{\mathrm{W}\times\mathrm{H}}$ be an input frame or feature map to be resized. Then an intermediate frame or map can be generated by

$$\tilde{x} = f_{\uparrow L}(x) \in \mathbb{R}^{\mathrm{W}L\times\mathrm{H}L}. \tag{2}$$

Then, $\tilde{x}$ is processed by the downsampling network $f_{\downarrow N}$, resulting in a downscaled output

$$\hat{x} = f_{\downarrow N}(\tilde{x}) = f_{\downarrow N}\left(f_{\uparrow L}(x)\right), \tag{3}$$

where $\hat{x} \in \mathbb{R}^{\frac{\mathrm{W}L}{N}\times\frac{\mathrm{H}L}{N}}$ has its resolution reduced by a factor M along both dimensions. This structure is quite similar to the polyphase representation used in multirate signal processing. The trainable convolutional layers placed after the interpolation/decimation modules are analogous to polyphase-decomposed filters. However, one problem with this direct approach can occur when either the upsampling factor L, or the input resolution is large, creating very high requirements on memory. For example, converting a 2160p input to 1440p ($M = 1.5$) with $(L, N) = (2, 3)$ may introduce out-of-memory issues, since feature maps of 4320p resolution have to be stored in the upsampling network $f_{\uparrow L}$.

3.1.2. Using a conv-resize block

Another approach is to utilize an alternative building block, as we recently proposed in our preliminary work [10].[1] We begin by comparing two commonly used convolutional blocks, as shown in Figs. 3(a) and 3(b). Given an input x of spatial resolution $W \times H$, a typical trainable convolutional block with downsampling outputs a feature map $\mathcal{F}_{\downarrow M}(x)$, yielding a map having reduced resolution $\frac{W}{M} \times \frac{H}{M}$. Normally, downsampling by a factor M is accomplished as (Fig. 3(a))

$$\mathcal{F}_{\downarrow M}(x) = \mathcal{C}_{s=M}(x), \tag{4}$$

[1] We have become aware of a near-simultaneous submission [53] using similar ideas from Google Research.

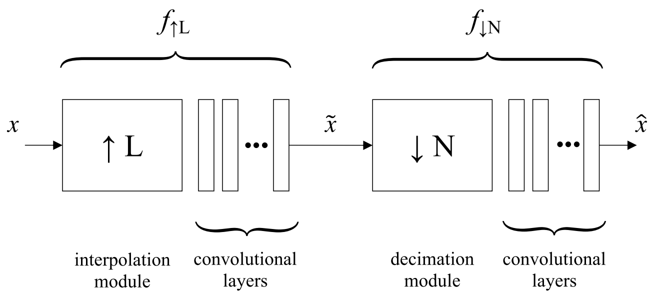


Fig. 2. A conceptual network structure that allows for fractional resizing factors. The '↑ L' block can be realized by a deconvolution layer (e.g. transposed convolution), while the '↓ N' block can be as simple as a convolutional layer having integer stride.

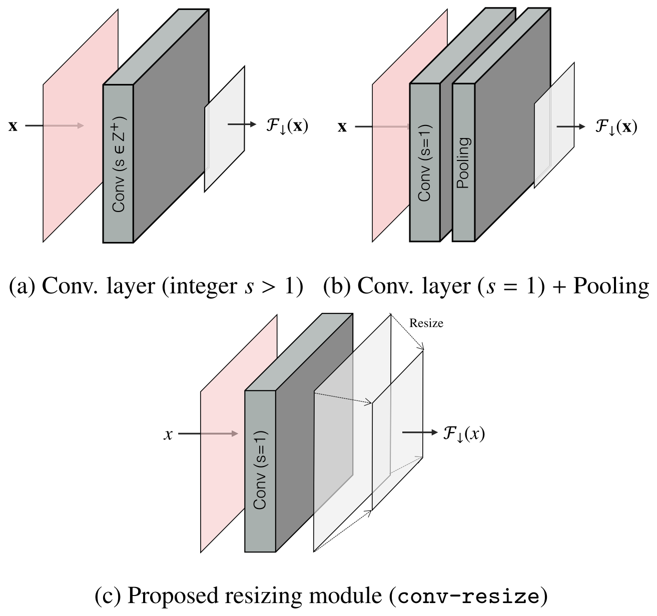

x
Conv (s ∈ Z⁺)
$\mathcal{F}_{\downarrow}(\mathbf{x})$
x
Conv (s=1)
Pooling
$\mathcal{F}_{\downarrow}(\mathbf{x})$
(a) Conv. layer (integer $s > 1$)
(b) Conv. layer ($s = 1$) + Pooling
Resize
x
Conv (s=1)
$\mathcal{F}_{\downarrow}(x)$
(c) Proposed resizing module (conv-resize)

Fig. 3. Comparison of three convolutional blocks yielding down-sampled outputs. (a) A convolutional block resizes by controlling the integer stride parameter s. (b) A convolutional block with $s = 1$ resizes using an additional pooling layer. (c) Our proposed resizing module is constructed as a convolutional layer with $s = 1$ followed by a resize operation, allowing for arbitrary resizing factors.

where $\mathcal{C}_s$ denotes convolution outputs computed at samples separated by an (*integer*) stride parameter s. Alternatively, a pooling layer $\mathcal{P}_{\downarrow M}$ using (typically) max pooling or average pooling, yielding a reduced resolution map (Fig. 3(b)) may be applied following convolution with stride $s = 1$,

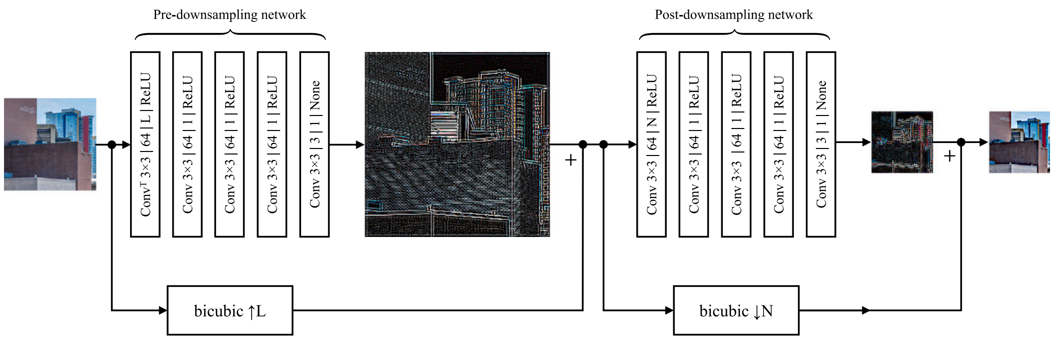
$$\mathcal{F}_{\downarrow M}(x) = \mathcal{P}_{\downarrow M}\left[\mathcal{C}_{s=1}(x)\right]. \tag{5}$$

Note that the resolution is reduced by a factor of *integer* patch size, since only one value is pooled from each patch in a feature map.

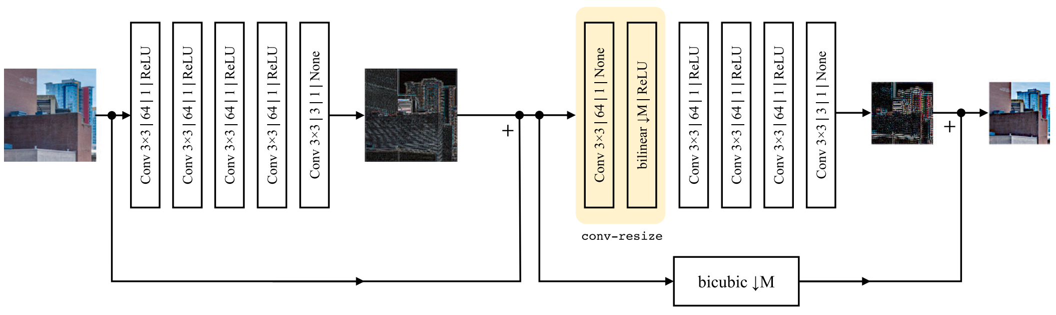
Unfortunately, these two schemes only allow downsampling by *integer* scale factors $M \in \mathbb{Z}_+$, limiting flexibility which may be required when implementing resolution changes in many broader applications. As shown in Fig. 3(c), we address this problem by replacing the pooling layer $\mathcal{P}$ in (5) with a differentiable resizer $\mathcal{R}$ that supports arbitrary scale factors $M \in \mathbb{Q}_+$

$$\mathcal{F}_{\downarrow M}(x) = \mathcal{R}_{\downarrow M}\left[\mathcal{C}_{s=1}(x)\right], \tag{6}$$

which we dub the `conv-resize` block. Thus, the proposed building block is a convolutional layer `conv` followed by a `bilinear` resizer, and the entire network can be trained end-to-end by back-propagating through the forward model. A a similar "`resize-conv`"



(a) The *ProgDown* network architecture. Note that the "bicubic ↑L" block is the identity function when $L = 1$



(b) The *ProgDownLite* network architecture. The yellow shading denotes the proposed `conv-resize` block (Fig. 3(c)) followed by ReLU activation.

Fig. 4. Architecture of the two proposed models. Both consist of a pre-downsampling network followed by a post-downsampling network. The convolution parameters are denoted as: kernel size (height×width) | output channel (number of filters) | stride | activation function.

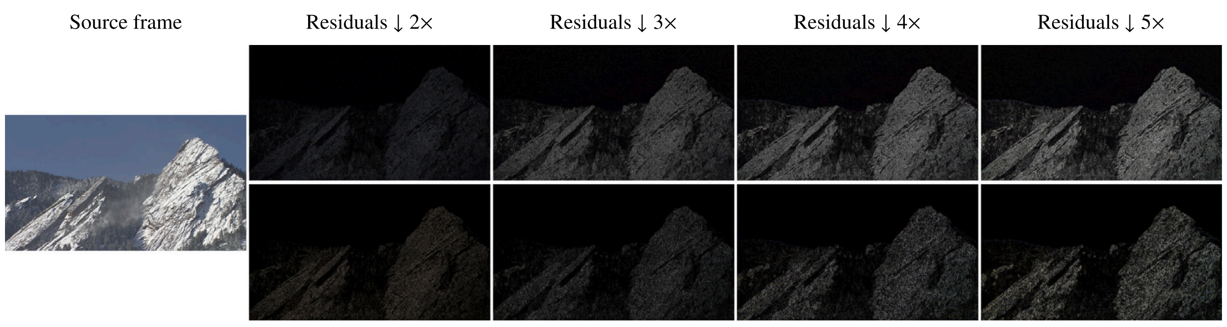


Fig. 5. Residuals output from the pre-downsampling network (top row) and the post-downsampling network (bottom row) trained using different scale factors. The first column shows the corresponding source frame extracted from the test sequence *snow_mnt*. For visualization, we scaled the bottom row back to the original resolution and processed all residuals by the same amount of contrast adjustment.

block has been used to mitigate artifacts that can arise from summing uneven overlapped responses in transposed convolution [54]. Another building block, the Spatial Transformer Network (STN) [55], also comprises a bilinear interpolation layer, making it applicable to fractional resize factors. However, interpolation in STN is used to sample affine-transformed coordinates. Recent work reported in [53] also proposed a bilinear resizer to handle arbitrary resize factors in a network. The authors show that resizing an input image by a learned fractional resizer can lead to better performance in various computer vision tasks.

3.2. *Proposed progressive network for downsampling*

We construct our downsampling models in a progressive manner. Generally, they take a 3-channel RGB signal at original resolution as input, and progressively predict residual signals at two scales by cascading two sub-networks. The spirit behind this approach is simple: the first sub-network pre-filters the original image, thereby capturing the spatial information that we aim to preserve during reconstruction. We will refer this sub-network to as the "pre-downsampling network". The output is then fed to the second sub-network (post-downsampling), whose goal is to extract features that can "repair" the spatially degraded image.

As illustrated in Fig. 4, each sub-network consists of five stages of convolutional layers. In this example, the sizes of the convolutional and transposed convolutional layers are fixed at 3×3 in all layers, while the number of filters is 64 in each of the first 4 stages. We zero pad the boundaries of the feature maps before applying each convolution, so that the output size is not reduced. Except for the last layer, all of the convolutional layers are activated by a ReLU nonlinearity. Finally, a 3-channel output is produced, yielding a residual that is added element-wise to the (bicubic-resized) input image. The parameterization of each layer is detailed in the figure.

We propose non-integer scaling via the two approaches described in Section 3.1, resulting in two model variants:

The *ProgDown* model: The *ProgDown* model depicted in Fig. 4(a) converts the input dimension by combining upsampling and downsampling by strides of L and N, respectively. The pre-downsampling network serves to "interpolate" the input, resulting in a residual map

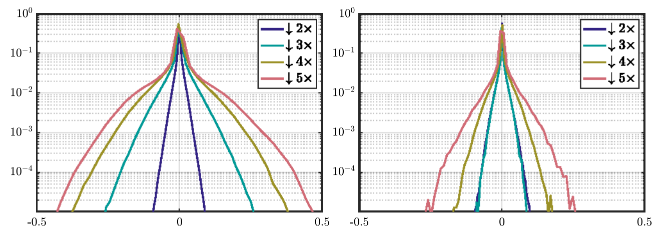


Fig. 6. Histograms of the residuals output from the pre-downsampling network (left) and the post-downsampling network (right) shown in Fig. 5.

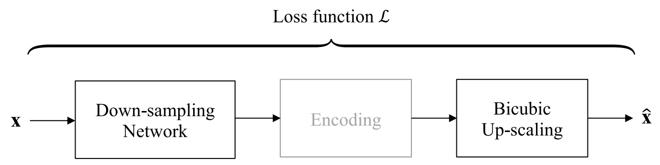


Fig. 7. Detailed training framework of a downsampling network for adaptive video streaming. The arrows indicate the flow of data in the network. As a non-differentiable component, the gray 'Encoding' block is not presented in our implementation.

with dimensions increased by a factor of L. The post-downsampling network decimates, and restores the intermediate image. This network architecture is designed using the approach described in Section 3.1.1. By selecting (L, N), any *rational* scaling ratio can be determined. For example, a video frame is scaled from 1080p to 720p by setting $(L, N) = (2, 3)$, equivalent to a network with $M = 1.5$.

The *ProgDownLite* model: In the other variation (Fig. 4(b)), the image is subsampled by the proposed `conv-resize` block (Fig. 3(c)) presented in the first layer of the post-downsampling network. Therefore, the pre-downsampling network always reconstructs a residual having the same size as its input. Note that the two models have identical pre-downsampling networks when M is an integer, since then $L = 1$. The first sub-network of this model is less computationally intensive when M is not an integer, since only L^{-2} of the pixels are processed as compared with the *ProgDown* model. Hence, we named the second variation the *ProgDownLite* model.

3.3. Visualization of learned residuals

To illustrate how the networks reduce the reconstruction error, we selected a 1080p test sequence from the Xiph Video Test Media,[2] and downsampled it using the *ProgDown* models by four integer scale factors. The residuals learned from the two sub-networks are depicted in Fig. 5. It may be observed that (non-zero) residuals are mostly located in regions containing significant details. This is not unexpected, since, high-frequency components are more susceptible to degradation from scaling. The histograms of the learned residuals are plotted in Fig. 6. From Figs. 5 and 6, it may also be observed that the *pre-downsampling* network produces more "active" residuals when the scale factor M is large. The residual map in such case tends to have larger magnitudes, and a heavy-tailed histogram. On the other hand, the residuals output by the *post-downsampling* network are relatively invariant to the scale factor. This strongly suggests the importance of filtering before decimating pixels, especially when the scale factor is large.

3.4. End-to-end training framework

Our approach to training a downsampling model is depicted in Fig. 7. With the aim to end-to-end optimize the downsampling network for easy insertion into adaptive streaming scenarios, we constructed a training framework similar to the encoding pipeline shown in Fig. 1. The input training data is down-scaled by a network with trainable parameters, encoded, and reconstructed to the original resolution by an upscaler. Since all conventional hybrid video encoders, such as H.264, are not *differentiable* components, they are not feasible for back-propagation. Thus, we relax this problem by simply removing the encoder from the pipeline. We implemented the upscaling algorithm using bicubic interpolation, as a generic and reasonably high-performance comparison. As we will show, this design choice achieves consistent performance gains for all the widely adopted upscaling algorithms that were tested.

3.5. Loss function

Let $\mathbf{x}$ be a input source batch and ϕ be the model parameters of the downsampling network f. The reconstructed batch $\hat{\mathbf{x}}$ is formulated by

$$\hat{\mathbf{x}} = \text{bicubic}_{\uparrow M}(\tilde{\mathbf{x}}) = \text{bicubic}_{\uparrow M}(f(\mathbf{x}; \phi)), \tag{7}$$

where $\text{bicubic}_{\uparrow M}$ denotes the bicubic upsampling operation with a factor of M and $\tilde{\mathbf{x}} = f(\mathbf{x}; \phi)$ is the network output. Both $\mathbf{x}$ and $\hat{\mathbf{x}}$ reside in RGB color space. Our goal is to optimize the parameters ϕ, such that the pipeline can generate a reconstructed batch $\hat{\mathbf{x}}$ that has high fidelity relative to the ground-truth(the source batch $\mathbf{x}$). To this end, we train the model against the following losses.

3.5.1. Pixel-wise loss

The loss function is defined as the Euclidean distance d between $\mathbf{x}$ and $\hat{\mathbf{x}}$:

$$\mathcal{L}_{\text{pixel}}(\mathbf{x}, \hat{\mathbf{x}}; \phi) = d(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2. \tag{8}$$

Minimizing (8) maximizes the PSNR of the reconstructed images.

3.5.2. Guided loss

As the scale factor M grows, we occasionally observe unpleasant artifacts in extreme subsampling scenarios. For instance, aliasing may occur on contents with significant high-frequency components, which may manifest as flickering noise. Aliasing can arise because the pixel-wise loss is fundamentally limited in penalizing these artifacts. To address this aspect, we apply an ℓ_1 regularization term on the residuals between network output and a guided image batch $\mathbf{x}_g$, since it is less sensitive to outliers:

$$\mathcal{L}_{\text{guide}}(\mathbf{x}, \hat{\mathbf{x}}; \phi) = \begin{cases} \left\|\hat{\mathbf{x}} - \mathbf{x}_g\right\|_1, & \text{if } M \geq 4 \\ 0, & \text{if } M < 4 \end{cases}. \tag{9}$$

We use external software (ffmpeg) to generate Lanczos-downsampled batches as the guided images. Since Lanczos produces results having fewer less high-frequency artifacts, but does promote blur, this term provides control over a trade-off between blur and aliasing.

Finally, the net loss for optimizing the network f is defined as a combination of the losses from Eqs. (8) and (9):

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}; \phi) = \mathcal{L}_{\text{pixel}}(\mathbf{x}, \hat{\mathbf{x}}; \phi) + \lambda \mathcal{L}_{\text{guide}}(\mathbf{x}, \hat{\mathbf{x}}; \phi), \tag{10}$$

where λ weights the pixel loss against the guided loss. We empirically set $\lambda = 3$ for both $M = 4$ and 5. This was accomplished by spacing values along the reals and choosing the value that best mitigated artifacts. By back-propagating through the forward model, the loss derivative was used to drive updates of ϕ.

3.6. Implementation and training details

We developed and trained our proposed deep downsampling models using the TensorFlow framework (version 1.15). The Tensorflow APIs `tf.image.resize_bilinear` and `tf.image.resize_bicubic` were utilized to conduct bilinear and bicubic operation in the network, respectively. We set the parameter `half_pixel_centers=True` to align the output pixel index with the

[2] https://media.xiph.org/video/derf/

Table 1
Performance comparison of downsampling algorithms followed by bicubic upscaling on four SR benchmarking datasets.

Scale	Downsampler	Set5 [58]			Set14 [59]			Urban100 [60]			BSDS100 [61]		
		PSNR	SSIM	VMAF	PSNR	SSIM	VMAF	PSNR	SSIM	VMAF	PSNR	SSIM	VMAF
2×	Lanczos↓	34.25	0.9477	89.32	30.51	0.8978	83.70	27.24	0.8695	78.87	29.92	0.8762	80.13
	DPID↓ [46]	33.95	0.9471	85.01	30.30	0.8990	79.12	27.04	0.8724	73.25	29.74	0.8796	76.12
	L_0-regularized↓ [47]	30.99	0.9138	72.80	28.82	0.8752	70.93	26.10	0.8559	68.49	28.62	0.8543	68.62
	CNN-CR↓ [48]	34.98	0.9555	92.25	31.08	0.9136	86.93	27.79	0.8886	82.43	30.47	0.8965	84.07
	ProgDownLite↓ (proposed)	34.98	**0.9556**	**92.27**	**31.09**	**0.9137**	86.93	27.79	**0.8887**	**82.46**	30.47	**0.8966**	**84.10**
4×	Lanczos↓	28.79	0.8427	63.15	26.08	0.7445	53.49	23.31	0.6979	44.80	26.14	0.7100	48.76
	DPID↓ [46]	28.36	0.8382	49.10	25.75	0.7418	40.02	23.00	0.6961	30.22	25.84	0.7088	36.01
	L_0-regularized↓ [47]	28.05	0.8203	47.12	25.68	0.7259	39.07	23.08	0.6859	31.05	25.79	0.6885	33.54
	CNN-CR↓ [48]	29.15	0.8552	68.85	26.22	0.7594	55.12	23.48	0.7151	45.70	26.24	0.7252	48.76
	ProgDownLite↓ (proposed)	**29.26**	**0.8582**	**71.35**	**26.39**	**0.7667**	**60.53**	**23.59**	**0.7212**	**52.43**	**26.40**	**0.7338**	**56.10**

Table 2
Summary of the tested datasets.

Dataset (Acronym)	# Vid.	Reso.	FPS	Sec.	Public?
Xiph.org (Xiph)	24	1080p	25 – 50	8 – 44	Yes
Ultra Video Group (UVG)	7	1080p	120	2.5 – 5	Yes
Netflix titles (NFLX)	68	1080p	30	∼ 5	No

resizing algorithms in ffmpeg. The Adam solver [56] was used to optimize the networks, with parameters $(\beta_1, \beta_2) = (0.9, 0.999)$ and a batch size of 16. The networks were trained on 500K iterations of back-propagation, using a learning rate that was fixed at $1e-4$. All of the models were trained using NVIDIA Tesla K80 GPU cards.

We used DIV2K [57], an image dataset consisting of 1000 very high quality pictures stored in uncompressed format, as training data. This dataset was originally designed for studying image super-resolution problems. All of the images in it have 2K pixels along either the vertical or horizontal axis. The training images were randomly cropped to $M\lfloor \frac{256}{M} \rfloor \times M\lfloor \frac{256}{M} \rfloor$, which is divisible by the scale factor M. No augmentation was applied on the training data.

4. Experiments and analysis

4.1. Quantitative comparison on image dataset

As a preliminary evaluation, we measured the reconstruction ability of the compared image downscaling models on four SR benchmark image datasets: Set5 [58], Set14 [59], Urban100 [60] and BSDS100 [61]. Using the same settings as in [48], both CNN-based models were trained using a bicubic upsampler against mean squared error (MSE) loss. Table 1 tabulates the reconstruction quality as measured by three different perception-based picture quality models. The first thing to notice is that DPID [46] and the L_0-regularized algorithm [47] performed far below desired levels. It may also be observed that CNN-CR and *ProgDownLite* delivered similar levels of performance at the 2× scale. However, at the larger scale (4×), the reconstruction quality of CNN-CR was much worse than that of *ProgDownLite*, providing evidence of the efficacy of the progressive architecture. In addition to reconstruction quality, it should also be noted that CNN-CR is not applicable to non-integer scale factors, such as 1.5× and 2.5× downscaling.

4.2. Evaluation experiments

Evaluation Dataset To evaluate our method under the adaptive video streaming scenario, we collected 31 test video contents having 1080p resolution, that were obtained from two publicly available sources commonly used for evaluating video codecs, the Xiph Video Test Media[3] and Ultra Video Group (UVG) datasets [62]. We also utilized 68 representative Full High Definition (FHD) video sources from amongst the licensed movies and TV shows in the Netflix catalog, yielding more diverse and realistic contents. All of the test sequences were converted to YUV420 (YCbCr color space with 4:2:0 sampling) 8-bit format prior to evaluation. The lengths of the sequences varied between 120 to 1335 frames. Sequences which were originally longer were clipped to less than 750 frames. The test datasets that we relied on are summarized in Table 2.

Input format transformation We integrated the trained networks into a video encoding pipeline. Since TensorFlow does not provide a YUV-friendly input/output interface, the videos were input to the network in RGB888 format. Each YUV420 input was converted to RGB888. Following resolution reduction, the video frames produced by the downsampling network were converted back to their original format (YUV420) prior to encoding. We refer the reader to the Appendix for details regarding ffmpeg commands for format conversion.

Evaluation Methodology We measured the objective coding efficiency of each downsampling model within the same video encoding pipeline using the Bjøntegaard-Delta bitrate (BD-rate) [63], which quantifies average differences in bitrate at a same distortion level relative to another reference encode. To calculate BD-rate, we encoded the down-sampled videos by H.264 (x264) at 15 different QPs, ranging from 17 to 46. Then, the encoded videos were up-scaled back to their original resolutions. The performances of all of the downsamplers were compared to the same baseline — the Lanczos downsampling algorithm implemented in ffmpeg. A negative BD-rate means that the bitrate was reduced as compared with the baseline. Lastly, the distortion levels that were used for BD-rate calculation were quantified using PSNR and VMAF,[4] all calculated on the luma channel.

4.3. Overall comparison

The main performance results are given in Table 3, with respect to different objective video quality assessment models. We report the BD-rate changes obtained relative to the baseline (Lanczos downsampling under the same conditions), averaged over all the videos in the test set. We comprehensively evaluated the proposed downsampling networks for various scale factors that are commonly used in practice, using three different upsampling algorithms. From the results in the table, we may draw a number of conclusions. First, the results show that bicubic interpolation yielded worse results than did Lanczos interpolation. On the other hand, performance of the learned downsampling models surpassed those of both of the two conventional downsampling algorithms. Indeed, significant BD-rate reductions were obtained in many cases. The progressive models generally competed quite well with the CNN-CR model at integer scale factors, but performed significantly

[3] We used 24 test videos of 1080p resolution from the "HD Content and Above" category of https://media.xiph.org/video/derf/.

[4] We used VMAF 0.6.2 with the "no enhancement gain" setting [64] enabled in our evaluation. This model penalizes overly amplified image enhancements (such as sharpening or contrasting), yielding fairer results when comparing codecs.

Table 3
Per-scale performance comparison of downsampling algorithms when used in an H.264 encoding pipeline on the combined Xiph + UVG dataset and on the Netflix (NFLX) dataset. Each cell shows the average change of BD-rate expressed as percent. The baseline comparison is against Lanczos downsampling using the same encoding recipe. Smaller or negative values indicate better coding efficiency. A "—" in a cell indicates the result is not applicable to the model.

Scale	Downsampler	Xiph + UVG						NFLX					
		bilinear↑		bicubic↑		Lanczos↑		bilinear↑		bicubic↑		Lanczos↑	
		PSNR	VMAF	PSNR	VMAF	PSNR	VMAF	PSNR	VMAF	PSNR	VMAF	PSNR	VMAF
1.5×	bicubic↓	+5.84	+1.94	+2.08	+1.30	+1.43	+1.07	+2.77	+1.51	+1.75	+1.45	+1.22	+1.25
	CNN-CR↓ [48]	—	—	—	—	—	—	—	—	—	—	—	—
	ProgDown↓ (proposed)	−5.73	+0.18	−1.34	+0.14	+0.23	+0.74	−3.85	−2.14	−1.89	−1.70	−0.64	−1.08
	ProgDownLite↓ (proposed)	−5.81	−0.31	−1.30	−0.10	+0.30	+0.51	−3.75	−2.30	−1.75	−1.76	−0.47	−1.14
2×	bicubic↓	+8.13	+4.66	+3.31	+3.66	+2.62	+3.26	+3.88	+3.61	+2.69	+3.42	+2.05	+3.04
	CNN-CR↓ [48]	−5.71	−2.01	−2.96	−1.79	−0.90	−0.89	−6.00	−4.55	−3.43	−4.02	−1.55	−3.11
	ProgDown↓ (proposed)	−5.69	−2.00	−2.93	−1.79	−0.87	−0.89	−6.01	−4.58	−3.43	−4.06	−1.55	−3.14
	ProgDownLite↓ (proposed)	−5.76	−2.03	−2.98	−1.80	−1.00	+0.66	−6.02	−4.60	−3.42	−4.06	−1.52	−3.13
2.5×	bicubic↓	+9.02	+9.12	+6.07	+7.59	+3.03	+6.00	+4.31	+6.46	+3.11	+5.66	+2.48	+5.18
	CNN-CR↓ [48]	—	—	—	—	—	—	—	—	—	—	—	—
	ProgDown↓ (proposed)	−10.40	−6.96	−3.83	−5.58	−1.90	−4.22	−6.14	−7.59	−3.91	−6.74	−2.08	−5.39
	ProgDownLite↓ (proposed)	−10.41	−6.76	−3.84	−5.50	−1.92	−4.14	−6.12	−7.40	−3.82	−6.57	−2.01	−5.23
3×	bicubic↓	+8.05	+11.57	+6.39	+9.38	+3.24	+7.50	+4.54	+8.93	+3.74	+7.58	+3.27	+7.02
	CNN-CR↓ [48]	−6.19	−11.44	−4.44	−9.69	−2.42	−8.13	−6.10	−10.92	−4.55	−9.63	−2.70	−8.17
	ProgDown↓ (proposed)	−6.25	−11.95	−4.44	−10.00	−2.41	−8.42	−6.22	−11.48	−4.62	−10.02	−2.75	−8.54
	ProgDownLite↓ (proposed)	−6.21	−12.01	−4.39	−10.03	−2.35	−8.43	−6.17	−11.48	−4.54	−9.99	−2.67	−8.51
4×	bicubic↓	+4.80	+22.00	+3.53	+16.31	+3.07	+14.12	+4.87	+15.67	+3.63	+13.07	+3.14	+11.60
	CNN-CR↓ [48]	−6.98	−19.91	−3.80	−15.95	−1.17	−14.16	−6.79	−17.82	−4.00	−14.92	−1.73	−13.23
	ProgDown↓ (proposed)	−7.19	−17.71	−5.46	−14.66	−3.98	−13.28	−6.79	−14.91	−5.25	−12.90	−3.98	−11.64
	ProgDownLite↓ (proposed)	−7.15	−17.66	−5.28	−14.03	−3.84	−12.83	−6.87	−14.99	−5.24	−12.72	−3.98	−11.44
5×	bicubic↓	+9.09	+52.85	+6.99	+30.57	+3.53	+22.93	+4.88	+22.24	+3.88	+18.86	+3.50	+17.04
	CNN-CR↓ [48]	−5.71	−33.51	−3.04	−27.11	+0.28	−22.43	−5.45	−25.60	−2.92	−23.05	−0.43	−19.95
	ProgDown↓ (proposed)	−7.10	−31.81	−5.47	−27.19	−3.89	−23.85	−6.98	−23.01	−5.54	−21.50	−4.11	−19.37
	ProgDownLite↓ (proposed)	−7.05	−31.99	−5.42	−27.34	−3.81	−24.04	−7.11	−23.50	−5.55	−21.87	−4.08	−19.73

better when M was large. We also obtained reasonable improvements of BD-rates when M was not an integer, whereas CNN-CR was not applicable to these cases. It may also be observed that, when using bicubic upsampling at $M = 5$ on the public test set, the PSNR BD-rate obtained using CNN-CR was worse than the baseline scenario, possibly because it was trained without considering encoding effects, such as compression distortions and rate consumption, hence resulting in a sub-optimal result. Conversely, our two proposed models outperformed the baseline by saving more than 3.8% of bitrate, likely due to the better reconstructions yielded by the progressive architecture. Finally, given the nearly identical performance attained by the *ProgDown* and *ProgDownLite* architectures, *we recommend using the* ***ProgDownLite*** *model*. It is less computationally complex under non-integer scale factors, due to the elegant way it handles fractional resizing. Another interesting observation that can be made is that the CNN-based models delivered coding gains with respect to all of the video quality measurements. This suggests that the encoding pipeline is not only more efficient in a pixel-wise (PSNR) sense, but also perceptually optimized.

Despite training with a fixed bicubic upsampler, the models were still able to generalize well to different upsampling algorithms. However, more significant BD-rate improvements were obtained on bilinear upsampling, which usually results in worse quality, leaving greater room for improvement of objective video quality. We have also observed very similar trends using more sophisticated upscaling algorithms, including Lanczos. The performance results, however, reveal less significant improvements than bicubic perhaps due to the counter reason to bilinear upsampling.

4.4. Optimization on multi-resolution representation

In addition to analyzing model performance at single scale factors, we investigated the effects of combining encoding results over multi-resolutions to simulate the realistic mechanism of a chunk level R–D optimization. In this scenario, a source video was first downscaled to 6 different resolutions (720p, 540p, 432p, 360p, 270p, and 216p). Each resolution was further encoded using various QPs, yielding 6 R–D curves. We followed the procedure described in [65] to construct

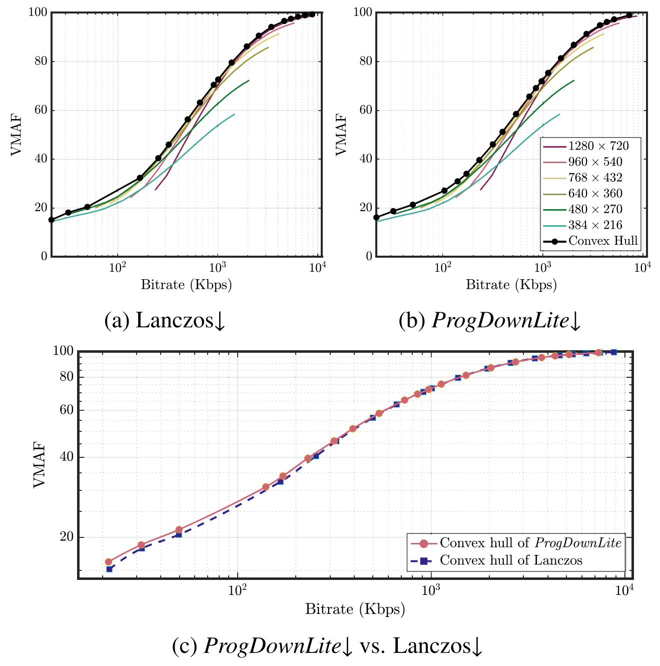


Fig. 8. An example of constructing a multiple resolution representation and convex hull of R–D curves on the *tractor* sequence using (a) Lanczos interpolation and (b) the proposed *ProgDownLite* model. (c) compares the convex hull R–D curves in (a) and (b). At each resolution, each video was encoded using x264 and upsampled back using bicubic interpolation.

the convex hull of the 6 R–D curves, and from it determined the R–D points to stream. An example of generating convex hulls from multiple representations in resolution is demonstrated in Fig. 8. In this case, a BD-rate improvement of −3.11% was achieved when comparing the

Table 4
BD-rate change (in percentage) obtained by applying a learned downsampler in several encoding pipelines relative to traditional interpolation methods on the Xiph + UVG dataset and on the Netflix (NFLX) dataset.

Upsampler	Codec	Xiph + UVG			NFLX		
		PSNR	SSIM	VMAF	PSNR	SSIM	VMAF
bilinear↑	H.264	−3.47	−3.98	−2.78	−3.90	−3.37	−4.79
	HEVC	−5.36	−3.87	−2.48	−4.52	−2.55	−3.33
	VP9	−4.81	−3.49	−3.26	−4.39	−2.11	−4.02
	AV1	−5.72	−5.13	−2.52	−4.50	−2.93	−4.05
bicubic↑	H.264	−1.40	−2.98	−3.09	−2.09	−2.14	−5.25
	HEVC	−3.11	−3.00	−3.04	−2.69	−1.67	−4.23
	VP9	−2.43	−2.50	−3.22	−2.27	−0.88	−4.25
	AV1	−3.41	−4.34	−2.70	−2.91	−1.89	−4.35
Lanczos↑	H.264	+0.20	−2.06	−2.74	−0.59	−1.08	−5.08
	HEVC	−1.21	−2.34	−2.91	−1.18	−0.83	−4.23
	VP9	−0.68	−1.62	−3.13	−0.72	−0.15	−4.04
	AV1	−1.58	−3.57	−2.41	−1.44	−1.04	−4.04

convex hulls obtained from the *ProgDownLite* models (Fig. 8(a)) against those from Lanczos (Fig. 8(b)).

In addition to H.264, we also performed the same evaluation on more advanced video codecs, HEVC (HM 16.22), VP9 (libvpx 1.8.1), and AV1 (libaom 2.0.0), with the BD-rate results reported in Table 4. The experimental results clearly show that our models still significantly outperformed Lanczos downsampling, when jointly utilized to optimize R–D performance with respect to different target quality models. Furthermore, they generalized well among the different codec standards, perhaps due to not including compression in training. Surprisingly, as may be observed, the attained BD-rates were generally better when tested on more sophisticated codecs, such as HEVC or AV1. This is likely because fewer compression artifacts arise when using an advanced video codec at a given bitrate.

4.5. Which upsampler should be used during training?

We studied the design choice of training methodology by comparing the *ProgDown* models trained with bilinear and bicubic upsampling. We denote these two models as *ProgDown*-BL and *ProgDown*-BC, respectively. To exclude the factor of video encoding, we only measured the reconstruction performance of different downsampling–upsampling combinations, with results reported in Table 5. The first thing to notice is the performance attained by each testing upsampler. As expected, the best performances were achieved when using the same upsampler in training and testing. Despite being the top-performer when testing with bilinear, the *ProgDown*-BL model failed on many cases when testing with bicubic. One counter-intuitive example is that, applying bicubic upsampling on frames downsampled by *ProgDown*-BL performed worse than using bilinear upsampling, which suggests the possibility of overfitting. In addition, *ProgDown*-BL was worse than the Lanczos baseline when measured against PSNR, making it hard to justify its use in practical applications. By contrast, models trained with bicubic upsampling (*ProgDown*-BC) transferred very well to both upsamplers, gaining more than 0.5 dB over Lanczos in many cases. Therefore, we argue that using bicubic interpolation in training appears to yield better generalizability.

Moreover, Fig. 9 shows a visual comparison of training strategies. We downsampled the source frame by 2× using different downsampling models, then upsampled back to the original resolution via bicubic interpolation. It may be observed that *ProgDown*-BC (4th column) effectively learned a better low-resolution representation than did the Lanczos method, leaving textures and edges well preserved after the upsampling process. However, close examination reveals that *ProgDown*-BL (3rd column) tended to over-sharpen textures as compared to the pristine source, which could explain its reduced objective performance. In fact, the results from Table 5 and Fig. 9 can be understood by the nature of the upsampling algorithms. Simple bilinear averaging is low-pass, hence causes blur. Thus, the network trained with respect to bilinear upsampling learns to preserve details to compensate for these smoothing effects.

Table 5
Comparison of models trained with different upsamplers on the UVG dataset. Each cell shows the average reconstructed video quality, expressed as PSNR / SSIM / VMAF. The worst performance is highlighted in **red boldface**.

Scale	Downsampler	Test with bilinear↑	Test with bicubic↑
2×	Lanczos↓	**41.015** / **0.9672** / **86.497**	42.740 / 0.9739 / 95.468
	ProgDown-BL↓	43.366 / 0.9769 / 96.406	**41.026** / **0.9697** / **94.376**
	ProgDown-BC↓	42.036 / 0.9729 / 88.345	43.323 / 0.9739 / 95.690
3×	Lanczos↓	**37.693** / **0.9397** / **76.731**	38.828 / **0.9481** / **88.397**
	ProgDown-BL↓	38.738 / 0.9490 / 90.827	**38.745** / 0.9510 / 89.806
	ProgDown-BC↓	38.401 / 0.9470 / 81.346	39.399 / 0.9529 / 92.210
5×	Lanczos↓	**33.792** / **0.8848** / **47.233**	34.685 / **0.8952** / **64.576**
	ProgDown-BL↓	34.787 / 0.8974 / 72.677	**34.501** / 0.8992 / 71.327
	ProgDown-BC↓	34.448 / 0.8948 / 57.678	35.149 / 0.9021 / 75.086

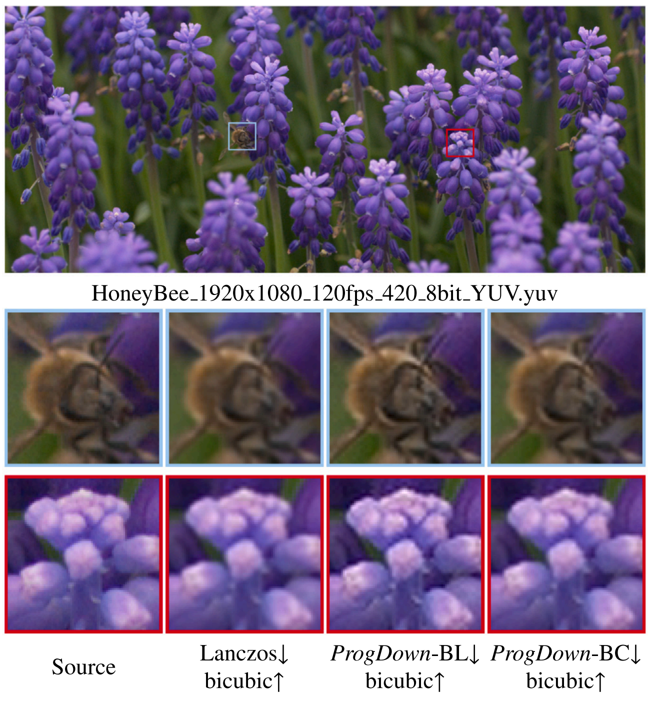


Fig. 9. Visual comparison of 2× models trained with different upsamplers on *HoneyBee* of the UVG dataset. All the downsampled frames were upsampled by 2× with bicubic interpolation before display.

4.6. Determining the weight on guided loss

As described in Section 3.5, we use an additional guided loss function to prevent corner cases where artifacts can arise. To determine the

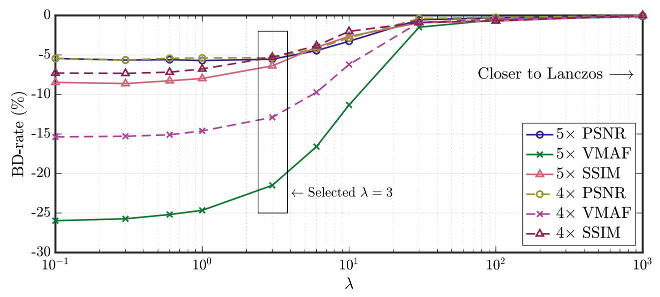


Fig. 10. Averaged BD-rate of *ProgDownLite* for 4× and 5× downsampling as a function of the weight parameter λ.

Table 6
Runtime comparison of downsampling algorithms using different scale factors. All computational speeds are given in frames per second (fps).

Model/Factor		1.5×	2×	2.5×	3×	4×	5×
ffmpeg bicubic↓	CPU	259.7	354.3	328.6	385.1	452.5	483.2
ffmpeg Lanczos↓	CPU	167.9	229.5	266.1	286.2	338.8	383.4
CNN-CR↓ [48]	CPU	–	1.073	–	1.492	1.717	1.839
	GPU	–	1.379	–	1.692	1.767	1.849
ProgDown↓	CPU	0.347	0.827	0.379	0.909	0.968	0.939
	GPU	0.761	1.250	0.836	1.426	1.452	1.522
ProgDownLite↓	CPU	0.638	0.758	0.833	0.853	0.915	0.943
	GPU	1.047	1.253	1.376	1.422	1.492	1.519

weight on the guided loss in (10), we uniformly sampled ten values ranging along a log scale from 0 to 10^{10}. Fig. 10 plots the averaged BD-rate on a validation set as a function of the weight parameter λ. Larger values of λ drives the model towards Lanczos downsampling, yielding less BD-rate improvement. Overall, we found that fixing $\lambda = 3$ yielded the lowest SSIM/VMAF BD-rate, while maintaining the same level of PSNR BD-rate, thereby preserving both objective (metric) performance while mitigating perceptual artifacts.

4.7. Execution time

The execution times of the various downscaling models at different scale factors are summarized in Table 6. The results were calculated by averaging the runtime (in terms of fps) over all 7 1080p test videos from the UVG dataset on the same machine equipped with 16 logical Intel Xeon Platinum 8259CL CPUs@2.50 GHz, 128G RAM, and an NVIDIA Tesla K80 GPU (12G Memory version). We used the median value from 5 runs for each content. Note that the pixel format conversion and model loading times are included for all of the CNN-based models. From Table 6, it may be observed that the compute speed increased with downsampling, since fewer pixels were processed. As comparing with the *ProgDown* model, the complexity of the *ProgDownLite* model was significantly less at fractional scaling factors, since the network does not increase resolution. Of course, the runtimes of the deep models would be significantly reduced if implemented on a GPU.

Since we used the publicly-available precompiled version of Tensorflow, which may not be optimal on every machine, we would expect further accelerations to be possible by re-compiling with a low-level instruction set, or by utilizing optimizers such as Intel's Math Kernel Library. In addition to inferencing speed, integrating the trained CNN models into optimized multimedia pipelines, such as ffmpeg, could reduce end-to-end execution time. It should also be noted that, even given the acceleration of modern GPUs, the CNN-based downsampling models are still too slow for high-throughput applications, such as live video streaming. However, these processing speeds are still acceptable for most non-realtime HTTP Adaptive Streaming applications.

Table 7
Summary of the subjective study design.

	Group A	Group B
# Contents	8	8
# Subjects	39	39
Encoding Resolutions	{720p, 432p, 360p}/{540p, 432p, 360p}	
Encoding QPs	∼ {80, 105, 130}	
# Comparisons (sess. 1)	48	48
# Comparisons (sess. 2)	60	60
# Comparisons (total)	108	108

4.8. Subjective study and analysis

Since humans are the ultimate receiver of streamed videos, we conducted a human subject study to better understand perceptual preferences of the different downsampling algorithms. We selected 16 source videos from the Netflix open content[5] library, as well as from licensed Netflix titles. The source videos exhibit a variety of characteristics, such as different amounts of local motion, dynamic textures, simple/complex camera movements, and so on. None of the videos contain audio components. Ten of the videos are of resolution 3840 × 2160, while the remaining six contents are of resolution 1920 × 1080. We divided 74 participants into two groups, and assigned 8 distinct video contents to each group. Each content was downsampled to three encoding resolutions, then encoded at three different bitrates by VP9.[6] To fairly compare videos at equal bitrates, we encoded the videos using a 2D grid of QP values $(\mathrm{QP}_{ProgDownLite}, \mathrm{QP}_{\mathrm{Lanczos}}) \in \{k, k \pm 1, \ldots, k{\pm}5\} \times \{k, k{\pm}1, \ldots, k{\pm}10\}$, where $k \in \{80, 105, 130\}$. For each value of k, we selected QP pairs that minimized percent bitrate differences, while guaranteeing that *ProgDownLite* would always associated with a smaller bitrate than Lanczos. Among all the distorted video pairs, we obtained a maximum bitrate deviation of 1.12% between two downsamplers.

As summarized in Table 7, we obtained 9 different scaling-compression distorted versions of that particular content. To test the model generalizability to different upsamplers, we randomly applied either bicubic interpolation or a deep super-resolution model, VDSR [42], to each content. We followed standard practice to divide the subjective study into several separate viewing sessions, to avoid visual fatigue. Each subject's sessions were separated by at least 24 hours for the same reason. After viewing each video pair, a subject was asked to select the one they preferred. Among the 74 recruited volunteer participants, about 51% were somewhat knowledgeable about image/video processing, while the others were naive. In sum, we asked each participant to compare 72 pairs of videos divided into two sessions, each of which lasted about 20 minutes.

Since compressed videos obtained using different downsampling algorithms often have subtle perceptual differences at similar bitrates, it can be easier for humans to make comparisons between simultaneously displayed videos. Therefore, we adopted a double stimulus *paired comparison* method instead of the Absolute Category Rating (ACR) protocol. The videos processed by Lanczos and *ProgDownLite* were cropped to resolution 960 × 1080 and synchronously played on the same monitor so that the subjects could view and compare them. We designed a web-based user interface to carry out the human study, whereby participants could easily compare pairs of videos and render relative quality decisions. To obtain the human opinion scores, a straightforward strategy is to deploy the Bradley–Terry (BT) model [66]. For each video content impaired to create d different distorted versions, one can estimate the Mean Opinion Score (MOS) from $\binom{d}{2}$ comparisons. However, the large number of comparisons needed by such a method would limit the

[5] https://opencontent.netflix.com/home

[6] We used the proprietary Eve-VP9 encoder in the subjective study.

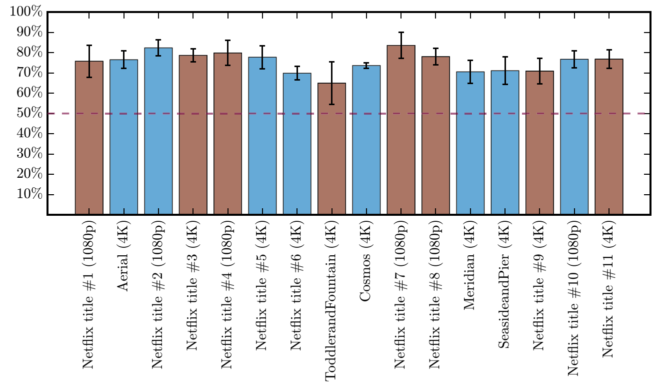


Fig. 11. Subjective performance of average win rate of *ProgDownLite* (with 95% confidence intervals) for each content. Blue: contents upsampled by bicubic interpolation; brown: contents upsampled by VDSR.

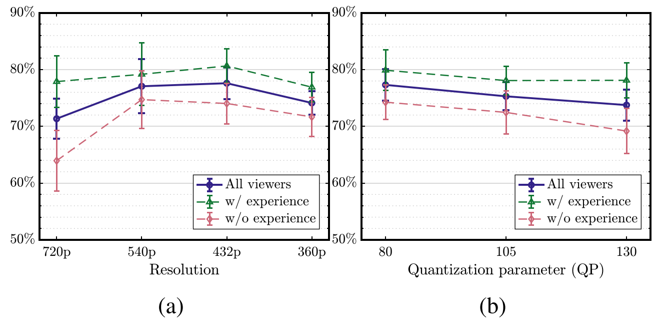


Fig. 12. Subjective performance of average win rate of *ProgDownLite* with respect to (a) encoding resolution (b) encoding QP. The error bars indicate the 95% confidence intervals.

diversity of contents and distortions in a practical study. In light of this, we chose to report the win rate of the head-to-head comparisons at the same bitrate at which users preferred our method over the Lanczos baseline.

Given the collected subjective comparisons, we analyzed the results by computing the percentage of subjects preferring *ProgDownLite* over Lanczos, and plotted the results with respect to each content in Fig. 11. On average, *ProgDownLite* was preferred by 76.8% of the subjects. It may be observed that, on some specific video contents, *ProgDownLite* only obtained around 66% of the votes, indicating that performance close to that of Lanczos. This could be due to temporal masking effects arised from large motions or camera panning. It is also interesting to note that our model still delivered superior performance on the 2160p contents, although the low-resolution encodings were upsampled and displayed at 1080p in our experiment. We have also observed similar performance of the two different upsampling models, bicubic interpolation and VDSR, further validating the model generalizability.

Towards better understanding the downsampling models, we also analyzed their performances against encoding resolution and QP, as shown in Fig. 12. The lowest performance occurred on the 720p encodings (Fig. 12(a)). This could be because both distorted videos were of very high quality, whereby some subjects may have been unable to distinguish differences in subjective quality. There was one anomaly on the other end, where the win rate of *ProgDownLite* dropped drastically at 360p. When interviewing the subjects after finishing two sessions, we learned that some disliked the jaggedness artifacts that occasionally appeared on the edges of objects in videos encoded at 360p using *ProgDownLite*, although they were sharper than those produced using Lanczos. Interestingly, Fig. 12(b) shows that the win rate of *ProgDownLite* slightly decreased as the QP was raised. Overall, these results suggest that *ProgDownLite* is able to yield favorable perceptual video quality relative to the widely deployed Lanczos algorithm.

Table 8
Outcomes of significance tests of the relative performances of *ProgDownLite* against Lanczos interpolation, on each of 16 video contents encoded at three QP/resolutions (see text in Section 4.9).

QP / Res.	720p / 540p	432p	360p
80	1111111 -111 -1111	1111111111111111	-11111111111111 -1
105	11111 -1 -11111 -11	111111 - -111 - -111	-11111111 -111111
130	1 -11 -1111111 - -11	111111 - -111 - -111	-111111 -11111 -11

4.9. Significance test

To evaluate whether our model performed significantly better than Lanczos, we performed a 95% two-tailed z-test with the Wilson score interval [67] on the votes obtained from each distorted video (also known as inference for one proportion). For each video pair, we tested whether the win rate of *ProgDownLite* was significantly different from the null hypothesis H_0 at a level $\hat{p} = 0.5$. Table 8 shows the results of the statistical significance tests. Each cell in the table consists of 16 symbols, corresponding to the 16 selected video contents. A value of '1' in the table indicates that *ProgDownLite* was statistically superior to Lanczos, while a value of '0' means the opposite. A value of '-' indicates there was no statistical difference between the two downsampling algorithms. From the results shown in the Table, it may be observed that *ProgDownLite* statistically surpassed Lanczos on more than 83% of the cases considered.

5. Conclusion and future work

In this paper, we proposed a progressive residual learning network architecture for video downsampling in streaming. In particular, we explored two different ways to address non-integer scale factors, resulting in models called *ProgDown* and *ProgDownLite*.[7] We believe that the ideas we discovered in this work can applied to other image transformation problems. For instance, within the *ProgDownLite* model, we developed a new convolutional block that allows fractional resizing, which is simple and can be effectively implemented. With proper modifications, it should also be applicable to super-resolution problems. To achieve better performance and faster processing speed, different machine learning models, such as GANs [69] and Transformers [70] could be used to define resizing modules.

The current training framework, which minimizes the reconstruction error, is not perfect. We have observed that CNN-downsampled videos generally exhibit more details, enlarging bit consumption at similar QP values. This suggests that there is room for improving rate–distortion tradeoffs, perhaps by applying a "rate term" in the loss function. Our subjective investigation in Section 4.6 also indicates that widely-used VQA models are not capable of capturing temporal flicker artifacts. By designing new features, it is possible to achieve more accurate video quality prediction and further improve loss functions for training. Looking further ahead, we plan to extend the concept of learned preprocessing to other scenarios, such as mitigating banding artifacts [71] and high bit-depth HDR pipelines. We are also aware of the emergence of consumer televisions having more sophisticated upsampling algorithms. Therefore, verifying the efficacy of approaches like these on emerging high-end devices will be an important task.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

[7] The *ProgDownLite* model has been further optimized and is currently running at scale on the Netflix service [68].

Data availability

No data was used for the research described in the article.

Acknowledgments

This work is supported by Netflix and by grant number 2019844 for the National Science Foundation AI Institute for Foundations of Machine Learning (IFML). The authors thank the Video Algorithms team at Netflix for providing fruitful feedback on this work, and the Machine Learning Infrastructure team at Netflix for providing technical support on Metaflow (https://metaflow.org/). The authors would also like to thank all the volunteers who took part in the subjective study. The human study was conducted under the approval of the Institutional Review Board (IRB) protocol 2007-11-0066.

Appendix

In Section 4.2, we use the following ffmpeg commands for conversion between YUV420 and RGB888 with minimal introduced numerical error:

YUV to RGB conversion. To mitigate the reconstruction error in the luma channel, the source video frames were first converted to YUV444 format

```
ffmpeg -s WxH -pix_fmt yuv420p -i i420.yuv \
        -pix_fmt yuv444p i444.yuv
```

Then, the intermediate `i444.yuv` was converted to RGB888 by

```
ffmpeg -s WxH -pix_fmt yuvj444p -i i444.yuv \
```

where `i420.yuv` and `i888%08d.png` are source videos in YUV420 and RGB888 format (input to the downsampling models), respectively, and `W` and `H` are the dimensions of the source video.

RGB to YUV conversion. The downsampled video frames `cnn%08d.png` were converted back to their original format (YUV420) prior to encoding via

```
ffmpeg -i cnn%08d.png -pix_fmt yuvj420p \
             o420.yuv
```

where `o420.yuv` is the downsampled video in YUV420 format. We set `-pix_fmt` to `yuvj444p` / `yuvj420p` in order to avoid the conversion between limited range and full range.

References

[1] C. Chen, Y.-C. Lin, S. Benting, A. Kokaram, Optimized transcoding for large scale adaptive streaming using playback statistics, in: Proc. IEEE Int. Conf. Image Process., 2018, pp. 3269–3273.
[2] I. Katsavounidis, Dynamic optimizer – a perceptual video encoding optimization framework, 2018, The NETFLIX Tech Blog. URL https://netflixtechblog.com/dynamic-optimizer-a-perceptual-video-encoding-optimization-framework-e19f1e3a277f.
[3] P.-H. Wu, V. Kondratenko, I. Katsavounidis, Fast encoding parameter selection for convex hull video encoding, in: Proc. SPIE Applications Digital Image Process. XLIII, 2020.
[4] Z. Wang, A. Bovik, H. Sheikh, E. Simoncelli, Image quality assessment: From error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.
[5] H.R. Sheikh, A.C. Bovik, Image information and visual quality, IEEE Trans. Image Process. 15 (2) (2006) 430–444.
[6] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, M. Manohara, Toward a practical perceptual video quality metric, 2016, The NETFLIX Tech Blog. URL https://netflixtechblog.com/toward-a-practical-perceptual-video-quality-metric-653f208b9652.
[7] H.C. Burger, C.J. Schuler, S. Harmeling, Image denoising: Can plain neural networks compete with BM3D? in: Proc. IEEE Conf. Comput. Vision Pattern Recog., 2012, pp. 2392–2399.
[8] J. Ballé, V. Laparra, E.P. Simoncelli, End-to-end optimized image compression, in: Proc. Int. Conf. Learn. Represent., 2017, pp. 1–27.
[9] S. Paul, A. Norkin, A.C. Bovik, Speeding up VP9 intra encoder with hierarchical deep learning-based partition prediction, IEEE Trans. Image Process. 29 (2020) 8134–8148.
[10] L.-H. Chen, C.G. Bampis, Z. Li, C. Chen, A.C. Bovik, Convolutional block design for learned fractional downsampling, in: Proc. IEEE Asilomar Conf. on Signals, Syst., and Comput., 2022, pp. 640–644.
[11] Apple, HLS authoring specification for apple devices, 2017.
[12] A. Aaron, Z. Li, M. Manohara, J.D. Cock, D. Ronca, Per-title encode optimization, 2015, The NETFLIX Tech Blog. URL https://netflixtechblog.com/per-title-encode-optimization-7e99442b62a2.
[13] M. Bhat, J.-M. Thiesse, P.L. Callet, Can small be beautiful?: Assessing image resolution requirements for mobile tv, in: Proc. 13th Annu. ACM Int. Conf. Multimedia, 2005, pp. 829–838.
[14] G. Cermak, M. Pinson, S. Wolf, The relationship among video quality, screen resolution, and bit rate, IEEE Trans Broadcast. 57 (2) (2011) 258–262.
[15] G. Georgis, G. Lentaris, D. Reisis, Reduced complexity superresolution for low-bitrate video compression, IEEE Trans. Circuits Syst. Video Technol. 26 (2) (2016) 332–345.
[16] L. Toni, R. Aparicio-Pardo, K. Pires, G. Simon, A. Blanc, P. Frossard, Optimal selection of adaptive streaming representations, ACM Trans. Multimedia Comput. Commun. Appl. 11 (2s) (2015) 1–26.
[17] C. Li, L. Toni, P. Frossard, H. Xiong, J. Zou, Complexity constrained representation selection for dynamic adaptive streaming, in: Proc. IEEE Visual Commun. Image Process., 2016.
[18] Y. Sani, A. Mauthe, C. Edwards, Adaptive bitrate selection: A survey, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2985–3014.
[19] M. Shen, P. Xue, C. Wang, Down-sampling based video coding using super-resolution technique, IEEE Trans. Circuits Syst. Video Technol. 21 (6) (2011) 755–765.
[20] X. Li, N. Oertel, A. Hutter, A. Kaup, Laplace distribution based Lagrangian rate distortion optimization for hybrid video coding, IEEE Trans. Circuits Syst. Video Technol. 19 (2) (2009) 193–205.
[21] M. Bhat, J.-M. Thiesse, P.L. Callet, A case study of machine learning classifiers for real-time adaptive resolution prediction in video coding, in: Proc. IEEE Int. Conf. Multimedia Expo, 2020, pp. 1–6.
[22] M. Afonso, F. Zhang, D.R. Bull, Video compression based on spatio-temporal resolution adaptation, IEEE Trans. Circuits Syst. Video Technol. 29 (1) (2019) 275–280.
[23] F. Zhang, M. Afonso, D.R. Bull, ViSTRA2: Video coding using spatial resolution and effective bit depth adaptation, Signal Process., Image Commun. 97 (2021) 116355.
[24] Z. Wang, A. Bovik, Mean squared error: Love it or leave it? A new look at signal fidelity measures, IEEE Signal Process. Mag. 26 (1) (2009) 98–117.
[25] K. Seshadrinathan, A. Bovik, Motion tuned spatio-temporal quality assessment of natural videos, IEEE Trans. Image Process. 19 (2) (2010) 335–350.
[26] P.V. Vu, C.T. Vu, D.M. Chandler, A spatiotemporal most-apparent-distortion model for video quality assessment, in: Proc. IEEE Int. Conf. Image Process., 2011, pp. 2505–2508.
[27] M. Pinson, S. Wolf, A new standardized method for objectively measuring video quality, IEEE Trans. Broadcast. 50 (3) (2004) 312–322.
[28] S. Wolf, Variable frame delay (VFD) parameters for video quality measurements, 2011, U.S. Dept. Commer., Nat. Telecommun. Inf. Admin., Boulder, CO, USA, Tech. Memo TM-11-475.
[29] M.H. Pinson, L.K. Choi, A.C. Bovik, Temporal video quality model accounting for variable frame delay distortions, IEEE Trans. Broadcast. 60 (4) (2014) 637–649.
[30] A. Hekstra, J. Beerends, D. Ledermann, F. de Caluwe, S. Kohler, R. Koenen, S. Rihs, M. Ehrsam, D. Schlauss, PVQM – A perceptual video quality measure, Signal Process., Image Commun. 17 (10) (2002) 781–798.
[31] P. Tao, A.M. Eskicioglu, Video quality assesment using M-SVD, in: Proc. SPIE, Vol. 6494, 2007, 649408.
[32] R. Soundararajan, A.C. Bovik, Video quality assessment by reduced reference spatio-temporal entropic differencing, IEEE Trans. Circuits Syst. Video Technol. 23 (4) (2013) 684–694.
[33] Z. Tu, Y. Wang, N. Birkbeck, B. Adsumilli, A.C. Bovik, UGC-VQA: Benchmarking blind video quality assessment for user generated content, IEEE Trans. Image Process. 30 (2021) 4449–4464.
[34] Z. Wang, E.P. Simoncelli, A.C. Bovik, Multi-scale structural similarity for image quality assessment, in: Proc. IEEE Asilomar Conf. on Signals, Syst., and Comput., 2003, pp. 1398–1402.
[35] R. Keys, Cubic convolution interpolation for digital image processing, IEEE Trans. Acoust. Speech Signal Process. 29 (6) (1981) 1153–1160.
[36] D. Glasner, S. Bagon, M. Irani, Super-resolution from a single image, in: Proc. IEEE Int. Conf. Comput. Vision, 2009.
[37] A. Singh, N. Ahuja, Super-resolution using sub-band self-similarity, in: Proc. Asia Conf. Comput. Vision, 2015, pp. 552–568.
[38] W. Freeman, T. Jones, E. Pasztor, Example-based super-resolution, IEEE Comput. Graph. Appl. 22 (2) (2002) 56–65.
[39] K.I. Kim, Y. Kwon, Single-image super-resolution using sparse regression and natural image prior, IEEE Trans. Pattern Anal. Mach. Intell. 32 (6) (2010) 1127–1133.

[40] C. Dong, C.C. Loy, K. He, X. Tang, Image super-resolution using deep convolutional networks, IEEE Trans. Pattern Anal. Mach. Intell. 38 (2) (2016) 295–307.
[41] Z. Wang, D. Liu, J. Yang, W. Han, T. Huang, Deep networks for image super-resolution with sparse prior, in: Proc. IEEE Int. Conf. Comput. Vision, 2015.
[42] J. Kim, J.K. Lee, K.M. Lee, Accurate image super-resolution using very deep convolutional networks, in: Proc. IEEE Conf. Comput. Vision Pattern Recog., 2016, pp. 1646–1654.
[43] Y. Romano, J. Isidoro, P. Milanfar, RAISR: Rapid and accurate image super resolution, IEEE Trans. Comput. Imaging 3 (1) (2017) 110–125.
[44] J. Kopf, A. Shamir, P. Peers, Content-adaptive image downscaling, ACM Trans. Graph. 32 (6) (2013) 1–8.
[45] A.C. Öztireli, M. Gross, Perceptually based downscaling of images, ACM Trans. Graph. 34 (4) (2015) 1–10.
[46] N. Weber, M. Waechter, S.C. Amend, S. Guthe, M. Goesele, Rapid, detail-preserving image downscaling, ACM Trans. Graph. 35 (6) (2016) 1–6.
[47] J. Liu, S. He, R.W.H. Lau, L_0 -Regularized image downscaling, IEEE Trans. Image Process. 27 (3) (2018) 1076–1085.
[48] Y. Li, D. Liu, H. Li, L. Li, Z. Li, F. Wu, Learning a convolutional neural network for image compact-resolution, IEEE Trans. Image Process. 28 (3) (2019) 1092–1107.
[49] H. Kim, M. Choi, B. Lim, K.M. Lee, Task-aware image downscaling, in: Proc. Eur. Conf. Comput. Vision, 2018, pp. 399–414.
[50] W. Sun, Z. Chen, Learned image downscaling for upscaling using content adaptive resampler, IEEE Trans. Image Process. 29 (2020) 4027–4040.
[51] Di Ma, F. Zhang, D.R. Bull, Video compression with low complexity CNN-based spatial resolution adaptation, in: Proc. SPIE Applications Digital Image Process. XLIII, 2020.
[52] E. Bourtsoulatze, A. Chadha, I. Fadeev, V. Giotsas, Y. Andreopoulos, Deep video precoding, IEEE Trans. Circuits Syst. Video Technol. 30 (12) (2020) 4913–4928.
[53] H. Talebi, P. Milanfar, Learning to resize images for computer vision tasks, 2021, arXiv preprint arXiv:2103.09950.
[54] A. Odena, V. Dumoulin, C. Olah, Deconvolution and checkerboard artifacts, Distill (2016) URL https://distill.pub/2016/deconv-checkerboard/.
[55] M. Jaderberg, K. Simonyan, A. Zisserman, koray kavukcuoglu, Spatial transformer networks, in: Proc. Adv. Neural Inf. Process. Syst., 2015, pp. 2017–2025.
[56] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: Proc. Int. Conf. Learn. Represent., 2015, pp. 1–15.
[57] E. Agustsson, R. Timofte, NTIRE 2017 challenge on single image super-resolution: Dataset and study, in: Proc. IEEE Conf. Comput. Vision Pattern Recog. Workshops, IEEE, 2017, pp. 1122–1131.
[58] M. Bevilacqua, A. Roumy, C. Guillemot, M.L. Alberi-Morel, Combining full-reference image visual quality metrics by neural network, in: Proc. Brit. Mach. Vis. Conf., 2012, pp. 1–10.
[59] R. Zeyde, M. Elad, M. Protter, On single image scale-up using sparse-representations, in: Proc. Int. Conf. Curves Surfaces, 2010, pp. 711–730.
[60] J.-B. Huang, A. Singh, N. Ahuja, Single image super-resolution from transformed self-exemplars, in: Proc. IEEE Conf. Comput. Vision Pattern Recog., 2015, pp. 5197–5206.
[61] C.-Y. Yang, M.-H. Yang, Fast direct super-resolution by simple functions, in: Proc. IEEE Int. Conf. Comput. Vis., 2013, pp. 561–568.
[62] A. Mercat, M. Viitanen, J. Vanne, UVG dataset: 50/120fps 4K sequences for video codec analysis and development, in: Proc. ACM Multimedia Syst. Conf., 2020.
[63] G. Bjøntegaard, Calculation of average PSNR differences between RD-curves, in: Document VCEG-M33, ITU-T Video Coding Experts Group (VCEG) Thirteenth Meeting, 2001.
[64] Z. Li, K. Swanson, C. Bampis, LukaśǨrasula, A. Aaron, Toward a better quality metric for the video community, 2020, The NETFLIX Tech Blog. URL https://netflixtechblog.com/toward-a-better-quality-metric-for-the-video-community-7ed94e752a30.
[65] A. Ortego, K. Ramchandran, Rate-distortion methods for image and video compression, IEEE Signal Process. Mag. 15 (6) (1998) 23–50.
[66] R.A. Bradley, M.E. Terry, Rank analysis of incomplete block designs: The method of paired comparisons, Biometrika 39 (3–4) (1952) 324–345.
[67] E. Wilson, Probable inference, the law of succession, and statistical inference, J. Amer. Stat. Assoc. 22 (158) (1927) 209–212.
[68] C.G. Bampis, L.-H. Chen, Z. Li, For your eyes only: improving netflix video quality with neural networks, 2022, The NETFLIX Tech Blog. URL https://netflixtechblog.com/for-your-eyes-only-improving-netflix-video-quality-with-neural-networks-5b8d032da09c.
[69] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Proc. Adv. Neural Inf. Process. Syst., 2014, pp. 2672–2680.
[70] A. Dosovitskiy, L. Beyer, A. Kolesnikov, X.Z. Dirk Weissenborn, M.D. Thomas Unterthiner, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, N. Houlsby, An image is worth 16 × 16 words: Transformers for image recognition at scale, in: Proc. Int. Conf. Learn. Represent., 2021, pp. 1–22.
[71] Z. Tu, J. Lin, Y. Wang, B. Adsumilli, A.C. Bovik, Adaptive debanding filter, IEEE Signal Process. Lett. 27 (2020) 1715–1719.