# DVC: An End-to-end Deep Video Compression Framework

Guo Lu[1], Wanli Ouyang[2,3], Dong Xu[2], Xiaoyun Zhang[1], Chunlei Cai[1], and Zhiyong Gao[1]

[1]Shanghai Jiao Tong University, {luguo2014, xiaoyun.zhang, caichunlei, zhiyong.gao}@sjtu.edu.cn
[2]The University of Sydney, {wanli.ouyang, dong.xu}@sydney.edu.au
[3]The University of Sydney, SenseTime Computer Vision Research Group

## Abstract

*Conventional video compression approaches use the predictive coding architecture and encode the corresponding motion information and residual information. In this paper, taking advantage of both classical architecture in the conventional video compression method and the powerful non-linear representation ability of neural networks, we propose the first end-to-end video compression deep model that jointly optimizes all the components for video compression. Specifically, learning based optical flow estimation is utilized to obtain the motion information and reconstruct the current frames. Then we employ two auto-encoder style neural networks to compress the corresponding motion and residual information. All the modules are jointly learned through a single loss function, in which they collaborate with each other by considering the trade-off between reducing the number of compression bits and improving quality of the decoded video. Experimental results show that the proposed approach can outperform the widely used video coding standard H.264 in terms of PSNR and be even on par with the latest standard H.265 in terms of MS-SSIM. Code will be publicly available upon acceptance.*

## 1. Introduction

Nowadays, video content contributes to more than 80% internet traffic [24], and the percentage is expected to increase even further. Therefore, it is critical to build an efficient video compression system and generate higher quality frames at given bandwidth budget. In addition, most video related computer vision tasks such as video object detection or video object tracking are sensitive to the quality of compressed videos, and efficient video compression may bring benefits for other computer vision tasks. Meanwhile, the techniques in video compression are also helpful for action recognition [38] and model compression [16].

(a) Original frame (Bpp/MS-SSIM) (b) H.264 (0.0540Bpp/0.945)
(c) H.265 (0.082Bpp/0.960) (d) Ours ( **0.0529Bpp/ 0.961**)

Figure 1: Visual quality of the reconstructed frames from different video compression systems. (a) is the original frame. (b)-(d) are the reconstructed frames from H.264, H.265 and our method. Our proposed method only consumes 0.0529pp while achieving the best perceptual quality (0.961) when measured by MS-SSIM. (Best viewed in color.)

However, in the past decades, video compression algorithms [36, 29] rely on hand-crafted modules, e.g., block based motion estimation and Discrete Cosine Transform (DCT), to reduce the redundancies in the video sequences. Although each module is well designed, the whole compression system is not end-to-end optimized. It is desirable to further improve video compression performance by jointly optimizing the whole compression system.

Recently, deep neural network (DNN) based auto-encoder for image compression [32, 11, 33, 8, 12, 17, 31, 19, 26, 9] has achieved comparable or even better performance than the traditional image codecs like JPEG [34], JPEG2000 [27] or BPG [1]. One possible explanation is that the DNN based image compression methods can exploit large scale end-to-end training and highly non-linear transform, which are not used in the traditional approaches.

However, it is non-trivial to directly apply these techniques to build an end-to-end learning system for video compression. **First**, it remains an open problem to learn

how to generate and compress the motion information tailored for video compression. Video compression methods heavily rely on motion information to reduce temporal redundancy in video sequences. A straightforward solution is to use the learning based optical flow to represent motion information. However, current learning based optical flow approaches aim at generating flow fields as accurate as possible. But, the precise optical flow is often not optimal for a particular video task [39]. In addition, the data volume of optical flow increases significantly when compared with motion information in the traditional compression systems and directly applying the existing compression approaches in [36, 29] to compress optical flow values will significantly increase the number of bits required for storing motion information. **Second**, it is unclear how to build a DNN based video compression system by minimizing the rate-distortion based objective for both residual and motion information. Rate-distortion optimization (RDO) aims at achieving higher quality of reconstructed frame (*i.e.*, less distortion) when the number of bits (or bit rate) for compression is given. RDO is important for video compression performance. In order to exploit the power of end-to-end training for learning based compression system, the RDO strategy is required to optimize the whole system.

In this paper, we propose the first end-to-end deep video compression (DVC) model that jointly learns motion estimation, motion compression, and residual compression. The advantages of the this network can be summarized as follows:

- All key components in video compression, *i.e.*, motion estimation, motion compensation, residual compression, motion compression, quantization, and bit rate estimation, are implemented with an end-to-end neural networks.
- The key components in video compression are jointly optimized based on rate-distortion trade-off through a single loss function, which leads to higher compression efficiency.
- There is one-to-one mapping between the components of conventional video compression approaches and our proposed DVC model. This work serves as the bridge for researchers working on video compression, computer vision, and deep model design. For example, better model for optical flow estimation and image compression can be easily plugged into our framework. Researchers working on these fields can use our DVC model as a starting point for their future research.

Experimental results show that estimating and compressing motion information by using our neural network based approach can significantly improve the compression performance. Our framework outperforms the widely used video codec H.264 when measured by PSNR and be on par with the latest video codec H.265 when measured by the multi-scale structural similarity index (MS-SSIM) [35].

## 2. Related Work

### 2.1. Image Compression

A lot of image compression algorithms have been proposed in the past decades [34, 27, 1]. These methods heavily rely on handcrafted techniques. For example, the JPEG standard linearly maps the pixels to another representation by using DCT, and quantizes the corresponding coefficients before entropy coding[34]. One disadvantage is that these modules are separately optimized and may not achieve optimal compression performance.

Recently, DNN based image compression methods have attracted more and more attention [32, 33, 11, 12, 31, 8, 19, 26, 22, 9]. In [32], recurrent neural networks (RNNs) are utilized to build a progressive image compression scheme. Their approach is further improved by using other RNN architectures, learning based entropy model, priming and spatial adaptive bitrate allocation [33, 17]. Other methods employed the CNNs to design an auto-encoder style network for image compression [11, 12, 31]. To optimize the neural network, the work in [32, 33, 17] only tried to minimize the distortion (*e.g.*, mean square error) between original frames and reconstructed frames without considering the number of bits used for compression. Rate-distortion optimization technique was adopted in [11, 12, 31, 19] for higher compression efficiency by introducing the number of bits in the optimization procedure. To estimate the bit rates, context models are learned for the adaptive arithmetic coding method in [26, 19, 22], while non-adaptive arithmetic coding is used in [11, 31]. In addition, other techniques such as generalized divisive normalization (GDN) [11], multi-scale image decomposition [26], adversarial training [26], importance map [19, 22] and intra prediction [23, 10] are proposed to improve the image compression performance. These existing works are important building blocks for our video compression network.

Although current state-of-the-art DNN based image compression methods can outperform traditional image compression approaches [34, 1], they only utilize the spatial information. Therefore, it is not efficient enough to directly apply them to compress video sequences, which are highly redundant in the temporal domain.

### 2.2. Video Compression

In the past decades, several traditional video compression algorithms have been proposed, such as H.264 [36] and H.265 [29]. Most of these algorithms follow the predictive coding architecture, in which a block is predicted from neighbouring frames. Although they provide highly efficient compression performance, they are manually designed

and cannot be jointly optimized in an end-to-end way.

For the video compression task, a lot of DNN based methods have been proposed for intra prediction and residual coding[13] , mode decision [20], entropy coding [28], post-processing [21]. These methods are used to improve the performance of one particular module of the traditional video compression algorithms instead of building an end-to-end compression scheme. In [14], Chen *et al.* proposed a block based learning approach for video compression. However, it will inevitably generate blockness artifact in the boundary between blocks, especially for low bit rate compression. And instead of employing the motion information between current frame and reference frame, they used the motion information propagated from previous reconstructed frames through traditional block based motion estimation, which will degrade compression performance.

The most related work is the RNN based approach in [37], where video compression is formulated as image interpolation. However, the motion information in their approach is also generated by traditional block based motion estimation, which is encoded by the existing non-deep learning based image compression method [5]. In other words, estimation and compression of motion are not accomplished by deep model and jointly optimized with other components. In addition, the video codec in [37] only aims at minimizing the distortion (*i.e.*, mean square error) between the original frame and reconstructed frame without considering rate-distortion trade-off in the training procedure, which may degrade the compression efficiency. In comparison, in our network, motion estimation and compression are achieved by DNN, which is jointly optimized with other components by considering the rate-distortion trade-off of the whole compression system.

### 2.3. Motion Estimation

Motion estimation is a key component in the video compression system. Traditional video codecs use the block based motion estimation algorithm [36], which well supports hardware implementation.

In the computer vision tasks, optical flow is widely used to exploit temporal relationship. Recently, a lot of learning based optical flow estimation methods [15, 25, 30] have been proposed. These approaches motivate us to integrate optical flow estimation into our end-to-end learning framework. Compared with block based motion estimation method in the existing video compression approaches, learning based optical flow methods can provide accurate motion information at pixel-level, which can be also optimized in an end-to-end manner. However, much more bits are required to compress motion information if optical flow values are encoded by traditional video compression appraoches.

## 3. Proposed Method

**Introduction of Notations.** Let $\mathcal{V} = \{x_1, x_2, ..., x_{t-1}, x_t, ...\}$ denote the current video sequences, where $x_t$ is the frame at time step $t$. The predicted frame is denoted as $\bar{x}_t$ and the reconstructed/decoded frame is denoted as $\hat{x}_t$. $r_t$ represents the residual (error) between the original frame $x_t$ and the predicted frame $\bar{x}_t$. $\hat{r}_t$ represents the reconstructed/decoded residual. In order to reduce temporal redundancy, motion information is required. Among them, $v_t$ represents the motion vector or optical flow value. And $\hat{v}_t$ is its corresponding reconstructed version. Linear or nonlinear transform can be used to improve the compression efficiency. Therefore, residual information $r_t$ is transformed to $y_t$, and motion information $v_t$ can be transformed to $m_t$, $\hat{r}_t$ and $\hat{m}_t$ are the corresponding quantized versions, respectively.

### 3.1. Brief Introduction of Video Compression

In this section, we give a brief introduction of video compression. More details are provided in [36, 29]. Generally, the video compression encoder generates the bitstream based on the input current frames. And the decoder reconstructs the video frames based on the received bitstreams. In Fig. 2, all the modules are included in the encoder side while blue color modules are not included in the decoder side.

The classic framework of video compression in Fig. 2(a) follows the predict-transform architecture. Specifically, the input frame $x_t$ is split into a set of blocks, *i.e.*, square regions, of the same size (*e.g.*, $8 \times 8$). The encoding procedure of the traditional video compression algorithm in the encoder side is shown as follows,

**Step 1. Motion estimation.** Estimate the motion between the current frame $x_t$ and the previous reconstructed frame $\hat{x}_{t-1}$. The corresponding motion vector $v_t$ for each block is obtained.

**Step 2. Motion compensation.** The predicted frame $\bar{x}_t$ is obtained by copying the corresponding pixels in the previous reconstructed frame to the current frame based on the motion vector $v_t$ defined in Step 1. The residual $r_t$ between the original frame $x_t$ and the predicted frame $\bar{x}_t$ is obtained as $r_t = x_t - \bar{x}_t$.

**Step 3. Transform and quantization.** The residual $r_t$ from Step 2 is quantized to $\hat{y}_t$. A linear transform (*e.g.*, DCT) is used before quantization for better compression performance.

**Step 4. Inverse transform.** The quantized result $\hat{y}_t$ in Step 3 is used by inverse transform for obtaining the reconstructed residual $\hat{r}_t$.

**Step 5. Entropy coding.** Both the motion vector $v_t$ in Step 1 and the quantized result $\hat{y}_t$ in Step 3 are encoded into bits by the entropy coding method and sent to the decoder.

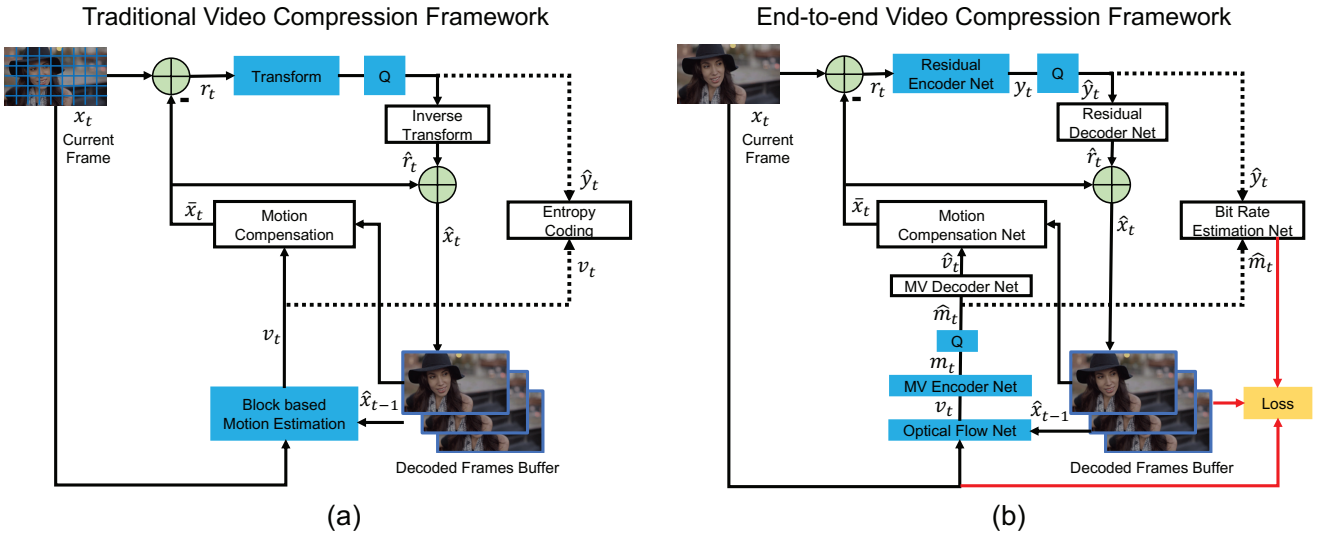



Figure 2: (a): The predictive coding architecture used by the traditional video codec H.264 [36] or H.265 [29]. (b): The proposed end-to-end video compression network. The modules with blue color are not included in the decoder side.

**Step 6. Frame reconstruction.** The reconstructed frame $\hat{x}_t$ is obtained by adding $\bar{x}_t$ in Step 2 and $\hat{r}_t$ in Step 4, *i.e.* $\hat{x}_t = \hat{r}_t + \bar{x}_t$. The reconstructed frame will be used by the $(t+1)$th frame at Step 1 for motion estimation.

For the decoder, based on the bits provided by the encoder at Step 5, motion compensation at Step 2, inverse quantization at Step 4, and then frame reconstruction at Step 6 are performed to obtain the reconstructed frame $\hat{x}_t$.

### 3.2. Overview of the Proposed Method

Fig. 2 (b) provides a high-level overview of our end-to-end video compression framework. There is one-to-one correspondence between the traditional video compression framework and our proposed deep learning based framework. The relationship and brief summarization on the differences are introduced as follows:

**Step N1. Motion estimation and compression.** We use a CNN model to estimate the optical flow [25], which is considered as motion information $v_t$. Instead of directly encoding the raw optical flow values, an MV encoder-decoder network is proposed in Fig. 2 to compress and decode the optical flow values, in which the quantized motion representation is denoted as $\hat{m}_t$. Then the corresponding reconstructed motion information $\hat{v}_t$ can be decoded by using the MV decoder net. Details are given in Section 3.3.

**Step N2. Motion compensation.** A motion compensation network is designed to obtain the predicted frame $\bar{x}_t$ based on the optical flow obtained in Step N1. More information is provided in Section 3.4.

**Step N3-N4. Transform, quantization and inverse transform.** We replace the linear transform in Step 3 by using a highly non-linear residual encoder-decoder network, and the residual $r_t$ is non-linearly maped to the representation $y_t$. Then $y_t$ is quantized to $\hat{y}_t$. In order to build an end-to-end training scheme, we use the quantization method in [11]. The quantized representation $\hat{y}_t$ is fed into the residual decoder network to obtain the reconstructed residual $\hat{r}_t$. Details are presented in Section 3.5 and 3.6.

**Step N5. Entropy coding.** At the testing stage, the quantized motion representation $\hat{m}_t$ from Step N1 and the residual representation $\hat{y}_t$ from Step N3 are coded into bits and sent to the decoder. At the training stage, to estimate the number of bits cost in our proposed approach, we use the CNNs (Bit rate estimation net in Fig. 2) to obtain the probability distribution of each symbol in $\hat{m}_t$ and $\hat{y}_t$. More information is provided in Section 3.6.

**Step N6. Frame reconstruction.** It is the same as Step 6 in Section 3.1.

### 3.3. MV Encoder and Decoder Network

In order to compress motion information at Step N1, we design a CNN to transform the optical flow to the corresponding representations for better encoding. Specifically, we utilize an auto-encoder style network to compress the optical flow, which is first proposed by [11] for the image compression task. The whole MV compression network is shown in Fig. 3. The optical flow $v_t$ is fed into a series of convolution operation and nonlinear transform. The number of output channels for convolution (deconvolution) is 128 except for the last deconvolution layer, which is equal to 2. Given optical flow $v_t$ with the size of $M \times N \times 2$, the MV encoder will generate the motion representation $m_t$ with the size of $M/16 \times N/16 \times 128$. Then $m_t$ is quantized to $\hat{m}_t$. The MV decoder receives the quantized representation and reconstruct motion information $\hat{v}_t$. In addition, the quantized representation $\hat{m}_t$ will be used for entropy coding.

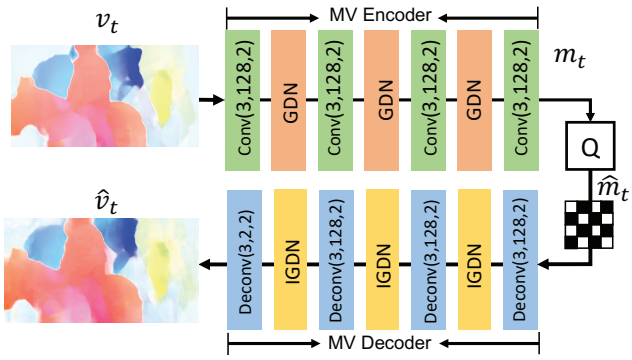



Figure 3: Our MV Encoder-decoder network. Conv(3,128,2) represents the convolution operation with the kernel size of 3x3, the output channel of 128 and the stride of 2. GDN/IGDN [11] is the nonlinear transform function. The binary feature map is only used for illustration.

### 3.4. Motion Compensation Network

Given the previous reconstructed frame $\hat{x}_{t-1}$ and the motion vector $\hat{v}_t$, the motion compensation network obtains the predicted frame $\bar{x}_t$, which is expected to as close to the current frame $x_t$ as possible. First, the previous frame $\hat{x}_{t-1}$ is warped to the current frame based on the motion information $\hat{v}_t$. The warped frame still has artifacts. To remove the artifacts, we concatenate the warped frame $w(\hat{x}_{t-1}, \hat{v}_t)$, the reference frame $\hat{x}_{t-1}$, and the motion vector $\hat{v}_t$ as the input, then feed them into another CNN to obtain the refined predicted frame $\bar{x}_t$. The overall architecture of the proposed network is shown in Fig. 4. The detail of the CNN in Fig. 4 is provided in supplementary material. Our proposed method is a pixel-wise motion compensation approach, which can provide more accurate temporal information and avoid the blockness artifact in the traditional block based motion compensation method. It means that we do not need the hand-crafted loop filter or the sample adaptive offset technique [36, 29] for post processing.

### 3.5. Residual Encoder and Decoder Network

The residual information $r_t$ between the original frame $x_t$ and the predicted frame $\bar{x}_t$ is encoded by the residual encoder network as shown in Fig. 2. In this paper, we rely on the highly non-linear neural network in [12] to transform the residuals to the corresponding latent representation. Compared with discrete cosine transform in the traditional video compression system, our approach can better exploit the power of non-linear transform and achieve higher compression efficiency.

### 3.6. Training Strategy

**Loss Function.** The goal of our video compression framework is to minimize the number of bits used for encoding the video, while at the same time reduce the distortion between the original input frame $x_t$ and the reconstructed frame $\hat{x}_t$. Therefore, we propose the following rate-distortion optimization problem,

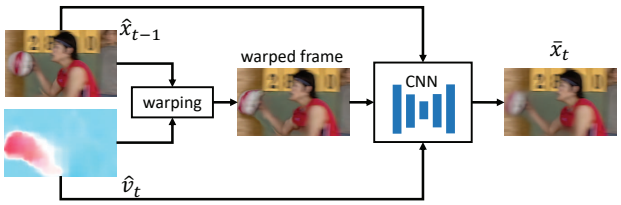



Figure 4: Our Motion Compensation Network.

$$D + \lambda R = d(x_t, \hat{x}_t) + \lambda(H(\hat{m}_t) + H(\hat{y}_t)), \quad (1)$$

where $d(x_t, \hat{x}_t)$ denotes the distortion between $x_t$ and $\hat{x}_t$, and we use mean square error (MSE) in our implementation. $H(\cdot)$ represents the number of bits used for encoding the representations. In our approach, both residual representation $\hat{y}_t$ and motion representation $\hat{m}_t$ should be encoded into the bitstreams. $\lambda$ is the Lagrange multiplier that determines the trade-off between the number of bits and distortion. As shown in Fig. 2(b), the reconstructed frame $\hat{x}_t$, the original frame $x_t$ and the estimated bits are input to the loss function.

**Quantization.** Latent representations such as residual representation $y_t$ and motion representation $m_t$ are required to be quantized before entropy coding. However, quantization operation is not differential, which makes end-to-end training impossible. To address this problem, a lot of methods have been proposed [32, 8, 11]. In this paper, we use the method in [11] and replace the quantization operation by adding uniform noise in the training stage. Take $y_t$ as an example, the quantized representation $\hat{y}_t$ in the training stage is approximated by adding uniform noise to $y_t$, *i.e.*, $\hat{y}_t = y_t + \eta$, where $\eta$ is uniform noise. In the inference stage, we use the rounding operation directly, *i.e.*, $\hat{y}_t = round(y_t)$.

**Bit Rate Estimation.** In order to optimize the whole network for both number of bits and distortion, we need to obtain the bit rate ($H(\hat{y}_t)$ and $H(\hat{m}_t)$) of the generated latent representations ($\hat{y}_t$ and $\hat{m}_t$). The correct measure for bitrate is the entropy of the corresponding latent representation symbols. Therefore, we can estimate the probability distributions of $\hat{y}_t$ and $\hat{m}_t$, and then obtain the corresponding entropy. In this paper, we use the CNNs in [12] to estimate the distributions.

**Buffering Previous Frames.** As shown in Fig. 2, the previous reconstructed frame $\hat{x}_{t-1}$ is required in the motion estimation and motion compensation network when compressing the current frame. However, the previous reconstructed frame $\hat{x}_{t-1}$ is the output of our network for the previous frame, which is based on the reconstructed frame $\hat{x}_{t-2}$, and so on. Therefore, the frames $x_1, \ldots, x_{t-1}$ might be required during the training procedure for the frame $x_t$,

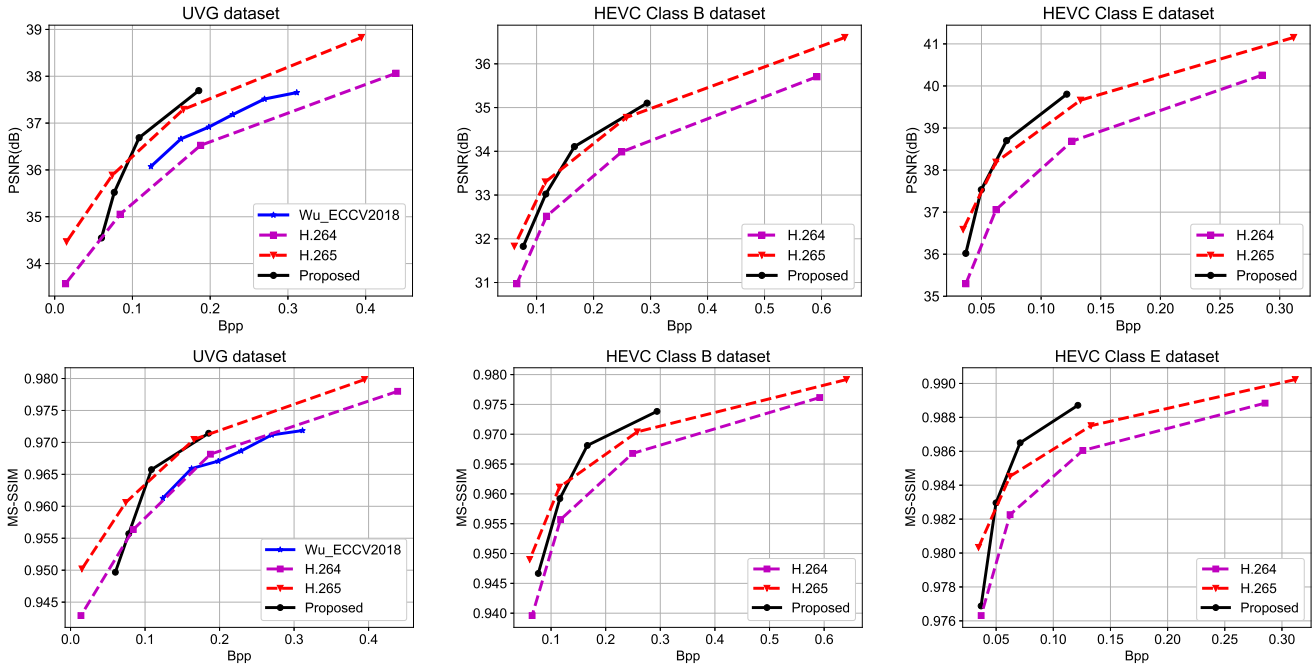



Figure 5: Comparsion between our proposed method with the learning based video codec in [37], H.264 [36] and H.265 [29]. Our method outperforms H.264 when measured by both PSNR ans MS-SSIM. Meanwhile, our method achieves similar or better compression performance when compared with H.265 in terms of MS-SSIM.

which reduces the variation of training samples in a mini-batch and could be impossible to be stored in a GPU when $t$ is large. To solve this problem, we adopt an on line updating strategy. Specifically, the reconstructed frame $\hat{x}_t$ in each iteration will be saved in a buffer. In the following iterations, $\hat{x}_t$ in the buffer will be used for motion estimation and motion compensation when encoding $x_{t+1}$. Therefore, each training sample in the buffer will be updated in an epoch. In this way, we can optimize and store one frame for a video clip in each iteration, which is more efficient.

# 4. Experiments

## 4.1. Experimental Setup

**Datasets.** We train the proposed video compression framework using the Vimeo-90k dataset [39], which is recently built for evaluating different video processing tasks, such as video denoising and video super-resolution. It consists of 89,800 independent clips that are different from each other in content.

To report the performance of our proposed method, we evaluate our proposed algorithm on the UVG dataset [4], and the HEVC Standard Test Sequences (Class B, Class C, Class D and Class E) [29]. The content and resolution of these datasets are diversified and they are widely used to measure the performance of video compression algorithms.

**Evaluation Method** To measure the distortion of the reconstructed frames, we use two evaluation metrics: PSNR and MS-SSIM [35]. MS-SSIM correlates better with human perception of distortion than PSNR. To measure the number of bits for encoding the representations, we use bits per pixel(Bpp) to represent the required bits for each pixel in the current frame.

**Implementation Details** We train four models with different $\lambda$ ($\lambda = 256, 512, 1024, 2048$). For each model, we use the Adam optimizer [18] by setting the initial learning rate as 0.0001, $\beta_1$ as 0.9 and $\beta_2$ as 0.999, respectively. The learning rate is divided by 10 when the loss becomes stable. The mini-batch size is set as 4. The resolution of training images is $256 \times 256$. The motion estimation module is initialized with the pretrained weights in [25]. The whole system is implemented based on Tensorflow and it takes about 7 days to train the whole network using two Titan X GPUs.

## 4.2. Experimental Results

In this section, both H.264 [36] and H.265 [29] are included for comparison. In addition, a learning based video compression system in [37], denoted by Wu_ECCV2018, is also included for comparison. To generate the compressed frames by the H.264 and H.265, we follow the setting in [37] and use the FFmpeg with *very fast* mode. The GOP sizes for the UVG dataset and HEVC dataset are 12 and 10, respectively. Please refer to supplementary material for more details about the H.264 or H.265 settings in our experiments.

Fig. 5 shows the experimental results on the UVG dataset, the HEVC Class B and Class E datasets. The results for HEVC Class C and Class D are provided in sup-

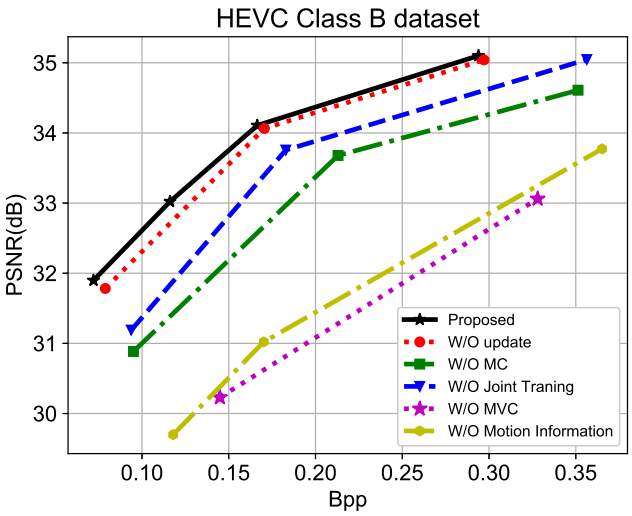


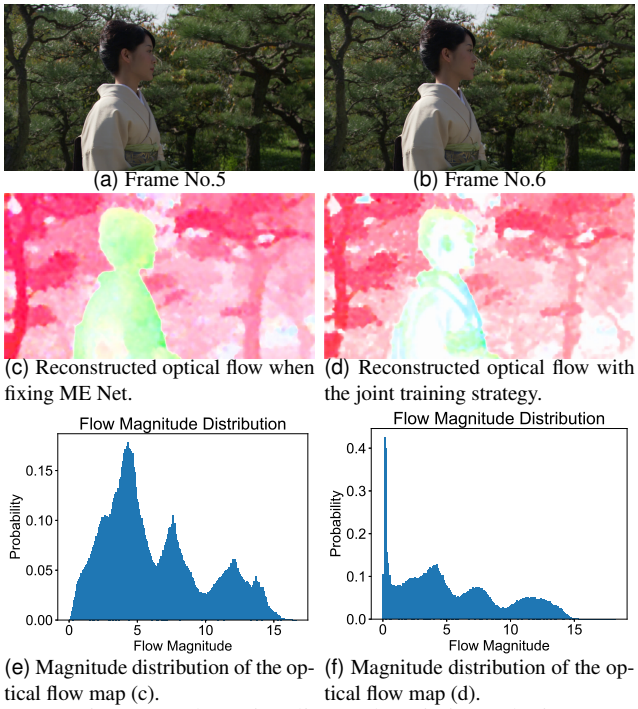

Figure 6: Ablation study. We report the compression performance in the following settings. 1. The strategy of buffering previous frame is not adopted(W/O update). 2. Motion compensation network is removed (W/O MC). 3. Motion estimation module is not jointly optimized (W/O Joint Training). 4. Motion compression network is removed (W/O MVC). 5. Without relying on motion information (W/O Motion Information).

plementary material. It is obvious that our method outperforms the recent work on video compression [37] by a large margin. On the UVG dataset, the proposed method achieved about 0.6dB gain at the same Bpp level. It should be mentioned that our method only uses one previous reference frame while the work by Wu *et al.* [37] utilizes bidirectional frame prediction and requires two neighbouring frames. Therefore, it is possible to further improve the compression performance of our framework by exploiting temporal information from multiple reference frames.

On most of the datasets, our proposed framework outperforms the H.264 standard when measured by PSNR and MS-SSIM. In addition, our method achieves similar or better compression performance when compared with H.265 in terms of MS-SSIM. As mentioned before, the distortion term in our loss function is measured by MSE. Nevertheless, our method can still provide reasonable visual quality in terms of MS-SSIM even if our framework is optimized based on MSE.

### 4.3. Ablation Study and Model Analysis

**Motion Estimation.** In our proposed method, we exploit the advantage of the end-to-end training strategy and optimize the motion estimation module within the whole network. Therefore, based on rate-distortion optimization, the optical flow in our system is expected to be more compressible, leading to more accurate warped frames. To demonstrate the effectiveness, we perform a experiment by fixing




(a) Frame No.5 (b) Frame No.6

(c) Reconstructed optical flow when fixing ME Net. (d) Reconstructed optical flow with the joint training strategy.

(e) Magnitude distribution of the optical flow map (c). (f) Magnitude distribution of the optical flow map (d).

Figure 7: Flow visualize and statistic analysis.

the parameters of the initialized motion estimation module in the whole training stage. In this case, the motion estimation module is pretrained only for estimating optical flow more accurately, but not for optimal rate-distortion. Experimental result in Fig. 6 shows that our approach with joint training for motion estimation can improve the performance significantly when compared with the approach by fixing motion estimation, which is the denoted by *W/O Joint Training* in Fig. 6 (see the blue curve).

We report the average bits costs for encoding the optical flow and the corresponding PSNR of the warped frame in Table 1. Specifically, when the motion estimation module is fixed during the training stage, it needs 0.044bpp to encode the generated optical flow and the corresponding PSNR of the warped frame is 27.33db. In contrast, we need 0.029bpp to encode the optical flow in our proposed method, and the PSNR of warped frame is higher (28.17dB). Therefore, the joint learning strategy not only saves the number of bits required for encoding the motion, but also has better warped image quality. These experimental results clearly show that putting motion estimation into the rate-distortion optimization improves compression performance.

In Fig. 7, we provide further visual comparisons. Fig. 7 (a) and (b) represent the frame 5 and frame 6 from the Kimono sequence. Fig. 7 (c) denotes the reconstructed optical flow map when the optical flow network is fixed during the training procedure. Fig. 7 (d) represents the reconstructed optical flow map after using the joint training strategy. Fig. 7 (e) and (f) are the corresponding probability distributions of optical flow magnitudes. It can be observed that the re-

| Fix ME | | W/O MVC | | Ours | |
|---|---|---|---|---|---|
| Bpp | PSNR | Bpp | PSNR | Bpp | PSNR |
| 0.044 | 27.33 | 0.20 | 24.32 | 0.029 | 28.17 |

Table 1: The bit cost for encoding optical flow and the corresponding PSNR of warped frame from optical flow for different setting are provided.

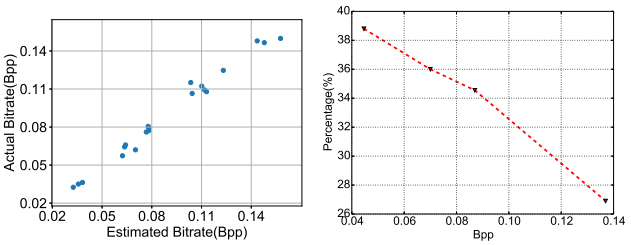
constructed optical flow by using our method contains more pixels with zero flow magnitude (e.g., in the area of human body). Although zero value is not the true optical flow value in these areas, our method can still generate accurate motion compensated results in the homogeneous region. More importantly, the optical flow map with more zero magnitudes requires much less bits for encoding. For example, it needs 0.045bpp for encoding the optical flow map in Fig. 7 (c) while it only needs 0.038bpp for encoding optical flow map in Fig. 7 (d).

It should be mentioned that in the H.264 [36] or H.265 [29], a lot of motion vectors are assigned to zero for achieving better compression performance. Surprisingly, our proposed framework can learn a similar motion distribution without relying on any complex hand-crafted motion estimation strategy as in [36, 29].

**Motion Compensation.** In this paper, the motion compensation network is utilized to refine the warped frame based on the estimated optical flow. To evaluate the effectiveness of this module, we perform another experiment by removing the motion compensation network in the proposed system. Experimental results of the alternative approach denoted by *W/O MC* (see the green curve in Fig. 6) show that the PSNR without the motion compensation network will drop by about 1.0 dB at the same bpp level.

**Updating Strategy.** As mentioned in Section 3.6, we use an on-line buffer to store previously reconstructed frames $\hat{x}_{t-1}$ in the training stage when encoding the current frame $x_t$. We also report the compression performance when the previous reconstructed frame $\hat{x}_{t-1}$ is directly replaced by the previous original frame $x_{t-1}$ in the training stage. This result of the alternative approach denoted by *W/O update* (see the red curve ) is shown in Fig. 6. It demonstrates that the buffering strategy can provide about 0.2dB gain at the same bpp level.

**MV Encoder and Decoder Network.** In our proposed framework, we design a CNN model to compress the optical flow and encode the corresponding motion representations. It is also feasible to directly quantize the raw optical flow values and encode it without using any CNN. We perform a new experiment by removing the MV encoder and decoder network. The experimental result in Fig. 6 shows that the PSNR of the alternative approach denoted by *W/O MVC* (see the magenta curve ) will drop by more than 2 dB after removing the motion compression network. In addition, the bit cost for encoding the optical flow in this setting and the corresponding PSNR of the warped frame are also provided in Table 1 (denoted by W/O MVC). It is obvious that it requires much more bits (0.20Bpp) to directly encode raw optical flow values and the corresponding PSNR(24.43dB) is much worse than our proposed method(28.17dB). Therefore, compression of motion is crucial when optical flow is used for estimating motion.




(a) Actual and estimated bit rate. (b) Motion information percentages.

Figure 8: Bit rate analysis.

**Motion Information.** In Fig. 2(b), we also investigate the setting which only retains the residual encoder and decoder network. Treating each frame independently without using any motion estimation approach (see the yellow curve denoted by *W/O Motion Information*) leads to more than 2 dB drop in PSNR when compared with our method.

**Running Time and Model Complexity.** The total number of parameters of our proposed end-to-end video compression framework is about 11M. It takes about 0.45s(2.22fps) to encode a frame with the resolution of 832x480. The corresponding encoding speeds of H.264 and H.265 based on the official software JM [2] and HM [3] are 0.42fps and 0.07fps, respectively. Although the commercial codec x264 [6] and x265 [7] can provide much faster encoding speed than ours, they need a lot of code optimization. Correspondingly, recent deep model compression approaches are off-the-shelf for making the deep model much faster, which is beyond the scope of this paper.

**Bit Rate Analysis.** In this paper, we use a probability estimation network in [12] to estimate the bit rate for encoding motion information and residual information. To verify the reliability, in Fig. 8(a) we compare the estimated bit rate and the actual bit rate by using arithmetic coding. It is obvious that the estimated bit rate is closed to the actual bit rate. In addition, we further investigate on the components of bit rate. In Fig. 8(b), the percentage of motion information occupied in the whole bitstream is reported. The motion takes $30\% \sim 40\%$ of the bitstream. It demonstrates that the encoder uses more bits to encode the residual information when the bpp is larger.

## 5. Conclusion

In this paper, we have proposed the fully end-to-end deep learning framework for video compression. Our framework inherits the advantages of both classic predictive coding scheme in the traditional video compression standards and the powerful non-linear representation ability from DNNs.

Experimental results show that our approach outperforms the widely used H.264 video compression standard and the recent learning based video compression system. The work provides a promising framework for applying deep neural network for video compression. Based on the proposed framework, other new techniques for optical flow, image compression, bi-directional prediction and rate control can be readily plugged into this framework.

## References


[1] F. bellard, bpg image format. http://bellard.org/bpg/. Accessed: 2018-10-30. 1, 2

[2] The h.264/avc reference software. http://iphome.hhi.de/suehring/. Accessed: 2018-10-30. 8

[3] Hevc test model (hm). https://hevc.hhi.fraunhofer.de/HM-doc/. Accessed: 2018-10-30. 8

[4] Ultra video group test sequences. http://ultravideo.cs.tut.fi. Accessed: 2018-10-30. 6

[5] Webp. https://developers.google.com/speed/webp/. Accessed: 2018-10-30. 3

[6] x264, the best h.264/avc encoder. https://www.videolan.org/developers/x264.html. Accessed: 2018-10-30. 8

[7] x265 hevc encoder / h.265 video codec. http://x265.org. Accessed: 2018-10-30. 8

[8] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *NIPS*, pages 1141–1151, 2017. 1, 2, 5

[9] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. Van Gool. Generative adversarial networks for extreme learned image compression. *arXiv preprint arXiv:1804.02958*, 2018. 1, 2

[10] M. H. Baig, V. Koltun, and L. Torresani. Learning to inpaint for image compression. In *NIPS*, pages 1246–1255, 2017. 2

[11] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016. 1, 2, 4, 5

[12] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*, 2018. 1, 2, 5, 8

[13] T. Chen, H. Liu, Q. Shen, T. Yue, X. Cao, and Z. Ma. Deepcoder: A deep neural network based video compression. In *VCIP*, pages 1–4. IEEE, 2017. 3

[14] Z. Chen, T. He, X. Jin, and F. Wu. Learning for video compression. *arXiv preprint arXiv:1804.09869*, 2018. 3

[15] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, pages 2758–2766, 2015. 3

[16] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[17] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. Jin Hwang, J. Shor, and G. Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *CVPR*, June 2018. 1, 2

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[19] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang. Learning convolutional networks for content-weighted image compression. In *CVPR*, June 2018. 1, 2

[20] Z. Liu, X. Yu, Y. Gao, S. Chen, X. Ji, and D. Wang. Cu partition mode decision for hevc hardwired intra encoder using convolution neural network. *TIP*, 25(11):5088–5103, 2016. 3

[21] G. Lu, W. Ouyang, D. Xu, X. Zhang, Z. Gao, and M.-T. Sun. Deep kalman filtering network for video compression artifact reduction. In *ECCV*, September 2018. 3

[22] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Conditional probability models for deep image compression. In *CVPR*, number 2, page 3, 2018. 2

[23] D. Minnen, G. Toderici, M. Covell, T. Chinen, N. Johnston, J. Shor, S. J. Hwang, D. Vincent, and S. Singh. Spatially adaptive image compression using a tiled deep network. In *ICIP*, pages 2796–2800. IEEE, 2017. 2

[24] C. V. networking Index. Forecast and methodology, 2016-2021, white paper. *San Jose, CA, USA*, 1, 2016. 1

[25] A. Ranjan and M. J. Black. Optical flow estimation using a spatial pyramid network. In *CVPR*, volume 2, page 2. IEEE, 2017. 3, 4, 6

[26] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *ICML*, 2017. 1, 2

[27] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal Processing Magazine*, 18(5):36–58, 2001. 1, 2

[28] R. Song, D. Liu, H. Li, and F. Wu. Neural network-based arithmetic coding of intra prediction modes in hevc. In *VCIP*, pages 1–4. IEEE, 2017. 3

[29] G. J. Sullivan, J.-R. Ohm, W.-J. Han, T. Wiegand, et al. Overview of the high efficiency video coding(hevc) standard. *TCSVT*, 22(12):1649–1668, 2012. 1, 2, 3, 4, 5, 6, 8

[30] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *CVPR*, pages 8934–8943, 2018. 3

[31] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017. 1, 2

[32] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*, 2015. 1, 2, 5

[33] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. In *CVPR*, pages 5435–5443, 2017. 1, 2

[34] G. K. Wallace. The jpeg still picture compression standard. *IEEE Transactions on Consumer Electronics*, 38(1):xviii–xxxiv, 1992. 1, 2

[35] Z. Wang, E. Simoncelli, A. Bovik, et al. Multi-scale structural similarity for image quality assessment. In *ASILOMAR*

*CONFERENCE ON SIGNALS SYSTEMS AND COMPUTERS*, volume 2, pages 1398–1402. IEEE; 1998, 2003. 2, 6

[36] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the h. 264/avc video coding standard. *TCSVT*, 13(7):560–576, 2003. 1, 2, 3, 4, 5, 6, 8

[37] C.-Y. Wu, N. Singhal, and P. Krahenbuhl. Video compression through image interpolation. In *ECCV*, September 2018. 3, 6, 7

[38] C.-Y. Wu, M. Zaheer, H. Hu, R. Manmatha, A. J. Smola, and P. Krähenbühl. Compressed video action recognition. In *CVPR*, pages 6026–6035, 2018. 1

[39] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman. Video enhancement with task-oriented flow. *arXiv preprint arXiv:1711.09078*, 2017. 2, 6

# Appendix A. Experimental Results

## A.1. BDBR and BD-PSNR(BD-MSSSIM) Results

In the video compression task, BDBR and BD-PSNR (BD-MSSSIM) are widely used to evaluate the performance [1] of different video compression systems. BDBR represents the average percentage of bit rate savings when compared with the baseline algorithm at the same PSNR (MS-SSIM). BD-PSNR (BD-MSSSIM) represents the gain (dB) when compared with the baseline algorithm at the same bit rate.

In Table 1, we provide the BDBR and BD-PSNR(BD-MSSSIM) results of H.265 and the our proposed method DVC when compared with H.264. When the distortion is measured by PSNR, our proposed model saves 19.22% bit rate, while H.265 saves 25.06% bit rate. When measured by MS-SSIM, our proposed method can save more than 29% bit rate, while H.265 only saves 21.73%. It clearly demonstrates that our proposed method outperforms H.264 in terms of PSNR and MS-SSIM. Meanwhile, the performance of our method is comparable or even better than H.265 in terms of PSNR and MS-SSIM, respectively.

## A.2. Compression Performance on the HEVC Class C and Class D

In the submitted paper, we provide the performance on the HEVC Class B and Class E datasets. In this section, we also compare our proposed method with the traditional video compression algorithms H.264 and H.265 on the HEVC Class C and Class D datasets in Figure 1. Our proposed method still outperforms the H.264 algorithm in terms of PSNR and MS-SSIM. According to Table 1, the newly proposed DVC model saves 3.88% and 9.68% bit rates when compared with H.264 on the HEVC Class C and Class D datasets, respectively.

## A.3. Result Using Kinetics Dataset as the Train Dataset

Wu *et al.* utilized the Kinetics datasets to train the video compression mdoel in [2]. In Figure 2, we also report the performance of our model when using the Kinetics dataset as the train dataset, which is denoted as *Ours_Kinetics* (see the green curve). It is obvious that our method trained based on Kinetics dataset outperforms the H.264 and the baseline method [2].

# Appendix B. Experimental Settings

## B.1. H.264 and H.265 Setting

In order to generate the compressed videos from H.264 and H.265, we follow the setting in [2] (confirmed from the first author) and use the FFmpeg with the *very fast* mode. Given the uncompressed video sequence *Video.yuv* with the resolution of $W \times H$, the command line for generating H.264 compressed video is provided as follows,

*ffmpeg -y -pix_fmt yuv420p -s WxH -r FR -i Video.yuv -vframes N -c:v libx264 -preset veryfast -tune zerolatency -crf Q -g GOP-bf 2 -b_strategy 0 -sc_threshold 0 -loglevel debug output.mkv*

The command line for generating H.265 is provided as follows,

*ffmpeg -pix_fmt yuv420p -s WxH -r FR -i Video.yuv -vframes N -c:v libx265 -preset veryfast -tune zerolatency -x265-params "crf=Q:keyint=GOP:verbose=1" output.mkv*

Among them, $FR, N, Q, GOP$ represents the frame rate, the number of encoded frames, quality, GOP size, respectively. $N$ is set to 100 for the HEVC datasets. $Q$ is set as 15,19,23,27 in our settings. $GOP$ is set as 10 for the HEVC dataset and 12 for the UVG dataset.

## B.2. PSNR, MS-SSIM and Bpp

In our experiments, PSNR and MS-SSIM are evaluated on the RGB channel. We average the PSNRs from all vdieo frames in each sequence to obtain the corresponding PSNR of this sequence. Given the compressed frame with the resolution of $W \times H$ and the corresponding bit cost $R$, the Bpp is calculated as follows,

$$Bpp = R/W/H \tag{1}$$

# Appendix C. Network Architecture of Our Motion Compensation Network

The motion compensation network is shown in Figure 3. $w(\hat{x}_{t-1}, \hat{v}_t)$ represents the warped frame. We use residual block with pre-activation structure.

# Appendix D. References

[1] G. Bjontegaard. Calculation of average psnr differences between rd-curves. VCEG-M33, 2001.

[2] C.-Y. Wu, N. Singhal, and P. Krahenbuhl. Video compression through image interpolation. In ECCV, September 2018.

Table 1: BDBR and BD-PSNR(BD-MSSSIM) performances of H.265 and our DVC model when compared with H.264.

| Sequences | | H.265 | | | | Ours | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | PSNR | | MS-SSIM | | PSNR | | MS-SSIM | |
| | | BDBR (%) | BD-PSNR(dB) | BDBR (%) | BD-MS SSIM(dB) | BDBR (%) | BD-PSNR(dB) | BDBR (%) | BD-MS SSIM(dB) |
| ClassB | BasketballDrive | -44.37 | 1.15 | -39.80 | 0.87 | -23.17 | 0.59 | -22.21 | 0.51 |
| | BQTerrace | -28.99 | 0.68 | -25.96 | 0.50 | -25.12 | 0.54 | -19.52 | 0.36 |
| | Cactus | -30.15 | 0.68 | -26.93 | 0.47 | -39.53 | 0.94 | -41.71 | 0.86 |
| | Kimono | -38.81 | 1.19 | -35.31 | 0.97 | -40.70 | 1.23 | -33.00 | 0.92 |
| | ParkScene | -16.35 | 0.45 | -13.54 | 0.29 | -25.20 | 0.77 | -29.02 | 0.77 |
| Average | | **-31.73** | **0.83** | **-28.31** | **0.62** | **-30.75** | **0.81** | **-29.09** | **0.68** |
| ClassC | BasketballDrill | -35.08 | 1.69 | -34.04 | 1.41 | -24.47 | 1.05 | -27.18 | 1.18 |
| | BQMall | -19.70 | 0.84 | -17.57 | 0.60 | 26.13 | -0.72 | -18.85 | 0.67 |
| | PartyScene | -13.41 | 0.60 | -13.36 | 0.53 | -9.14 | 0.29 | -37.18 | 1.61 |
| | RaceHorses | -17.28 | 0.69 | -17.01 | 0.57 | -8.06 | 0.19 | -29.24 | 1.05 |
| Average | | **-21.37** | **0.96** | **-20.50** | **0.78** | **-3.88** | **0.20** | **-28.11** | **1.13** |
| ClassD | BlowingBubbles | -12.51 | 0.50 | -10.28 | 0.35 | -17.79 | 0.62 | -35.44 | 1.53 |
| | BasketballPass | -19.26 | 0.99 | -17.98 | 0.85 | -0.39 | -0.01 | -20.53 | 1.01 |
| | BQSquare | -3.49 | 0.14 | 5.90 | -0.19 | -1.60 | 0.01 | -23.67 | 0.84 |
| | RaceHorses | -14.77 | 0.68 | -13.23 | 0.56 | -18.95 | 0.72 | -29.79 | 1.30 |
| Average | | **-12.51** | **0.58** | **-8.89** | **0.39** | **-9.68** | **0.34** | **-27.36** | **1.17** |
| ClassE | Vidyo1 | -37.12 | 1.11 | -31.67 | 0.55 | -36.05 | 1.20 | -36.80 | 0.72 |
| | Vidyo3 | -34.99 | 1.23 | -29.48 | 0.65 | -32.58 | 1.25 | -40.09 | 1.02 |
| | Vidyo4 | -34.71 | 1.05 | -27.41 | 0.61 | -30.84 | 1.03 | -24.84 | 0.66 |
| Average | | **-35.61** | **1.13** | **-29.52** | **0.61** | **-33.16** | **1.16** | **-33.91** | **0.80** |
| Average Over All Sequences | | **-25.06** | **0.85** | **-21.73** | **0.60** | **-19.22** | **0.61** | **-29.32** | **0.94** |

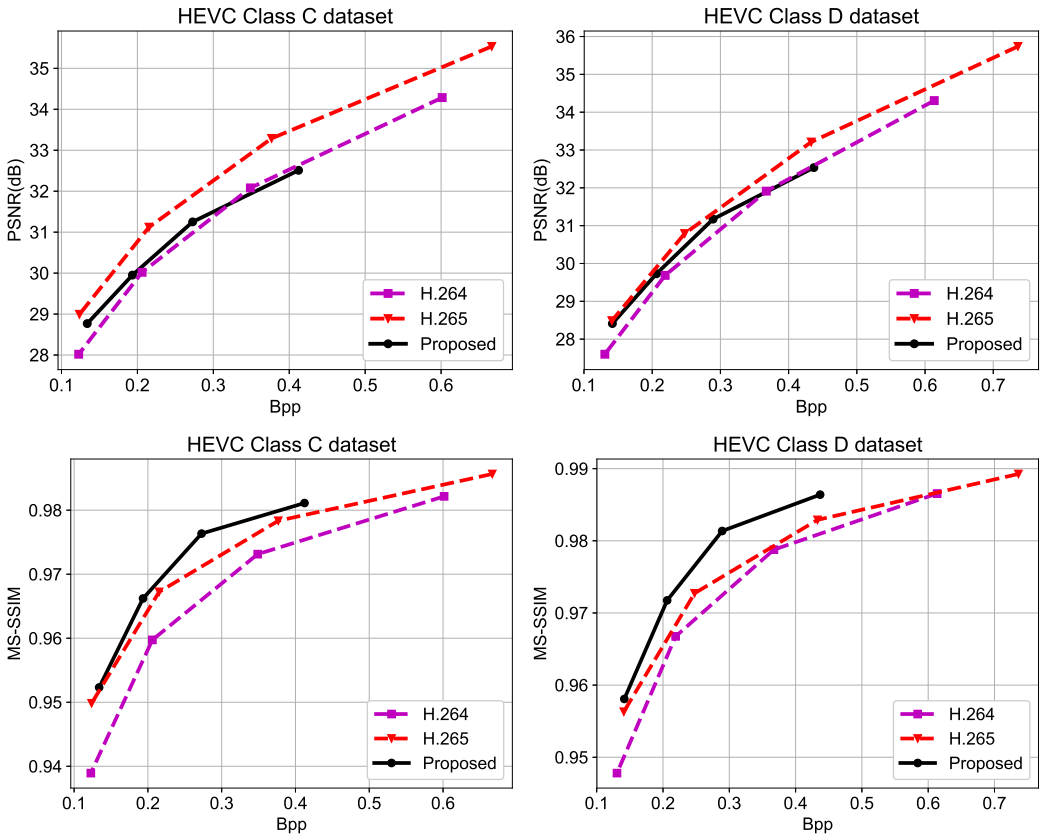



Figure 1: Compassion of our method with the traditional video compression methods H.264 and H.265 on the HEVC Class C and Class D datasets.

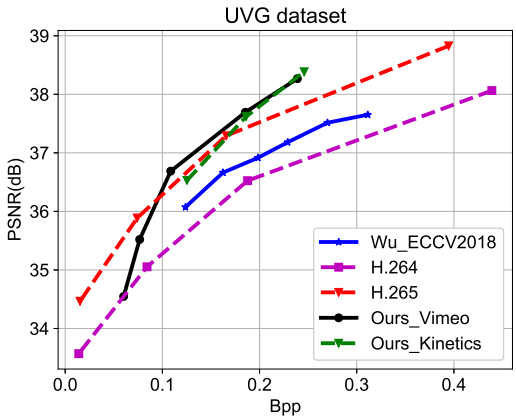



Figure 2: Compassion of our method with the traditional video compression methods H.264 and H.265 on the UVG dataset.

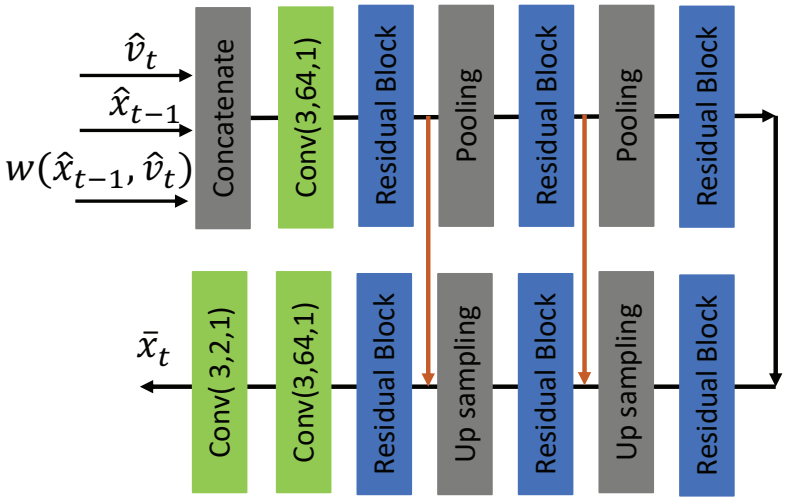



Figure 3: Network architecture of the motion compensation network.