Learned Video Compression

Oren Rippel, Sanjay Nair, Carissa Lew, Steve Branson, Alexander G. Anderson, Lubomir Bourdev
WaveOne, Inc.
{oren, sanjay, carissa, steve, alex, lubomir}@wave.one

Abstract

We present a new algorithm for video coding, learned end-to-end for the low-latency mode. In this setting, our approach outperforms all existing video codecs across nearly the entire bitrate range. To our knowledge, this is the first ML-based method to do so.

We evaluate our approach on standard video compression test sets of varying resolutions, and benchmark against all mainstream commercial codecs in the low-latency mode. On standard-definition videos, HEVC/H.265, AVC/H.264 and VP9 typically produce codes up to 60% larger than our algorithm. On high-definition 1080p videos, H.265 and VP9 typically produce codes up to 20% larger, and H.264 up to 35% larger. Furthermore, our approach does not suffer from blocking artifacts and pixelation, and thus produces videos that are more visually pleasing.

We propose two main contributions. The first is a novel architecture for video compression, which (1) generalizes motion estimation to perform any learned compensation beyond simple translations, (2) rather than strictly relying on previously transmitted reference frames, maintains a state of arbitrary information learned by the model, and (3) enables jointly compressing all transmitted signals (such as optical flow and residual).

Secondly, we present a framework for ML-based spatial rate control — a mechanism for assigning variable bitrates across space for each frame. This is a critical component for video coding, which to our knowledge had not been developed within a machine learning setting.

1. Introduction

Video content consumed more than 70% of all internet traffic in 2016, and is expected to grow threefold by 2021 [1]. At the same time, the fundamentals of existing video compression algorithms have not changed considerably over the last 20 years [46, 36, 35, ...]. While they have been very well engineered and thoroughly tuned, they are hard-coded, and as such cannot adapt to the growing demand and increasingly versatile spectrum of video use cases such as social media sharing, object detection, VR streaming, and so on.

Meanwhile, approaches based on deep learning have revolutionized many industries and research disciplines. In particular, in the last two years, the field of image compression has made large leaps: ML-based image compression approaches have been surpassing the commercial codecs by significant margins, and are still far from saturating to their full potential (survey in Section 1.3).

The prevalence of deep learning has further catalyzed the proliferation of architectures for neural network acceleration across a spectrum of devices and machines. This hardware revolution has been increasingly improving the performance of deployed ML-based technologies — rendering video compression a prime candidate for disruption.

In this paper, we introduce a new algorithm for video coding. Our approach is learned end-to-end for the low-latency mode, where each frame can only rely on information from the past. This is an important setting for live transmission, and constitutes a self-contained research problem and a stepping-stone towards coding in its full generality. In this setting, our approach outperforms all existing video codecs across nearly the entire bitrate range.

We thoroughly evaluate our approach on standard datasets of varying resolutions, and benchmark against all modern commercial codecs in this mode. On standard-definition (SD) videos, HEVC/H.265, AVC/H.264 and VP9 typically produce codes up to 60% larger than our algorithm. On high-definition (HD) 1080p videos, H.265 and VP9 typically produce codes up to 20% larger, and H.264 up to 35% larger. Furthermore, our approach does not suffer from blocking artifacts and pixelation, and thus produces videos that are more visually pleasing (see Figure 1).

In Section 1.1, we provide a brief introduction to video coding in general. In Section 1.2, we proceed to describe our contributions. In Section 1.3 we discuss related work, and in Section 1.4 we provide an outline of this paper.

1.1. Video coding in a nutshell

1.1.1 Video frame types

Video codecs are designed for high compression efficiency, and achieve this by exploiting spatial and temporal redundancies within and across video frames ([51, 47, 36, 34] provide great overviews of commercial video coding techniques). Existing video codecs feature 3 types of frames:

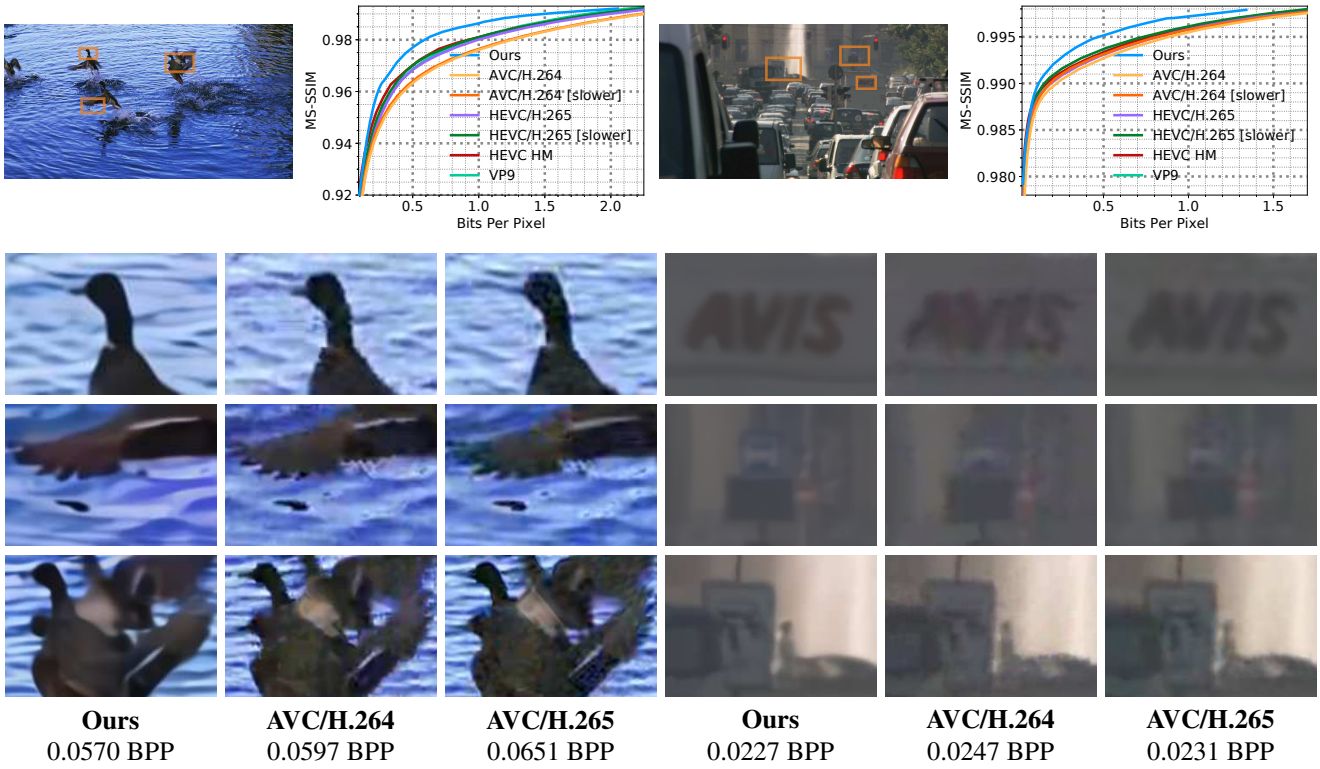


Figure 1. Examples of reconstructions by different codecs for the same bits per pixel (BPP) value. Videos taken from the Xiph HD library[2], commonly used for compression evaluation. Comprehensive benchmarking results can be found in Section 5. **Top left:** raw input frame, with boxes around areas zoomed-in on. **Top right:** rate-distortion curves for each video. **Bottom rows:** crops from the reconstruction by each codec for visual comparisons of fine details (better viewed electronically).

1. I-frames ("intra-coded"), compressed using an image codec and do not depend on any other frames;

2. P-frames ("predicted"), extrapolated from frames in the past; and

3. B-frames ("bi-directional"), interpolated from previously transmitted frames in both the past and future.

While introducing B-frames enables higher coding efficiency, it increases the latency: to decode a given frame, future frames have to first be transmitted and decoded.

1.1.2 Compression procedure

In all modern video codecs, P-frame coding is invariably accomplished via two separate steps: (1) motion compensation, followed by (2) residual compression.

Motion compensation. The goal of this step is to leverage temporal redundancy in the form of translations. This is done via block-matching (overview at [30]), which reconstructs the current target, say $\mathbf{x}_t$ for time step t, from a handful of previously transmitted *reference frames*. Specifically, different blocks in the target are compared to ones within the reference frames, across a range of possible displacements. These displacements can be represented as an optical flow map $\mathbf{f}_t$, and block-matching can be written as a special case of the *flow estimation* problem (see Section 1.3). In order to minimize the bandwidth required to transmit the flow $\mathbf{f}_t$ and reduce the complexity of the search, the flows are applied uniformly over large spatial blocks, and discretized to precision of half/quarter/eighth-pixel.

Residual compression. Following motion compensation, the leftover difference between the target and its motion-compensated approximation $\mathbf{m}_t$ is then compressed. This difference $\boldsymbol{\Delta}_t = \mathbf{x}_t - \mathbf{m}_t$ is known as the *residual*, and is independently encoded with an image compression algorithm adapted to the sparsity of the residual.

1.2. Contributions

This paper presents several novel contributions to video codec design, and to ML modeling of compression:

Compensation beyond translation. Traditional codecs are constrained to predicting temporal patterns strictly in the form of motion. However, there exists significant redundancy that cannot be captured via simple translations. Consider, for example, an out-of-plane rotation such as a person turning their head sideways. Traditional codecs will not be able to predict a profile face from a frontal view. In contrast, our system is able to learn arbitrary spatio-temporal patterns, and thus propose more accurate predictions, leading to bitrate savings.

Propagation of a learned state. In traditional codecs all "prior knowledge" propagated from frame to frame is expressed strictly via reference frames and optical flow maps, both embedded in raw pixel space. These representations are very limited in the class of signals they may characterize, and moreover cannot capture long-term memory. In contrast, we propagate an arbitrary *state* autonomously learned by the model to maximize information retention.

Joint compression of motion and residual. Each codec must fundamentally decide how to distribute bandwidth among motion and residual. However, the optimal trade-off between these is different for each frame. In traditional methods, the motion and residual are compressed separately, and there is no easy way to trade them off. Instead, we jointly compress the compensation and residual signals using the same bottleneck. This allows our network to reduce redundancy by learning how to distribute the bitrate among them as a function of frame complexity.

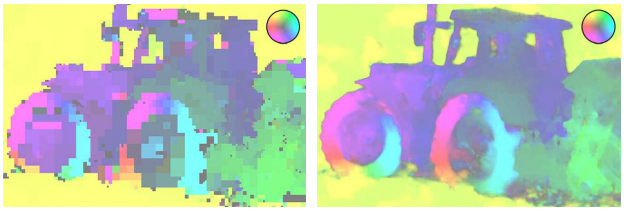
Flexible motion field representation. In traditional codecs, optical flow is represented with a hierarchical block structure where all pixels within a block share the same motion. Moreover, the motion vectors are quantized to a particular sub-pixel resolution. While this representation is chosen because it can be compressed efficiently, it does not capture complex and fine motion. In contrast, our algorithm has the full flexibility to distribute the bandwidth so that areas that matter more have arbitrarily sophisticated motion boundaries at an arbitrary flow precision, while unimportant areas are represented very efficiently. See comparisons in Figure 2.

Multi-flow representation. Consider a video of a train moving behind fine branches of a tree. Such a scene is highly inefficient to represent with traditional systems that use a single flow map, as there are small occlusion patterns that break the flow. Furthermore, the occluded content will have to be synthesized again once it reappears. We propose a representation that allows our method the flexibility to decompose a complex scene into a *mixture of multiple simple flows* and preserve occluded content.

Spatial rate control. It is critical for any video compression approach to feature a mechanism for assigning different bitrates at different spatial locations for each frame. In ML-based codec modeling, it has been challenging to construct a *single model* which supports R multiple bitrates, and achieves the same results as R *separate, individual models* each trained exclusively for one of the bitrates. In this work we present a framework for ML-driven spatial rate control which meets this requirement.

1.3. Related Work

ML-based image compression In the last two years, we have seen a great surge of ML-based image compression approaches [15, 44, 45, 5, 4, 14, 25, 43, 38, 23, 2, 27, 6, 3, 10, 32, 33]. These learned approaches have been reinventing many of the hard-coded techniques developed in traditional image coding: the coding scheme, transformations into and out of a learned codespace, quality assessment, and so on.



(a) H.265 motion vectors. (b) Our optical flow.

Figure 2. Optical flow maps for H.265 and our approach, for the same bitrate. Traditional codecs use a block structure to represent motion, and heavily quantize the motion vectors. Our algorithm has the flexibility to represent arbitrarily sophisticated motion.

ML-based video compression. To our knowledge, the only pre-existing end-to-end ML-based video compression approachs are [52, 8, 16]. [52] first encodes key frames, and proceeds to hierarchically interpolate the frames between them. [8] designs neural networks for the predictive and residual coding steps. [16] proposes a variational inference approach for video compression on 64x64 video samples.

Enhancement of traditional coding using ML. There have been several important contributions demonstrating the effectiveness of replacing or enhancing different components of traditional codecs with counterparts based on neural networks. These include improved motion compensation and interpolation [54, 21, 58, 31], intra-prediction coding [40], post-processing refinement [7, 55, 26, 56, 48, 57, 41, 17], and rate control [28].

Optical flow estimation. The problem of *optical flow estimation* has been widely studied over the years, with thousands of solutions developed using tools from partial differential equations [18, 29, 12, 13, ...] to, more recently, machine learning [50, 11, 22, 37, ...].

Given two similar frames $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{C\times H\times W}$, the task is to construct an optical flow field $\mathbf{f}^* \in \mathbb{R}^{2\times H\times W}$ of horizontal and vertical displacements, spatially "shuffling" values from $\mathbf{x}_1$ to best match $\mathbf{x}_2$. This can be more concretely written as

$$\mathbf{f}^* = \min_{\mathbf{f}} \mathscr{L}\left(\mathbf{x}_2, \mathbf{F}(\mathbf{x}_1, \mathbf{f})\right) + \lambda \mathscr{R}(\mathbf{f})$$

for some metric $\mathscr{L}(\cdot,\cdot)$, smoothness regularization $\mathscr{R}(\cdot)$ and where $\mathbf{F}(\cdot,\cdot)$ is the *inverse optical flow* operator $[\mathbf{F}(\mathbf{x},\mathbf{f})]_{chw} = \mathrm{x}_{c,h+\mathrm{f}_{1hw},w+\mathrm{f}_{2hw}}$. Note that while h, w are integer indices, $\mathrm{f}_{1hw}, \mathrm{f}_{2hw}$ can be real-valued, and so the right-hand side is computed using lattice interpolation. In this work we strictly discuss *inverse* flow, but often simply write "flow" for brevity.

1.4. Paper organization

The paper is organized in the following way:

- In Section 2, we motivate the overall design of our model, and present its architecture.
- In Section 3, we describe our coding procedure of generating variable-length bitstreams from the fixed-size codelayer tensors.
- In Section 4, we present our framework for ML-based spatial rate control, and discuss how we train/deploy it.
- In Section 5, we discuss our training/evaluation procedures, and present the results of our benchmarking and ablation studies.
- In Appendix A, we completely specify the architectural details of our model.

2. Model Architecture

Notation. We seek to encode a video with frames $\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^{3\times H\times W}$. Throughout this section, we discuss different strategies for video model construction. At a high level, all video coding models share the generic input-output structure in the below pseudo-code.

Algorithm Video coder structure for time step t

Input:

1: Target frame $\mathbf{x}_t \in \mathbb{R}^{3\times H\times W}$
2: Previous state $\mathbf{S}_{t-1}$

Output:

1: Bitstream $\mathbf{e}_t \in \{0,1\}^{\ell(\mathbf{e})}$ to be transmitted
2: Frame reconstruction $\hat{\mathbf{x}}_t \in \mathbb{R}^{3\times H\times W}$
3: Updated state $\mathbf{S}_t$

The state $\mathbf{S}_t$ is made of one or more tensors, and intuitively corresponds to some *prior memory* propagated from frame to frame. This concept will be clarified below.

2.1. State propagator

To motivate and provide intuition behind the final architecture, we present a sequence of steps illustrating how a traditional video encoding pipeline can be progressively adapted into an ML-based pipeline that is increasingly more general (and cleaner).

Note that in this section, we only aim to provide a high-level description of our model: we completely specify all the finer architectural details in Appendix A.

Step #1: ML formulation of the flow-residual paradigm. Our initial approach is to simulate the traditional flow-residual pipeline featured by the existing codecs (see Section 1.1), using building blocks from our ML toolbox. We first construct a learnable *flow estimator* network $\mathbf{M}(\cdot)$, which outputs an (inverse) flow $\mathbf{f}_t \in \mathbb{R}^{2\times H\times W}$ that motion-compensates the last reconstructed frame $\hat{\mathbf{x}}_{t-1}$ towards the current target $\mathbf{x}_t$.

We then construct a learnable *flow compresso''r* with encoder $\mathbf{E}_f$ and decoder $\mathbf{D}_f$ networks, which auto-encode $\mathbf{f}_t$ through a low-bandwidth bottleneck and reconstructs it as

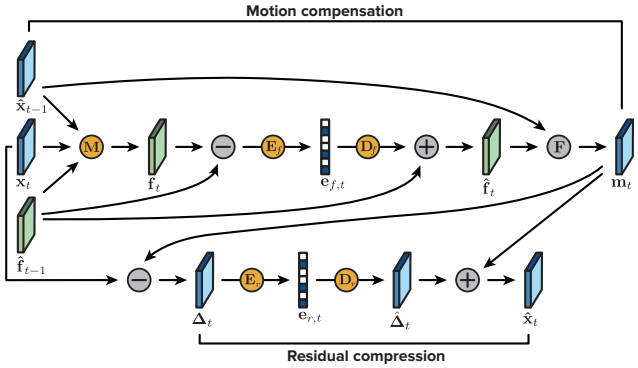


Figure 3. Graph of Step #1, which formulates the traditional flow-residual pipeline using tools from ML. **Blue** tensors correspond to frames and **green** to flows, both embedded in raw pixel space. **Yellow** operators are learnable networks, and **gray** operators are hard-coded differentiable functions. $\mathbf{M}(\cdot)$ is a flow estimator and $\mathbf{F}(\cdot,\cdot)$ is the optical flow operator described in Section 1.3.

$\hat{\mathbf{f}}_t$. Traditional codecs further increase the coding efficiency of the flow by encoding only the *difference* $\mathbf{f}_t - \hat{\mathbf{f}}_{t-1}$ from the previously-reconstructed flow.

Next, we use our reconstruction of the flow to compute a motion-compensated reconstruction of the frame itself as $\mathbf{m}_t = \mathbf{F}(\hat{\mathbf{x}}_{t-1}, \hat{\mathbf{f}}_t)$, where $\mathbf{F}(\cdot,\cdot)$ denotes the inverse optical flow operator (see Section 1.3).

Finally, we build a *residual compressor* with learnable encoder $\mathbf{E}_r(\cdot)$ and decoder $\mathbf{D}_r(\cdot)$ networks to auto-encode the residual $\mathbf{\Delta}_t = \mathbf{x}_t - \mathbf{m}_t$. Any state-of-the-art ML-based image compression architectures can be used for the core encoders/decoders of the flow and residual compressors.

See Figure 3 for a visualization of this graph. While this setup generalizes the traditional approach via end-to-end learning, it still suffers from several important impediments, which we describe and alleviate in the next steps.

Step #2: Joint compression of flow and residual. In the previous step, we encoded the flow and residual separately through distinct codes. Instead, it is advantageous in many ways to compress them jointly through a single bottleneck: this removes redundancies among them, and allows the model to automatically ascertain how to distribute bandwidth among them as function of input complexity. To that end, we consolidate to a single encoder $\mathbf{E}(\cdot)$ network and single decoder $\mathbf{D}(\cdot)$ network. See Figure 4 for a graph of this architecture.

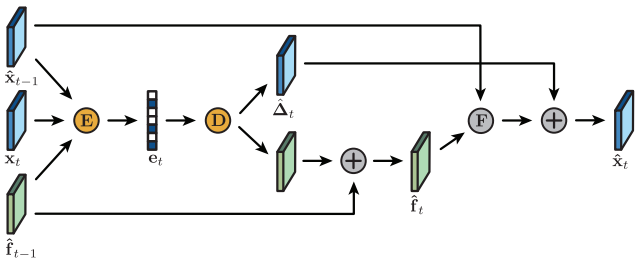


Figure 4. Graph of Step #2, which generalizes the traditional architecture of Step #1 by jointly compressing the flow and residual.

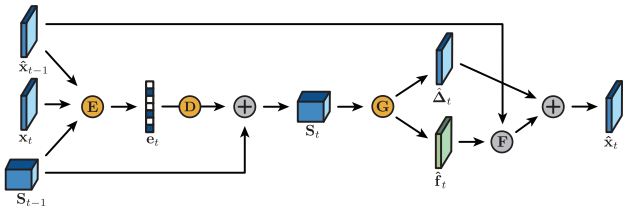


Figure 5. Graph of Step #3. Rather than relying on reference frames and flows embedded in pixel space, we instead propagate a generalized state containing information learned by the model.

Step #3: Propagation of a learned state. We now observe that all prior memory being propagated from frame to frame is represented strictly through the previously reconstructed frame $\hat{\mathbf{x}}_{t-1}$ and flow $\hat{\mathbf{f}}_{t-1}$, both embedded in raw pixel space. These representations are not only computationally inefficient, but also highly suboptimal in their expressiveness, as they can only characterize a very limited class of useful signals and cannot capture longer-term memory. Hence, it is greatly beneficial to define a generic and learnable *state* $\mathbf{S}_t$ of one or more tensors, and provide the model with a mechanism to automatically decide how to populate it and update it across time steps.

Our state propagation can be understood as an extension of a recursive neural network (RNN), where $\mathbf{S}_t$ accumulates temporal information through recursive updates. Unlike a traditional RNN, updates to $\mathbf{S}_t$ must pass through a low-bandwidth bottleneck, which we achieve through integration with modules for encoding, decoding, bitstream compression, compensation, and so on. Each frame reconstruction $\hat{\mathbf{x}}_t$ is computed from the updated state $\mathbf{S}_t$ using a module we refer to as *state-to-frame*, and denote as $\mathbf{G}(\cdot)$.

We provide an example skeleton of this architecture in Figure 5. In Figure 9, it can be seen that introducing a learned state results in 10-20% bitrate savings.

Step #4: Arbitrary compensation. We can further generalize the architecture proposed in the previous step. We observe that the form of compensation in $\mathbf{G}(\cdot)$ still simulates the traditional flow-based approach, and hence is limited to compensation of simple translations only. That is, flow-based compensation only allows us to "move pixels around", but does not allow us change their actual values. However, since we have a tool for end-to-end training, we can now learn any arbitrary compensation beyond motion. Hence, we can generalize $\mathbf{G}(\cdot)$ to generate multiple flows instead of a single one, as well as multiple reference frames to which the flows can be applied respectively.

2.2. Architecture building blocks

We have empirically found that a multiscale dual path backbone works well as a fundamental building block within our architecture. Specifically, we combine the multiscale rendition [19] of DenseNet [20] with dual path [9]. The core of our video encoder and decoder is simply repeated application of this block. Full descriptions of the choices for $\mathbf{E}(\cdot), \mathbf{D}(\cdot), \mathbf{G}(\cdot)$ can be found in Appendix A.

3. Coding procedure

We assume we have applied our encoder network, and have reached a fixed-size tensor $\mathbf{c} \in [-1, 1]^{C \times Y \times X}$ (we omit timestamps for notational clarity). The goal of the coding procedure is to map $\mathbf{c}$ to a bitstream $\mathbf{e} \in \{0, 1\}^{\ell(\mathbf{e})}$ with variable length $\ell(\mathbf{e})$. The coder is expected to achieve high coding efficiency by exploiting redundancy injected into $\mathbf{c}$ by the regularizer (see below) through the course of training. We follow the coding procedure described in detail in [38] and summarized in this section.

Bitplane decomposition. We first transform $\mathbf{c}$ into a binary tensor $\mathbf{b} \in \{0, 1\}^{B \times C \times Y \times X}$ by decomposing it into B bitplanes. This operation transformation maps each value c_{chw} into its binary expansion $\mathrm{b}_{1chw}, \ldots, \mathrm{b}_{Bchw}$ of B bits. This is a lossy operation, since the precision of each value is truncated. In practice, we use $B = 6$.

Adaptive entropy coding (AEC). The AEC maps the binary tensor $\mathbf{b}$ into a bitstream $\mathbf{e}$. We train a classifier to compute the probability of activation $\mathbb{P}[\mathrm{b}_{bcyx} = 1 \,|\, C]$ for each bit value b_{bcyx} conditioned on some *context* C. The context consists of values of neighboring pre-transmitted bits, leveraging structure within and across bitplanes.

Adaptive codelength regularization. The regularizer is designed to reduce the entropy content of $\mathbf{b}$, in a way that can be leveraged by the entropy coder. In particular, it shapes the distribution of elements of the quantized codelayer $\hat{\mathbf{c}}$ to feature an increasing degree of sparsity as a function of bitplane index. This is done with the functional form

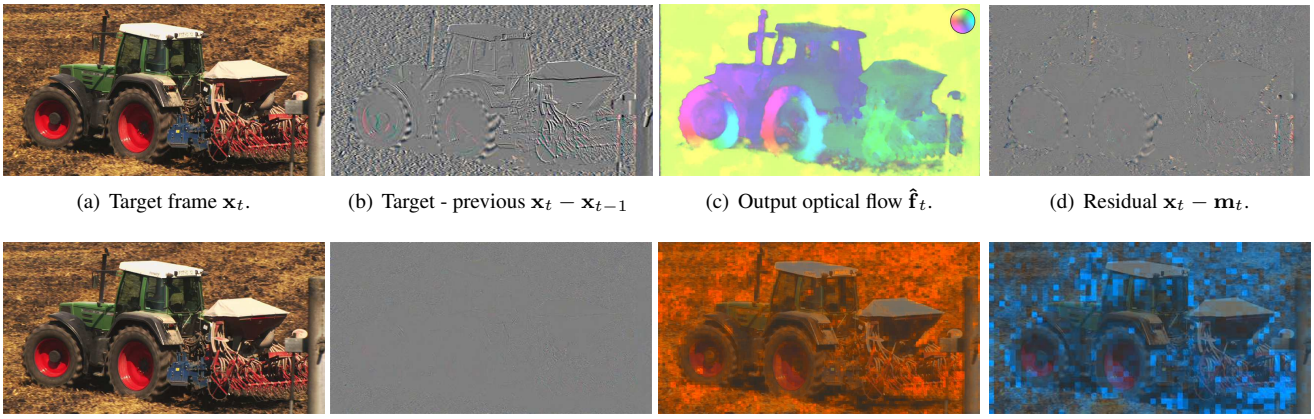
$$\mathscr{R}(\hat{\mathbf{c}}) = \frac{\alpha_i}{CYX} \sum_{cyx} \log |\hat{\mathrm{c}}_{cyx}|$$

for iteration i and scalar α_i. The choice of α_i allows training the mean codelength to match a target bitcount $\mathbb{E}_{\hat{\mathbf{c}}}[\ell(\mathbf{e})] \longrightarrow \ell^{\text{target}}$. Specifically, during training, we use the coder to monitor the average codelength. We then modulate α_i using a feedback loop as a function of the discrepancy between the target codelength and its observed value.

4. Spatial Rate Control

It is critical for any video compression approach to include support for *spatial rate control* — namely, the ability to independently assign arbitrary bitrates $\mathrm{r}_{h,w}$ at different spatial locations across each frame. A *rate controller* algorithm then determines appropriate values for these rates, as function of a variety of factors: spatiotemporal reconstruction complexity, network conditions, quality guarantees, and so on.

Why not use a single-bitrate model? Most of the ML-based image compression approaches train many individual models — one for each point on the R-D curve [5, 38, 6, ...]. It is tempting to extend this formulation to video coding, and use a model that codes at a fixed bitrate level. However, one will quickly discover that this leads to fast accumulation of error due to variability of coding complexity



(a) Target frame $\mathbf{x}_t$. (b) Target - previous $\mathbf{x}_t - \mathbf{x}_{t-1}$ (c) Output optical flow $\hat{\mathbf{f}}_t$. (d) Residual $\mathbf{x}_t - \mathbf{m}_t$.

(e) Output reconstruction $\hat{\mathbf{x}}_t$. (f) Final error $\mathbf{x}_t - \hat{\mathbf{x}}_t$. (g) MS-SSIM map. (h) Output bitrate map.

Figure 6. Visualization of intermediate outputs for an example video from the Xiph HD dataset. **(a)** The original target frame. **(b)** The difference between the target and previous frame. There are several distinct types of motion: camera pan, turning wheels, and moving tractor. **(c)** The output optical flow map as produced by the algorithm, compensating for the motion patterns as described in (b). **(d)** The leftover residual following motion compensation. **(e)** Output after addition of the residual reconstruction. **(f)** The difference between the target and its final reconstruction. **(g)** A map of the errors between the target and its reconstruction, as evaluated by MS-SSIM. Brighter color indicates larger error. **(h)** A map of the bitrates assigned by the spatial rate controller as a function of spatial location.

over both space and time. Namely, for a given frame, areas that are hard to reconstruct at a high quality using our fixed bitrate budget are going to be even more difficult in the next frame, since their quality will only degrade further. In the example in Figure 6, it can be seen that different bitrates are assigned adaptively as function of reconstruction complexity. In Figure 9, it can be seen that introducing a spatial rate controller results in 10-20% better compression.

Traditional video codecs enable rate control via variation of the quantization parameters: these control the numerical precision of the code at each spatial location, and hence provide a tradeoff between bitrate and accuracy.

In ML-based compression schemes, however, it has been quite challenging to design a high-performing mechanism for rate control. Concretely, it is difficult to construct a *single model* which supports R multiple bitrates, and achieves the same results as R *separate, individual models* each trained exclusively for one of the bitrates.

In Section 4.1, we present a framework for spatial rate control in a neural network setting, and in Section 4.2 discuss our controller algorithm for rate assignment. In Appendix A, we fully specify all architectural choices for the spatial rate controller.

4.1. Spatial multiplexing framework

Here we construct a mechanism which assigns variable bitrates across different spatial locations for each video frame. Specifically, we assume our input is a spatial map of integer rates $\mathbf{p} \in \{1, 2, \ldots, R\}^{Y \times X}$. Our goal is to construct a model that can arbitrarily vary the BPP/quality at each location (y, x), as function of the chosen rate p_{yx}.

To that end, we generalize our model featuring a single codelayer $\mathbf{c}$ to instead support R distinct codelayers $\mathbf{c}_r \in \mathbb{R}^{C_r \times Y_r \times X_r}$. Each codelayer is associated with a different rate, and is trained with a distinct entropy coder and codelength regularization (see Section 3) to match a different target codelength ℓ_r^{target}.

Our rate map $\mathbf{p}$ then specifies which codelayer is active at each spatial location. In particular, we map $\mathbf{p}$ into R binary masks $\mathbf{u}_r \in \{0, 1\}^{C_r \times Y_r \times X_r}$, one for each codelayer, of which a single one is active at each spatial location:

$$\mathrm{u}_{r,cyx} = \mathbb{I}_{\mathrm{p}_{yx}=r}\,, \qquad r = 1, \ldots, R$$

where $\mathbb{I}$ is the indicator function. Each map $\mathbf{u}_r$ masks codelayer $\mathbf{c}_r$ during entropy coding. The final bitstream then corresponds to encodings of all the active values in each codelayer, as well as the rate mask itself (since it must be available on the decoder side as well).

In terms of the architecture, towards the end of the encoding pipeline, the encoder is split into R branches $\mathbf{E}_1, \ldots, \mathbf{E}_R$, each mapping to a corresponding codelayer. Each decoder $\mathbf{D}_r$ then performs the inverse operation, mapping each masked codelayer back to a common space in which they are summed (see Figure 7). To avoid incurring considerable computational overhead, we choose the individual codelayer branches to be very lightweight: each en-

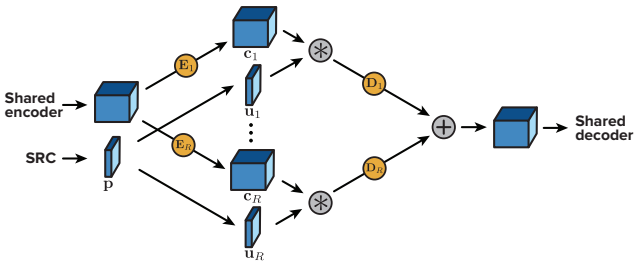
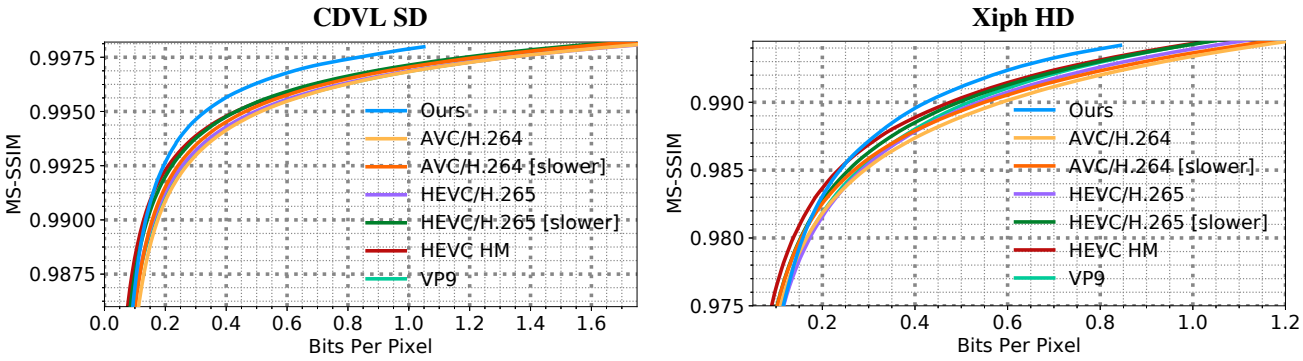


Figure 7. The architecture of the spatial multiplexer for rate control (Section 4.1). At each location, a value is chosen from one of the R codelayers, as function of the rate specified in the rate map $\mathbf{p}$. Full detail of the architecture in Appendix A.



MS-SSIM	H.264	H.265	HEVC HM	VP9
0.990	133%	122%	102%	123%
0.992	144%	133%	111%	133%
0.994	156%	148%	128%	152%
0.996	159%	152%	136%	154%
0.998	159%	160%	148%	160%

MS-SSIM	H.264	H.265	HEVC HM	VP9
0.980	121%	112%	95%	102%
0.984	132%	120%	108%	112%
0.988	131%	116%	111%	111%
0.992	115%	108%	106%	105%
0.994	120%	113%	109%	108%

Figure 8. Compression results for the CDVL SD and Xiph HD datasets. We benchmark against the default and slower presets of HEVC/H.265 and AVC/H.264, VP9, and the HEVC HM reference implementation, all in the low-latency setting (no B-frames). We tune each baseline codec to the best of our abilities. All details of the evaluation procedure can be found in Section 5.2. **Top row:** Rate-distortion curves averaged across all videos for each dataset. **Bottom row:** Average compressed sizes relative to ours, for representative MS-SSIM levels covering the BPP range for each dataset.

coder/decoder branch consists of only a single convolution.

In practice, we found that choosing target BPPs as $\ell_r^{\text{target}} = 0.01 \times 1.5^r$ leads to a satisfactory distribution of bitrates. We train a total of 5 different models, each covering a different part of the BPP range. During training, we simply sample $\mathbf{p}$ uniformly for each frame. Below we describe our use of the spatial multiplexer during deployment.

4.2. Rate controller algorithm

Video bitrate can be controlled in many ways, as function of the video's intended use. For example, it might be desirable to maintain a minimum guaranteed quality, or abide to a maximum bitrate to ensure low buffering under constraining network conditions (excellent overviews of rate control at [53, 39]). One common family of approaches is based on Lagrangian optimization, and revolves around assigning bitrates as function of an estimate of the *slope* of the rate-distortion curve. This can be intuitively interpreted as maximizing the quality improvement per unit bit spent.

Our rate controller is inspired by this idea. Concretely, during video encoding, we define some *slope threshold* λ. For a given time step, for each spatial location (y, x) and rate r, we estimate the slope $\frac{\mathscr{L}_{r+1,yx} - \mathscr{L}_{r,yx}}{\text{BPP}_{r+1,yx} - \text{BPP}_{r,yx}}$ of the local R-D curve, for some quality metric $\mathscr{L}(\cdot, \cdot)$. We then choose our rate map $\mathbf{p}$ such that at each spatial location, p_{yx} is the largest rate such that the slope is at least threshold λ.

5. Results

5.1. Experimental setup

Training data. Our training set comprises high-definition action scenes downloaded from YouTube. We found these work well due to their relatively undistorted nature, and higher coding complexity. We train our model on 128×128 video crops sampled uniformly spatiotemporally, filtering out clips which include scene cuts.

Training procedure. During training (and deployment) we encode the first frame using a learned image compressor; we found that the choice of this compressor does not significantly impact performance. We then unroll each video across 5 frames. We find diminishing returns from additional unrolling. We optimize the models with Adam [24] with momentum of 0.9 and learning rate of 2×10^{-4} which reduces by a factor of 5 twice during training. We use a batch size of 8, and for a total of $400,000$ iterations. In general, we observed no sign of overfitting, but rather the oppposite: the model has not reached the point of performance saturation as function of capacity, and seems to benefit from increasing its width and depth.

Metrics and color space. For each encoded video, we measure BPP as the total file size, including all header information, averaged across all pixels in the video.

We penalize discrepancies between the final frame reconstructions $\hat{\mathbf{x}}_t$ and their targets $\mathbf{x}_t$ using the Multi-Scale Structural Similarity Index (MS-SSIM) [49], which has been designed for and is known to match the human visual system significantly better than alternatives such as PSNR or ℓ_p-type losses. We penalize distortions in all intermediate motion-compensated reconstructions using the Charbonnier loss, known to work well for flow-based distortions [42].

Since the human visual system is considerably more sensitive to distortions in brightness than color, most existing video codecs have been designed to operate in the YCbCr color space, and dedicate higher bandwidth to luminance over chrominance. Similarly, we represent all colors in the

YCbCr domain, and weigh all metrics with Y, Cb, Cr component weights $6/8, 1/8, 1/8$.

5.2. Evaluation procedure

Baseline codecs. We benchmark against all mainstream commercial codecs: HEVC/H.265, AVC/H.264, VP9, and the HEVC HM 16.0 reference implementation. We evaluate H.264 and H.265 in both the default preset of `medium`, as well as `slower`. We use FFmpeg for all codecs, apart from HM for which we use its official implementation. We tune all codecs to the best of our ability. To remove B-frames, we use H.264/5 with the `bframes=0` option, VP9 with `-auto-alt-ref 0 -lag-in-frames 0`, and use the HM `encoder_lowdelay_P_main.cfg` profile. To maximize the performance of the baselines over the MS-SSIM metric, we tune them using the `-ssim` flag.

Video test sets. We benchmark all the above codecs on standard video test sets in SD and HD, frequently used for evaluation of video coding algorithms. In SD, we evaluate on a VGA resolution dataset from the Consumer Digital Video Library (CDVL)[1]. This dataset has 34 videos with a total of 15,650 frames. In HD, we use the Xiph 1080p video dataset[2], with 22 videos and 11,680 frames. We center-crop all 1080p videos to height 1024 (for now, our approach requires each dimension to be divisible by 32). Lists of the videos in each dataset can be found in the appendix.

Curve generation. Each video features a separate R-D curve computed from all available compression rates for a given codec: as a number of papers [5, 38] discuss in detail, different ways of summarizing these R-D curves can lead to very different results. In our evaluations, to compute a given curve, we sweep across values of the independent variable (such as bitrate). We interpolate the R-D curve for each video at this independent variable value, and average all the results across the dependent variable. To ensure accurate interpolation, we generate results for all available rates for each codec.

5.3. Performance

Rate-distortion curves. On the top row of Figure 8, we present the average MS-SSIM across all videos for each dataset and for each codec (Section 5.2), as function of BPP.

Relative compressed sizes. On the bottom row of Figure 8, we present average file sizes relative to our approach for representative MS-SSIM values. For each MS-SSIM point, we average the BPP for all videos in the dataset and compute the ratio to our BPP. Note that for this comparison, we are constrained to use MS-SSIM values that are valid for all videos in the dataset, which is 0.990-0.998 for the SD dataset and 0.980-0.994 for the HD dataset.

Ablation studies. In Figure 9, we present performance of different models with and without different components. The different configurations evaluated include:

- The full model presented in the paper;

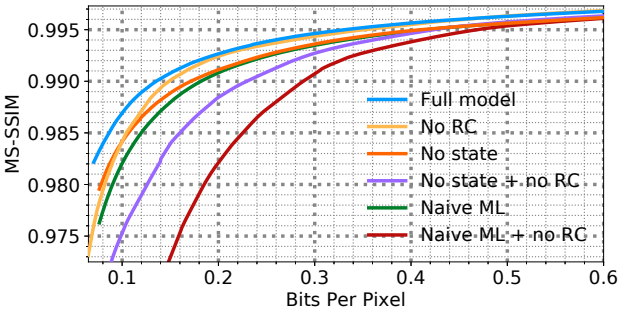


Figure 9. Ablation studies demonstrating the impact of individual architectural components on performance on the CDVL SD dataset. Factors of variation include introduction of a learned state, use of flow-based motion compensation, and spatial rate control (all described in Sections 2 and 4).

- The model described in Step #2, using previous frames and flows as prior knowledge, but without learning an arbitrary state; and
- A Naïve ML model, which does not include a learned state, and reconstructs the target frame directly without any motion compensation.

We evaluate all the above models with and without the spatial rate control framework described in Section 4.

Runtime. On an NVIDIA Tesla V100 and on VGA videos, our decoder runs on average at a speed of around 10 frames/second, and encoder at around 2 frames/second irrespective of bitrate. However, our algorithm should be regarded as a *reference implementation*: the current speed is not sufficient for real-time deployment, but is to be substantially improved in future work. For reference, on the same videos, HEVC HM encodes at around 0.3 frames/second for low BPPs and 0.04 frames/second for high BPPs.

6. Conclusion

In this work we introduced the first ML-based video codec that outperforms all commercial codecs, across nearly the entire bitrate range in the low-latency mode. However, our presented approach only supports the low-latency mode. Two clear directions of future work are to increase the computational efficiency of the model to enable real-time coding, as well as extend the model to support temporal interpolation modes (i.e, using B-frames).

Acknowledgements. We are grateful to Josh Fromm, Trevor Darrell, Sven Strohband, Michael Gelbart, Albert Azout, Bruno Olshausen and Vinod Khosla for meaningful discussions and input along the way.

[1]The Consumer Digital Video Library can be found at `http://www.cdvl.org/`. To retrieve the SD videos, we searched for VGA resolution at original and excellent quality levels. There were a few instances of near-duplicate videos: in those cases we only retrieved the first.

[2]The Xiph test videos can be found at `https://media.xiph.org/video/derf/`. We used all videos with 1080p resolution.

References

[1] White paper: Cisco vni forecast and methodology, 2016-2021. 2016.

[2] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 1141–1151. Curran Associates, Inc., 2017.

[3] Eirikur Agustsson, Michael Tschannen, Fabian Mentzer, Radu Timofte, and Luc Van Gool. Generative adversarial networks for extreme learned image compression. *arXiv preprint arXiv:1804.02958*, 2018.

[4] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. In *Picture Coding Symposium (PCS), 2016*, pages 1–5. IEEE, 2016.

[5] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

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

[7] Lukas Cavigelli, Pascal Hager, and Luca Benini. Cas-cnn: A deep convolutional neural network for image compression artifact suppression. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 752–759. IEEE, 2017.

[8] Tong Chen, Haojie Liu, Qiu Shen, Tao Yue, Xun Cao, and Zhan Ma. Deepcoder: A deep neural network based video compression. *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, 2017.

[9] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Advances in Neural Information Processing Systems*, pages 4467–4475, 2017.

[10] Thierry Dumas, Aline Roumy, and Christine Guillemot. Autoencoder based image compression: can the learning be quantization independent? *arXiv preprint arXiv:1802.09371*, 2018.

[11] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick Van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[12] David Fleet and Yair Weiss. Optical flow estimation. In *Handbook of mathematical models in computer vision*, pages 237–257. Springer, 2006.

[13] Denis Fortun, Patrick Bouthemy, and Charles Kervrann. Optical flow modeling and computation: a survey. *Computer Vision and Image Understanding*, 134:1–21, 2015.

[14] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3549–3557. Curran Associates, Inc., 2016.

[15] Karol Gregor and Yann LeCun. Learning representations by maximizing compression. 2011.

[16] Jun Han, Salvator Lombardo, Christopher Schroers, and Stephan Mandt. Deep probabilistic video compression. *arXiv preprint arXiv:1810.02845*, 2018.

[17] Xiaoyi He, Qiang Hu, Xintong Han, Xiaoyun Zhang, and Weiyao Lin. Enhancing hevc compressed videos with a partition-masked convolutional neural network. *arXiv preprint arXiv:1805.03894*, 2018.

[18] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[19] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *International Conference on Learning Representations*, 2018.

[20] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks.

[21] Shuai Huo, Dong Liu, Feng Wu, and Houqiang Li. Convolutional neural network-based motion compensation refinement for video coding. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–4. IEEE, 2018.

[22] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks.

[23] 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. *arXiv preprint arXiv:1703.10114*, 2017.

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

[25] Anders Boesen Lindbo Larsen, Sren Kaae Snderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[26] Chen Li, Li Song, Rong Xie, Wenjun Zhang, and Cooperative Medianet Innovation Center. Cnn based post-processing to improve hevc.

[27] Mu Li, Wangmeng Zuo, Shuhang Gu, Debin Zhao, and David Zhang. Learning convolutional networks for content-weighted image compression.

[28] Ye Li, Bin Li, Dong Liu, and Zhibo Chen. A convolutional neural network-based approach to rate control in hevc intra coding. In *Visual Communications and Image Processing (VCIP), 2017 IEEE*, pages 1–4. IEEE, 2017.

[29] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981.

[30] LC Manikandan and RK Selvakumar. A new survey on block matching algorithms in video coding.

[31] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[32] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Conditional probability models for deep image compression.

[33] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. *arXiv preprint arXiv:1809.02736*, 2018.

[34] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec vp9-an overview and preliminary results. In *Picture Coding Symposium (PCS), 2013*, pages 390–393. IEEE, 2013.

[35] J-R Ohm, Gary J Sullivan, Heiko Schwarz, Thiow Keng Tan, and Thomas Wiegand. Comparison of the coding efficiency of video coding standardsincluding high efficiency video coding (hevc). *IEEE Transactions on circuits and systems for video technology*, 22(12):1669–1684, 2012.

[36] Mahsa T Pourazad, Colin Doutre, Maryam Azimi, and Panos Nasiopoulos. Hevc: The new gold standard for video compression: How does hevc compare with h. 264/avc? *IEEE consumer electronics magazine*, 1(3):36–46, 2012.

[37] Zhe Ren, Junchi Yan, Bingbing Ni, Bin Liu, Xiaokang Yang, and Hongyuan Zha. Unsupervised deep learning for optical flow estimation. 2017.

[38] Oren Rippel and Lubomir Bourdev. Real-time adaptive image compression. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2922–2930, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[39] Werner Robitza. Understanding rate control modes (x264, x265, vpx). 2017.

[40] Rui Song, Dong Liu, Houqiang Li, and Feng Wu. Neural network-based arithmetic coding of intra prediction modes in hevc. In *Visual Communications and Image Processing (VCIP), 2017 IEEE*, pages 1–4. IEEE, 2017.

[41] Xiaodan Song, Jiabao Yao, Lulu Zhou, Li Wang, Xiaoyang Wu, Di Xie, and Shiliang Pu. A practical convolutional neural network as loop filter for intra frame. *arXiv preprint arXiv:1805.06121*, 2018.

[42] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2432–2439. IEEE, 2010.

[43] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszar. Lossy image compression with compressive autoencoders. 2016.

[44] 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. *arXiv preprint arXiv:1511.06085*, 2015.

[45] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full resolution image compression with recurrent neural networks. *arXiv preprint arXiv:1608.05148*, 2016.

[46] Jarno Vanne, Marko Viitanen, Timo D Hamalainen, and Antti Hallapuro. Comparative rate-distortion-complexity analysis of hevc and avc video codecs. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1885–1898, 2012.

[47] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: an overview. *IEEE Signal processing magazine*, 20(2):18–29, 2003.

[48] Tingting Wang, Mingjin Chen, and Hongyang Chao. A novel deep learning-based method of improving coding efficiency from the decoder-end for hevc. In *Data Compression Conference (DCC), 2017*, pages 410–419. IEEE, 2017.

[49] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004.*, volume 2, pages 1398–1402. Ieee, 2003.

[50] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

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

[52] Chao-Yuan Wu, Nayan Singhal, and Philipp Krähenbühl. Video compression through image interpolation. In *ECCV*, 2018.

[53] Zongze Wu, Shengli Xie, Kexin Zhang, and Rong Wu. Rate control in video coding. In *Recent Advances on Video Coding*. InTech, 2011.

[54] Ning Yan, Dong Liu, Houqiang Li, and Feng Wu. A convolutional neural network approach for half-pel interpolation in video coding. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[55] Ren Yang, Mai Xu, and Zulin Wang. Decoder-side hevc quality enhancement with scalable convolutional neural network. In *2017 IEEE International Conference on Multimedia and Expo (ICME)*, pages 817–822. IEEE, 2017.

[56] Ren Yang, Mai Xu, Zulin Wang, and Zhenyu Guan. Enhancing quality for hevc compressed videos. *arXiv preprint arXiv:1709.06734*, 2017.

[57] Ren Yang, Mai Xu, Zulin Wang, and Tianyi Li. Multi-frame quality enhancement for compressed video.

[58] Zhenghui Zhao, Shiqi Wang, Shanshe Wang, Xinfeng Zhang, Siwei Ma, and Jiansheng Yang. Cnn-based bi-directional motion compensation for high efficiency video coding. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pages 1–4. IEEE, 2018.