Online-trained Upsampler for Deep Low Complexity Video Compression

Jan P. Klopp
National Taiwan University
kloppjp@gmail.com

Keng-Chi Liu
Taiwan AI Labs
calvin89029@gmail.com

Shao-Yi Chien
National Taiwan University
sychien@ntu.edu.tw

Liang-Gee Chen
National Taiwan University
lgchen@ntu.edu.tw

Abstract

Deep learning for image and video compression has demonstrated promising results both as a standalone technology and a hybrid combination with existing codecs. However, these systems still come with high computational costs. Deep learning models are typically applied directly in pixel space, making them expensive when resolutions become large.

In this work, we propose an online-trained upsampler to augment an existing codec. The upsampler is a small neural network trained on an isolated group of frames. Its parameters are signalled to the decoder. This hybrid solution has a small scope of only 10s or 100s of frames and allows for a low complexity both on the encoding and the decoding side.

Our algorithm works in offline and in zero-latency settings. Our evaluation employs the popular x265 codec on several high-resolution datasets ranging from Full HD to 8K. We demonstrate rate savings between 8.6% and 27.5% and provide ablation studies to show the impact of our design decisions. In comparison to similar works, our approach performs favourably.

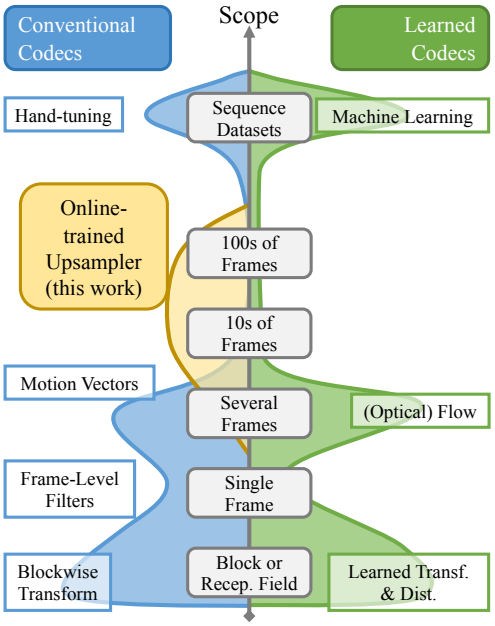

Figure 1. **Exploitation of different Scopes**. Qualitative degree and key methods that exploit redundancies in various scopes for conventional and learned codecs.

1. Introduction

Since the introduction of deep learning to image and video compression, steady improvements have led to learning algorithms outperforming many commonly used codecs. However, their principles are similar to those of conventional codecs. The main algorithmic drivers are modelling the distribution within a receptive field, sometimes also its neighbours, and the motion between two successive images. A large-scale video corpus enables learning the necessary functions. On the other hand, conventional codecs are hand-tuned and use block-wise transforms and predictions together with motion estimation.

This work proposes a method to exploit an extended scope of 10s or 100s of frames using an online-trained upsampler. The online-trained design has the advantage that computational expenses can be kept low, especially on the decoding side, compared to deep learning compression models or deep denoisers. The latter two may require 100K operations for a single pixel, while we add less than 600 operations to the conventional codec, which in turn requires only several dozens of operations.

Compared to how machine learning is most often applied, the scope of data we consider is small. However, it is large compared to what conventional codecs typically deal with (see Fig. 1 for a qualitative overview). Note that there are exceptions for some data, for example, a sequence of

images with little change where the first image is as good a predictor for the last image as any other. In such a case, a codec would implicitly cover a more extensive scope. However, this does not hold for significant motion or evolving textures like particle systems (water, smoke, fire).

We use a conventional codec to provide compression for a lower resolution signal to deal with these more difficult redundancies. At the same time, our proposed upsampler is trained on a group of frames to reconstruct the high-resolution signal efficiently. The encode then signals the upsampler's parameters to the decoder. They typically make up a small fraction of the bitstream and therefore have little effect on the coding gain.

Our experiments cover the offline setting used for conventional non-interactive video transmission and the zero-latency setting, which is suitable for interactive environments.

In summary, we

- propose a new architecture for an online-trained upsampler incorporating internal features and position encoding,
- demonstrate significant improvement over the commonly used x265 codec,
- and conduct extensive ablation studies to show how our proposed approach compares under different encoding settings.

2. Related Work

2.1. Learning-based Compression

Learning-based compression replaces each stage of conventional codecs (except entropy coding) with a differentiable equivalent from the deep learning domain. Their advantage is that they are conceptually more straightforward as their adaptation to signal statistics is learned and not designed manually. They pay for this with higher complexity in terms of operations per pixel. Early approaches [39, 40, 3] were simply image-based and grew more sophisticated by adding latent prediction strategies to allow modelling dependencies across larger receptive fields [27, 21, 4, 14, 32, 18, 2]. Given these approaches, adding differentiable optical flow-like motion estimation models can make for a simple video codec.

More complex video coding models abstract from explicit motion compensation and let the model decide what should be contained in the state that is forwarded from one frame to the next [37, 6, 23, 1, 44, 22, 33, 24, 11, 31]. These codecs typically operate only on two successive frames, giving them a low latency advantage.

This approach can also be transferred to conventional compression where a CNN is used to predict the codec's internal image representation directly, thereby tapping into the availability of accelerators while still using a standard codec, which allows efficient decoding [36, 35, 12]. The learned codec approaches are used to solve the same problem as we are. Nevertheless, there is no direct overlap. Our approach could easily be combined with learning-based compression techniques.

2.2. Learning-based Augmentation of Conventional Compression

Conventional compression can be augmented with machine learning by using an offline-trained model that reduces noise in the decoded signal. As mentioned in the introduction, such a model has to deal with both the codec's non-stationary noise distribution and natural image statistics at the same time, leading to huge models such as [25, 13, 47, 43] with over 200K Op/pixel. This complexity is two to three orders of magnitude above our approach. As argued in the introduction, this becomes difficult to implement in practice for high-resolution video.

Another promising approach is to adapt filters at encoding time instead. Lam et al. [19, 20] have proposed approaches that adapt network weights to the content and signal those weights. Their approach is similar to ours. However, the networks employed have many layers and take up a similarly large number of computations to offline-trained networks. [16] is conceptually even closer to us in that they focus on very low complexity and signal weights to the decoder. Unlike our method, they use the reconstructed signal with a feed-forward network structure, achieving lower coding gains than the proposed algorithm. Also, they do not handle the zero-latency case.

Finally, different from these approaches, [34, 7, 38, 17] have proposed different pre-filtering techniques that are applied prior to encoding to reduce coding cost without altering the decoder.

3. Online-trained Upsampler

Our approach augments an existing codec with an upsampling neural network trained on a group of successive frames. This strategy has the distinct advantage that the deep neural network responsible for the upsampling process requires a much lower complexity than one that has to deal with all possible content. With the already low complexity inherent to the conventional video codecs, this combined system requires less computation than an end-to-end differentiable convolutional neural network pipeline or a pre-trained denoiser.

In our design, the neural network takes on the task of super-resolving a decoded image instead of simply denoising it. There are two primary motivations behind this approach. For one, (conventional) codecs have difficulties capturing redundancies at an increasingly higher resolution

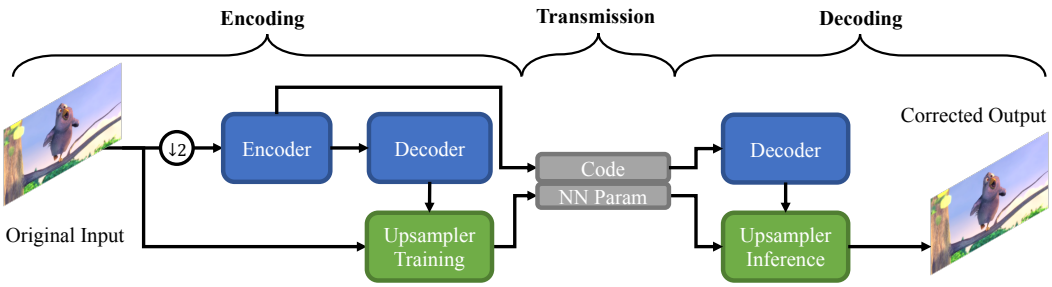

Figure 2. **Algorithm overview.** Simplified operation of our compression algorithm. The conventional codec (blue boxes) is applied to the downsampled signal, while the neural network (green boxes) is responsible for restoring the original resolution. It is trained at encoding time, and its parameters are signalled along with the conventional codec's code. At decoding time, the conventional decoder is invoked first. The upsampler takes the decoded signal and computes the final output in the original resolution.

(see Ohm et al. [29] for some experimental results on this issue). Secondly, because the conventional codec operates only on $^1/_4$th of the pixels, there is a potential for reducing the encoding time, whereas a denoiser would necessarily increase it.

The main obstacle for denoising is the resolution of ambiguities. Therefore, our design uses the conventional codec's internal image representation and position encoding to enable better ambiguity resolution without increasing computational or signalling overhead too much.

The goal of the online-trained upsampler is to minimise the rate-distortion criterion

$$\mathcal{L}_{RD} = R + \lambda D \tag{1}$$

In our case, the rate R is mostly fixed as the (subsampled) source sequence's code length makes up most of the bitrate. Even for smaller resolutions like full HD, our approach's signalling overhead amounts to less than 0.0005 bit/px for a short segment of 32 frames (about 0.5s at 60Hz). For most sequences at higher qualities, this is a small overhead. For low bitrate sequences, we will show that we can also take more extended groups of pictures, e.g., 128 frames, and still achieve coding gains. In addition, experimental results will show that for most sequences and quality settings, the additional overhead is mostly below 5%. Hence, beyond quantising parameters, we do not add any further network parameter compression in this work.

The basic steps of our algorithm are displayed in Figure 2. For implementation details, please consider Algorithm 1 in the supplementary.

3.1. Network Architecture

Our network architecture needs to integrate different sources of information and have a low number of parameters and a low number of operations per pixel. To reach this goal, we chose to use three small networks that follow the idea of MobileNets to factorise into 1×1 convolutions and 3×3 channel-wise convolutions. Each network has its own set of parameters. The data flow is shown in Figure 3. The feature network f_{Feat} processes the internal features z_{int} (see Sec. 3.3). The position network f_{Pos} takes those processed features z_{Enc} together with encoded positions p_{Enc} (see Sec. 3.2) to compute a weighting z_{Att} for the attention mechanism in the denoiser. The denoiser f_{Denoise} takes the reconstructed YUV420 input (we upsample U, V to the size of Y) to compute the upsampling-residual, which is added to the bilinearly upscaled reconstruction. The computational complexities for inference and for training for all networks are listed in Table 1. The detailled architectures and complexities are listed in Tables 12, 13, and 14 of the supplementary material.

Table 1. Total computational complexity (in MAC) for a single pixel during inference or a single pixel in a batch during training. F/W refers to forward, BN to BatchNorm (estimation of μ and σ), B/W to backward, and "Grad." to the gradient computation.

Mode	Inference	Training			
Network	F/W	F/W (Trn)	BN	B/W	Grad.
Feature	29.25	31.125	3.75	26.25	29.25
Position	173.50	188.500	30.00	187.00	171.75
Denoiser	329.00	359.250	60.50	340.25	325.50
Σ Op/Pix	531.75	578.875	94.25	553.50	526.50
Σ Op/Pix Training			1753.125		

3.2. Position Encoding

We propose to provide position information as an additional input to the upsampler to reduce ambiguity. The absolute position of a pixel in a group of pictures has a temporal (frame number) dimension t and two spatial dimensions x and y. The position tensor p_{Abs} has hence the dimensions $[T, 3, H, W]$ for T frames of size $H \times W$. The positions are stored in the second axis, like channels of a CNN tensor. Each entry is equal to its normalised index (t, x, y): $p_{\text{Abs}}[t, :, x, y] = \left[\frac{t}{T}, \frac{x}{H}, \frac{y}{W}\right]$.

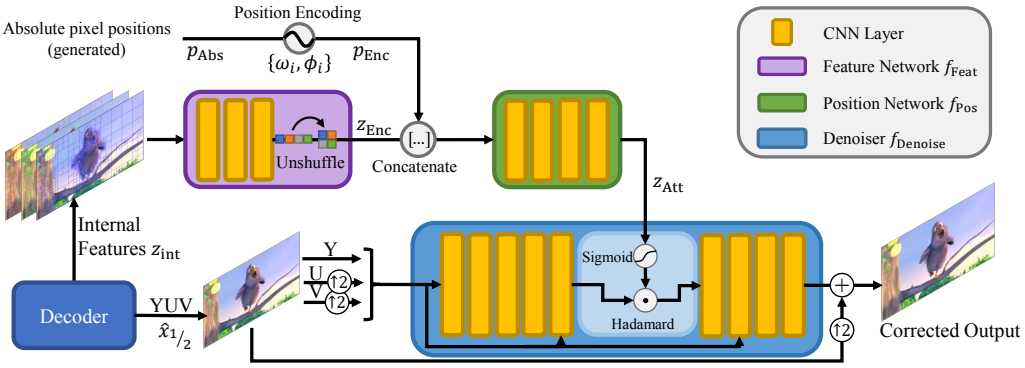

Figure 3. **Neural Network Architecture.** Three networks process the reconstructed low resolution sequence $\hat{x}_{1/2}$, the internal features z_{int}, and the encoded positisons p_{Enc}. The latter two are used to compute the attention signal z_{Att} that is used in the denoiser. The denoiser operates on low resolution and only provides the upsampled residual signal at the last layer. Each layer is a combination of BatchNorm, Convolution and ReLU (except output layers which are simply linear).

One could use these positions directly. However, position encoding has previously been shown to be beneficial to natural language processing models (see [45, 41, 10], for example). In particular, Vaswani et al. [41] used a simple trigonometric encoding, creating phasors of

$$\mathbf{p}_{\text{Enc},i} = \begin{pmatrix} \sin(2\pi\omega_i p_{\text{Abs}} + \phi_i) \\ \cos(2\pi\omega_i p_{\text{Abs}} + \phi_i) \end{pmatrix} \tag{2}$$

for different pairs of frequencies ω_i and phases ϕ_i. This performed similarly well on a machine translation task as the learned encodings of Gehring et al. [10]. Therefore, we adopt trigonometric encoding, with different frequencies ω_i for temporal and spatial coordinates. In the context of images, low frequencies correspond to distinguishing between positions far away (e.g., pixels in different quadrants of an image). In contrast, high frequencies change the encoding from one pixel to the next. We observed that low frequencies improved the predictive capability while high frequencies lead to opposite results, in some instances, dramatically reducing performance. As combining both frequencies didn't yield any benefit, we chose $\mathbf{\Omega}_t = [0.5, 0.25]$ for the temporal coordinate (frame no.) and $\mathbf{\Omega}_s = [0.5, 0.25, 0.125]$ for horizontal and vertical axes; the phase ϕ is initialised to 0. Each phasor has two channels, resulting in a total of $4 + 6 + 6 = 16$ channels for p_{Enc} (see Algorithm 5 in the supplementary for details). Note that each of these channels only varies along one dimension. After computing that dimension, the remainder of the tensor can be filled with copies. Therefore the complexity per pixel is negligible.

These encoded positions are concatenated with the features z_{Enc} that are described in the next section and serve as input for the position network f_{Pos} that computes the attention weights z_{Att}.

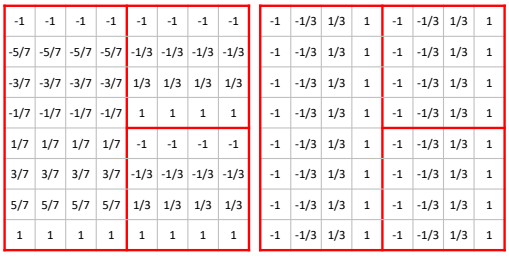

Figure 4. **Block position encoding.** Example with three blocks (red boxes), vertical coordinate on the left, horizontal on the right. The 2D coordinates are the relative position within a block.

3.3. Feature Generation

As described in the previous section, additional signals like positions can benefit error prediction performance as they provide information to distinguish ambiguities. Therefore, we add relative positions and prediction information based on the codec's internal structure besides encoded absolute position signals. Conventional codecs represent an image in terms of a hierarchy of blocks, controlling parameters like transform size or motion compensation. They are readily available at the decoder as they are needed to reconstruct the image. In our setting, we use information from the prediction blocks only: relative positions within each prediction block and motion vectors. Each block has up to two 2D motion vectors, giving the first four channels of z_{int}. We copy the motion vectors of a block to all points belonging to that block.

The relative position within a block is generated by

$$\mathbf{b}_{k,l} = \begin{pmatrix} -1.0 + \frac{2k}{K-2} \\ -1.0 + \frac{2l}{L-2} \end{pmatrix} \tag{3}$$

where $k = \{0, 1, \ldots, K-1\}$ and $l = \{0, 1, \ldots, L-1\}$ are the horizontal and vertical coordinates within a block of size $K \times L$. This encoding provides information about the

current block size (difference between two adjacent entries) and the position within a block. Figure 4 gives an example of a simple block setup. $\mathbf{b}_{k,l}$ adds two more channels to z_{int}, which we sample at $1/2$ the reconstruction resolution of the decoder (i.e. $1/4$ of the original video). z_{int} is then processed by the 4-layer feature network f_{Feat} (Tab. 12 of the supplementary).

3.4. Initialisation Pre-training

Meta-Learning techniques like MAML [9] and REPTILE [28] have been developed to improve transfer learning where a neural network learned on one task is to be adapted to similar tasks. To transfer these techniques to our setting, we interpret the optimisation for a single frame group as an adaptation, although our initial model is not trained but randomly initialised. We adopt REPTILE [28] as it's easier to implement and achieves results similar to MAML [9]. To obtain a better initialisation, we randomly choose G groups of pictures from different sequences (excluding the sequence we are applying our algorithm to) and apply our training algorithm for a few iterations, each time starting from the same initial parameters θ^0_{Init}. The combined changes of these parameters are then averaged and the initial parameters are updated:

$$\theta^t_{\text{Init}} = \theta^{t-1}_{\text{Init}} + \varepsilon \frac{1}{G} \sum_{g}^{G} (\theta_g - \theta^{t-1}_{\text{Init}}) \tag{4}$$

where θ_g is the final parameter vector after updating for a few iterations and ε is the meta learning rate. This process is repeated for several iterations, using different data in each (see Alg. 4 in the supplementary for the implementation).

Using this procedure has several advantages. Not only can we achieve higher gains and faster convergence, but, more importantly, the algorithm becomes more stable. This is important as the optimisation process always uses the same hyperparameters, and there is no manual tuning. Automatic tuning is possible yet very time-consuming. In practice, our implementation of REPTILE requires only about 50 updates, each being the average of updates to specific groups of pictures as denoted by Eq. 4. We use a meta-learning rate of $\varepsilon = 0.1$ in all our experiments. The ablation results in Fig. 5 show that almost all datasets evaluated profit off this simple initialisation scheme.

4. Experimental Results

4.1. Setup

Each frame group is optimised independently so that the proposed system can be applied to different scenes in parallel. Independent of the dataset, each group is optimised for 1250 iterations with a batch size of 48 over random samples of 80×160 pixels ($h \times w$) from the target sequence. Each group comprises 32 frames by default, larger groups are evaluated in Section 4.4. The parameters are optimised using Adam [15] with a learning rate of 0.01 and a weight decay of 0.0002. The exact quantisation procedure is described in Algorithm 3 of the supplementary material. All experiments are implemented in PyTorch [30] and use the same hyperparameters (unless indicated otherwise). We use x265 as a base codec because it is widely used in practice and a baseline in the learning-based coding literature. Unless specified, the codec is run in the `veryslow` setting, together with `tune` set to `psnr` or `ssim` (closest to MS-SSIM as possible) depending on the evaluation metric used. This ensures that we benchmark against the best performing setting.

4.2. Datasets

We apply our method to eight high-resolution datasets, spanning resolutions from full HD to 8K and frame rates from 30 to 120Hz. Four of those datasets were selected by video compression standardisation committees to act as indicators for performance progress. The remaining four were provided by researchers or video production professionals.

4.3. Measurements

PSNR and MS-SSIM [42] are used as measures. If not indicated otherwise, all sequences are processed and measured in the YUV 4:2:0. The measurement happens channel-wise. To account for human sensitivity to the luma component, the channels are weighted $[6/8, 1/8, 1/8]$ as is commonly done in video coding. Before aggregating results from different channels or frames, the MS-SSIM score is converted to decibel scale using $\text{MSSSIM}_{\text{dB}} = -10 \log_{10}(1 - \text{MSSSIM})$. To have an objective measure of comparison over different qualities, we compute the Bjontegaard Deltas [5] for the rate (i.e., how much rate is saved at the same quality, in %) as well as for the quality measure (i.e. how much quality is gained at the same rate, in dB). Following video coding standards, we perform measurements at four rate-distortion trade-offs, where the quantisation parameter (QP) is set to 22, 27, 32, and 37. A fixed QP offset of -5 is used to encode the low-resolution video signal in our algorithm.

4.4. Offline and High-Latency Encoding

The offline encoding setting is used by video-on-demand and similar services where sequences are encoded in various resolutions, optimised over various configurations to find the best possible format for each device. This setting is hence essential as it is widely used to provide video over the internet.

Table 2 lists rate savings and quality gains for both metrics. The two highest resolutions have slightly lower gains than the other datasets. One reason may be that they differ

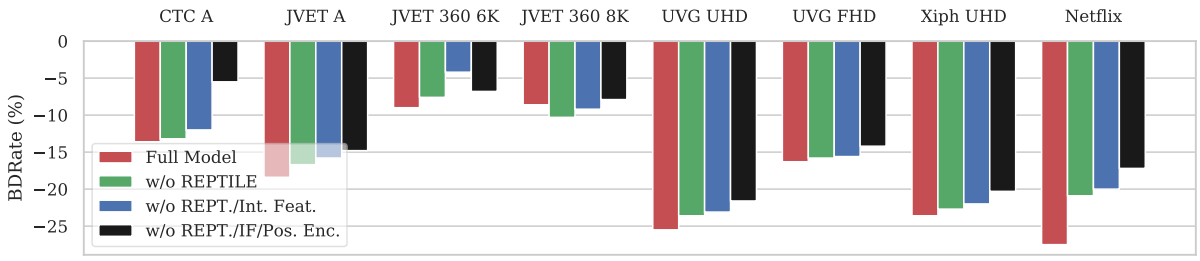

Figure 5. Ablation on different techniques used in our model for the eight datasets. The ablation successively removes first the pre-training (Sec. 3.4), then the internal features (Sec. 3.3), and finally the position encoding (Sec. 3.2) so that only the denoiser is left.

Table 2. Bjontegaard Deltas for Rate and Distortion as measured by PSNR and MS-SSIM over x265 with tunes `psnr` and `ssim`, respectively. Negative rate savings indicate that our method requires less bits of code to deliver the same quality. Both quality measures are taken in dB.

	PSNR		MSSSIM	
	ΔRate	ΔPSNR	ΔRate	ΔMSSSIM
CTC A	-13.6%	+0.2954	-9.0%	+0.1399
JVET A	-18.4%	+0.4813	-17.1%	+0.4966
JVET 360 6K	-9.0%	+0.2451	-12.3%	+0.4034
JVET 360 8K	-8.6%	+0.1864	-10.4%	+0.3484
UVG UHD [26]	-25.5%	+0.6247	-17.1%	+0.3979
UVG FHD [26]	-16.3%	+0.5156	-13.0%	+0.4277
Xiph UHD [8]	-23.6%	+0.4374	-21.2%	+0.4421
Netflix	-27.5%	+0.6632	-21.3%	+0.4399

in content as they depict 360° videos where large parts of the frame are occupied with relatively flat sky and ground textures that are easier to encode, leaving less room for improvement. For all other datasets, our proposed method delivers between 13.6% and 27.5% rate gains on PSNR, i.e., up to a quarter of data size can be saved. For MS-SSIM, the gains are slightly lower for most datasets, except for the two 360° sequence collections. Still, we can achieve up to 21.3% rate savings, i.e., one fifth bandwidth reduction. Next, we evaluate how the different measures introduced in the previous section have contributed to the overall results. Figure 5 shows rate savings and PSNR gains for the complete model, and with pre-training, internal features, and position encoding removed one after another. Over almost all datasets, the performance degrades significantly, both in terms of rate reduction and quality improvement. Again, the 360° videos do show slightly different results, which is probably specific to their content. Interestingly, for the CTC A dataset, where gains are comparably low at 13.6%, i.e., an improvement over the base codec is challenging, the proposed measures have the most impact.

Lastly, we compare to the online-trained denoiser of [16]. Their approach has a simpler network architecture

Table 3. Comparison between our method and [16]. Numbers in parentheses denote the number of frames in a group.

	Our's		Klopp et al. [16]	
Complexity	531.75 Op/Pix		488.75 Op/Pix	
	ΔRate	ΔPSNR	ΔRate	ΔPSNR
CTC A (32)	-13.6%	+0.2954	-9.1%	+0.2541
JVET A (32)	-18.4%	+0.4813	-5.7%	+0.1561
Netflix (32)	-27.5%	+0.6632	-5.0%	+0.1667
CTC A (128)	-13.4%	+0.2992	-9.1%	+0.2655
JVET A (128)	-19.6%	+0.4983	-6.3%	+0.1735
Netflix (128)	-20.6%	+0.6331	-6.4%	+0.1891

than ours, using only the codec's reconstruction as input. We use a 7-layer-20-filter architecture that requires 488.75 op/pixel for their approach (see Tables 15 and 16 in the supplementary). Note that this is the complexity for both the luma and the chroma network ([16] has two separate networks). Table 3 shows the results for three datasets and two different frame-group lengths. Our algorithm achieves higher gains for short and long frame groups, especially on the UHD and 4K-DCI datasets. For the CTC A dataset, taking into account the ablation study results shown in Fig. 5, we only outperform because of the additional input and the attention mechanism.

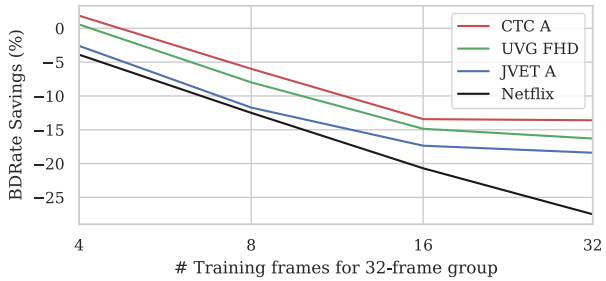

Figure 6. BDRate savings over the number of frames available for training in each group of pictures. Frames are taken from the beginning of the group. The group size is 32 frames.

4.4.1 Generalisation

We slightly change the training setup to evaluate generalisation: instead of training on all data in a frame-group, we only provide the first N frames. The evaluation still considers all frames. Fig. 6 shows the results for 32 frames per group, results for 128 frames are found in Fig. 11 of the supplementary. The performance drops sharply once less than the first half is available, indicating that in this setting, generalisation is possible if sufficient training data (relative to the amount of testing data) is available. We will exploit this property in the zero-latency scenario described below.

4.4.2 Encoding Time

Table 4 compares the encoding time per frame of our method to the original x265 on the JVET A UHD dataset. The run time for our method contains the time to run x265 on the low-resolution stream. We use a dual Intel Xeon 5680@3.33GHz with 24 cores total and an NVidia 1080 GPU with 8G memory. Our x265 is the most recent version and includes assembler speed-ups. For a group of 32 frames, our method takes longer than the baseline. However, as shown above, we can easily expand to 128 frames without much loss. In that setting our method reduces encoding time by up to one third, where coding gain is typically paid for with longer encoding times.

Table 4. Encoding duration in seconds/frame for x265 and our method (including low resolution x265) for the JVET A UHD dataset. Positive savings indicate our algorithm requires more time, negative that we save time compared to x265. The numbers in parentheses are the frame group lengths.

QP	x265	Ours (32)	Savings	Ours (128)	Savings
22	1.68s	2.40s	42.36%	1.11s	-34.15%
27	1.24s	2.23s	79.79%	0.94s	-24.33%
32	1.00s	2.10s	110.48%	0.81s	-18.60%
37	0.82s	2.01s	144.92%	0.72s	-12.02%

4.5. Zero-Latency

With the previous sections' results, it is straightforward to extend our algorithm to the zero-latency case. Exploiting generalisation, we change our algorithm to train on frames while they are zero-latency encoded and sent out by the conventional codec. As shown in Fig. 7 (top), we split training and test data: the encoder trains while data arrives, the decoder uses previously signalled parameters. To simulate a realistic scenario, we make frames only successively available to the training algorithm (see Fig. 7, bottom): we sample patches from the first $i\frac{N}{I}$ frames during the i-th iteration, where N and I are the total number of frames and the total iterations, respectively, for a frame-group. While the

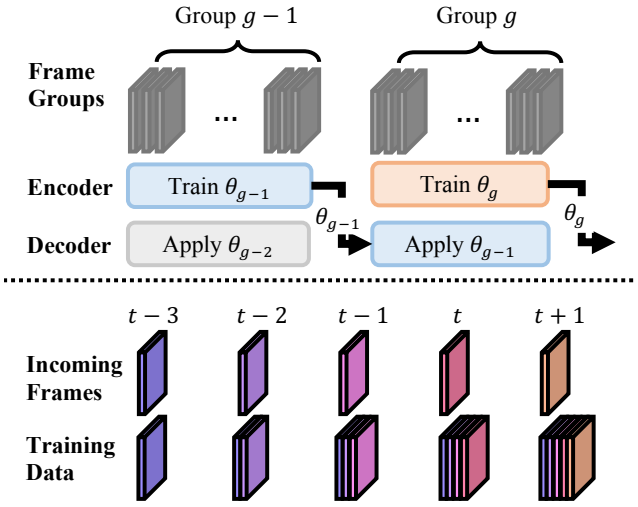

Figure 7. **Top:** zero-latency encoding realised by separating training and test data. **Bottom:** Training data is successively extended once a new frame becomes availble until the end of the frame group is reached.

Table 5. Rate savings for different signalling frequencies under zero-latency conditions.

Signalling Frequency	8	16	32
CTC A	-7.9%	**-11.5%**	-8.5%
JVET A	**-12.0%**	-11.6%	-10.8%
Netflix	**-17.2%**	-14.3%	-8.6%

encoder trains parameters θ_g on group g, the decoder uses θ_{g-1} on group g.

For our experiments, the base codec is switched to zero-latency mode using `bframes=0` as is common in other works. Because this zero-latency scenario is more time-critical, we reduce the number of iterations per frame-group to 200, the batch size to 40, and the patch size to 72×144. Table 5 shows the results for $N = \{8, 16, 32\}$. Although our algorithm is only applied to unseen data, it can still achieve significant rate gains. This indicates that jointly optimising over a large group of frames is not necessarily limited to offline processing.

To charter the territory for different coding approaches, Fig. 8 shows our algorithm ($N = 16$ frames), the x265 baseline, and two state-of-the-art learned video codecs. As demonstrated above, our codec outperforms x265, it even lifts the `medium` to the `veryslow` setting for low bitrates. Note that our algorithm is responsible for providing $3/4$ of the output data, making this improvement non-trivial. Learned codecs still require higher bit rates. However, it may very well be possible to use our method to advance them as well.

Regarding the encoding cost, because our network converges very quickly, our encoding complexity is at most

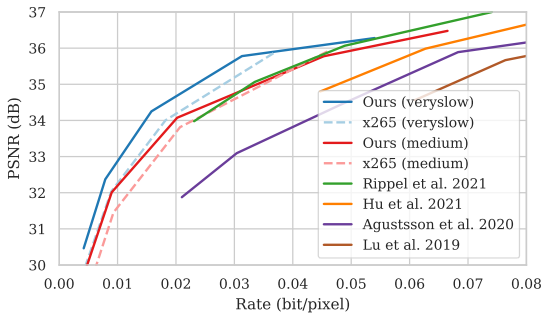

Figure 8. Rate-Distortion characteristics over the UVG FHD dataset, comparing the zero-latency version of our algorithm to x265 (in medium and veryslow settings) and to [31, 11, 1, 24], all using end-to-end deep-learned video codecs. Encoding happens without chroma subsampling (i.e. in YUV 4:4:4) and PSNR is measured in the RGB colour space for fair comparison. Note that we added a fifth data point with QP=15.

Table 6. Encoding complexity in Op/Pix. Computed as training pixels divided by total pixels in frame group times the pixel-wise training complexity from Tab. 1. In the high resolution/32 frames case our model's encode and decode complexity almost match.

Method	Signalling Freq.	8	16	32
Ours	UVG FHD	8766	4383	2192
	CTC A (WQXGA)	4438	2219	1110
	JVET A (UHD)	2192	1096	548
	Netflix (4K)	2045	1023	512
Agustsson et al. [1]		> 140000		
Lu et al. [24]		> 100000		

only an order above our decoding complexity as shown in Tab. 6. In addition, the underlying x265 codec is only responsible for $1/4$ of the pixels, allowing for faster encoding.

5. Discussion

Coding gain aside, our method faces two critical issues: encoding complexity and signalling overhead. In Sec. 4.5 we showed that the resulting encoding complexity does not need to be infeasibly high. The timing results from Sec. 4.4.2 support this. Besides, Sec. 7.1 of the supplementary demonstrates that our method converges quickly, i.e., encoding may be sped with little effect on the coding gain.

The signalling overhead is implicitly included in the measurements reported above. However, it is a crucial aspect of our design and should be investigated. Tab. 7 lists the network parameters' share in the code size for the UVG FHD dataset in the zero-latency and the offline setting. The share becomes significant when the quality is low, or the sequence has little dynamic (e.g. "HoneyBee"). However, lower qualities have more room for improvement, meaning

Table 7. Our method's network parameter size relative to the encoded size (incl. parameters) for the UVG FHD dataset.

	Zero-Latency				Offline			
Sequence	17	22	27	32	17	22	27	32
Beauty	3%	8%	15%	25%	3%	5%	9%	14%
Bosphorus	3%	6%	13%	26%	2%	4%	8%	14%
HoneyBee	10%	24%	36%	45%	3%	5%	8%	11%
Jockey	3%	6%	11%	17%	2%	4%	6%	10%
ReadySetGo	1%	2%	4%	8%	1%	2%	3%	4%
ShakeNDry	1%	4%	11%	24%	1%	2%	5%	8%
YachtRide	1%	2%	4%	10%	1%	1%	3%	7%
Average	3%	8%	13%	22%	2%	3%	6%	10%

that our algorithm typically leads to more distortion reduction. In the future, shorter bit-depth and entropy coding can help to lower the share even further.

Finally, Fig. 9 shows that our coding improvements are largely independent of rate and quality, i.e., the gains achieved here are not due to a skewed test data distribution and are likely to be realised in practice as well.

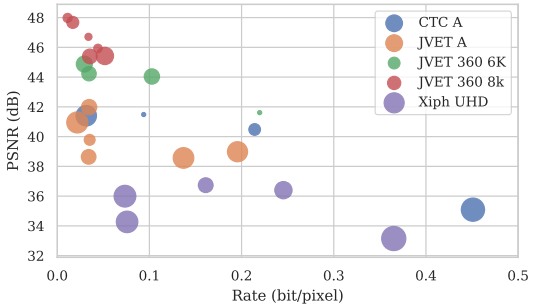

Figure 9. Distribution of rate savings over rate and distortion at QP 22. Savings are indicated by the radius of each blob.

6. Conclusion

In this work, we have introduced a super-resolution based online-trained augmentation method for conventional codecs with low computational overhead compared to pretrained or deep-learned approaches. Our evaluations demonstrate rate savings across various datasets compared to x265 and that a shorter encoding time is possible.

In the future, a straightforward improvement is to add parameter compression methods to lower the signalling overhead of our method. Besides, even better network architectures or optimisation techniques may further increase the coding gains of the proposed method, establishing a path for hybrid conventional-deep-learned codecs. On the other hand, it would also be interesting how our algorithm combines with learned codecs to provide better performance at a lower cost for high resolutions.

References

[1] Eirikur Agustsson, David Minnen, Nick Johnston, Johannes Ballé, Sung Jin Hwang, and George Toderici. Scale-space flow for end-to-end optimized video compression. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 8500–8509, 2020.

[2] Johannes Ballé, Nick Johnston, and David Minnen. Integer Networks for Data Compression with Latent-Variable Models. In *International Conference On Learning Representations*, 2019.

[3] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end Optimized Image Compression. *ICLR*, 2017.

[4] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. *International Conference On Learning Representations*, 2018.

[5] G Bjøntegaard. Calculation of Average PSNR Differences between RD curves. ITU-T SG16/Q6. Technical report, ITU-T SG16/Q6, Austin, Texas, USA, 2001.

[6] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learning image and video compression through spatial-temporal energy compaction. *arXiv*, pages 10071–10080, 2019.

[7] Jinyoung Choi and Bohyung Han. Task-Aware Quantization Network for JPEG Image Compression. In *European Conference on Computer Vision*, 2020.

[8] Derf. Xiph.org :: Derf's Test Media Collection, 2020.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *34th International Conference on Machine Learning, ICML 2017*, 3:1856–1868, 2017.

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

[11] Zhihao Hu, Guo Lu, and Dong Xu. FVC: A New Framework towards Deep Video Compression in Feature Space. pages 1502–1511, 2021.

[12] Yan Huang, Li Song, and Ebroul Izquierdo. CNN Accelerated Intra Video Coding, Where Is the Upper Bound? In *2019 Picture Coding Symposium, PCS 2019*, pages 1–5. IEEE, 2019.

[13] Chuanmin Jia, Shiqi Wang, Xinfeng Zhang, Shanshe Wang, Jiaying Liu, Shiliang Pu, and Siwei Ma. Content-Aware Convolutional Neural Network for In-loop Filtering in High Efficiency Video Coding. *IEEE Transactions on Image Processing*, pages 1–1, jan 2019.

[14] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks. *Computer Vision and Pattern Recognition*, 2017.

[15] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15, 2015.

[16] Jan P. Klopp, Liang-Gee Chen, and Shao-Yi Chien. Utilising Low Complexity CNNs to Lift Non-Local Redundancies in Video Coding. *IEEE Transactions on Image Processing*, pages 1–1, 2020.

[17] Jan P. Klopp, Keng-Chi Liu, Liang-Gee Chen, and Shao-Yi Chien. How To Exploit the Transferability of Learned Image Compression to Conventional Codecs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16165–16174, 2021.

[18] Jan P Klopp, Yu-chiang Frank Wang, and Liang-gee Chen. Learning a Code-Space Predictor by Exploiting Intra-Image-Dependencies Review of Learned Image Compression. In *British Machine Vision Conference*, pages 1–12, 2018.

[19] Yat Hong Lam, Alireza Zare, Caglar Aytekin, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska Hannuksela. Compressing Weight-updates for Image Artifacts Removal Neural Networks. In *Computer Vision and Pattern Recognition Workshop*, 2019.

[20] Yat-Hong Lam, Alireza Zare, Francesco Cricri, Jani Lainema, and Miska Hannuksela. Efficient Adaptation of Neural Network Filter for Video Compression. In *ACM Multimedia*, 2020.

[21] Haojie Liu, Tong Chen, Peiyao Guo, Qiu Shen, and Zhan Ma. Gated Context Model with Embedded Priors for Deep Image Compression. feb 2019.

[22] Jerry Liu, Shenlong Wang, Wei-Chiu Ma, Meet Shah, Rui Hu, Pranaab Dhawan, and Raquel Urtasun. Conditional Entropy Coding for Efficient Video Compression. 2020.

[23] Guo Lu, Chunlei Cai, Xiaoyun Zhang, Li Chen, Wanli Ouyang, Dong Xu, and Zhiyong Gao. Content Adaptive and Error Propagation Aware Deep Video Compression. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12347 LNCS:456–472, 2020.

[24] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. DVC: An End-to-end Deep Video Compression Framework. In *Computer Vision and Patter Recognition*, nov 2019.

[25] Di Ma, Fan Zhang, and David R. Bull. Video compression with low complexity CNN-based spatial resolution adaptation. 2020.

[26] Alexandre Mercat, Marko Viitanen, and Jarno Vanne. UVG dataset: 50/120fps 4K sequences for video codec analysis and development. In *MMSys 2020 - Proceedings of the 2020 Multimedia Systems Conference*, 2020.

[27] David Minnen, Johannes Ballé, and George Toderici. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In *Neural Information Processing Systems*, pages 10771–10780, 2018.

[28] Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. pages 1–15, 2018.

[29] Jens Rainer Ohm, Gary J. Sullivan, Heiko Schwarz, Thiow Keng Tan, and Thomas Wiegand. Comparison of the coding efficiency of video coding standards-including high efficiency video coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology*, 2012.

[30] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban

Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[31] Oren Rippel, Alexander G. Anderson, Kedar Tatwawadi, Sanjay Nair, Craig Lytle, and Lubomir Bourdev. ELF-VC: Efficient Learned Flexible-Rate Video Coding. 2021.

[32] Oren Rippel and Lubomir Bourdev. Real-Time Adaptive Image Compression. *ICML*, 2017.

[33] Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, and Lubomir Bourdev. Learned Video Compression. nov 2018.

[34] Yannick Strümpler, Ren Yang, and Radu Timofte. Learning to improve image compression without changing the standard decoder. In *Proceedings of the 16th European Conference on Computer Vision Workshops*, 2020.

[35] Hui Su, Mingliang Chen, Alexander Bokov, Debargha Mukherjee, Yunqing Wang, and Yue Chen. Machine Learning Accelerated Transform Search for AV1. In *2019 Picture Coding Symposium, PCS 2019*, pages 1–5. IEEE, 2019.

[36] Hui Su, Chi Yo Tsai, Yunqing Wang, and Yaowu Xu. Machine Learning Accelerated Partition Search for Video Encoding. In *Proceedings - International Conference on Image Processing, ICIP*, volume 2019-Septe, pages 2661–2665. IEEE, 2019.

[37] Wenyu Sun, Chen Tang, Weigui Li, Zhuqing Yuan, Huazhong Yang, and Yongpan Liu. High-Quality Single-Model Deep Video Compression with Frame-Conv3D and Multi-frame Differential Modulation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12375 LNCS:239–254, 2020.

[38] Hossein Talebi, Damien Kelly, Xiyang Luo, Ignacio Garcia Dorado, Feng Yang, Peyman Milanfar, and Michael Elad. Better compression with deep pre-editing. *arXiv*, 2020.

[39] George Toderici, Sean M. O'Malley, Sung Jin Hwang, Damien Vincent, David Minnen, Shumeet Baluja, Michele Covell, and Rahul Sukthankar. Variable Rate Image Compression with Recurrent Neural Networks. *International Conference On Learning Representations*, pages 1–9, 2015.

[40] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full Resolution Image Compression with Recurrent Neural Networks. *Computer Vision and Pattern Recognition*, 2016.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5999–6009, 2017.

[42] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. *IEEE Asilomar Conference on Signals, Systems and Computers*, 2:9–13, 2003.

[43] Ren Yang, Mai Xu, Tie Liu, Zulin Wang, and Zhenyu Guan. Enhancing Quality for HEVC Compressed Videos. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2018.

[44] Ruihan Yang, Yibo Yang, Joseph Marino, and Stephan Mandt. Hierarchical Autoregressive Modeling for Neural Video Compression. pages 1–15, 2020.

[45] Yunlun Yang, Yunhai Tong, Shulei Ma, and Zhi Hong Deng. A position encoding convolutional neural network based on dependency tree for relation classification. *EMNLP 2016 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, pages 65–74, 2016.

[46] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (1):586–595, 2018.

[47] Yongbing Zhang, Tao Shen, Xiangyang Ji, Yun Zhang, Ruiqin Xiong, and Qionghai Dai. Residual Highway Convolutional Neural Networks for in-loop Filtering in HEVC. *IEEE Transactions on Image Processing*, 27(8), 2018.