Deep Frame Prediction for Video Coding

Hyomin Choi, *Student Member, IEEE,* and Ivan V. Bajić, *Senior Member, IEEE*

***Abstract*—We propose a novel frame prediction method using a deep neural network (DNN), with the goal of improving video coding efficiency. The proposed DNN makes use of decoded frames, at both encoder and decoder, to predict textures of the current coding block. Unlike conventional inter-prediction, the proposed method does not require any motion information to be transferred between the encoder and the decoder. Still, both uni-directional and bi-directional prediction are possible using the proposed DNN, which is enabled by the use of the temporal index channel, in addition to color channels. In this study, we developed a jointly trained DNN for both uni- and bi-directional prediction, as well as separate networks for uni- and bi-directional prediction, and compared the efficacy of both approaches. The proposed DNNs were compared with the conventional motion-compensated prediction in the latest video coding standard, HEVC, in terms of BD-Bitrate. The experiments show that the proposed joint DNN (for both uni- and bi-directional prediction) reduces the luminance bitrate by about 3.9%, 2.2%, and 2.2% in the Low delay P, Low delay, and Random access configurations, respectively. In addition, using the separately trained DNNs brings further bit savings of about 0.4%–0.8%.**

***Index Terms*—Video compression, frame prediction, texture prediction, deep neural network, deep learning.**

I. Introduction

With increasing demand for video services [1], [2], the need for more efficient video coding is also growing. In response to such demand, Joint Video Experts Team (JVET) was established after the latest video coding standard, High Efficiency Video Coding (HEVC), and has been exploring future video coding technologies. This year, JVET called for proposals for the next video coding standard, tentatively named Versatile Video Coding (VVC). Some of the submitted proposals considerably surpass the performance of HEVC in terms of both subjective and objective qualities [3]. Interestingly, several contributions proposed Deep Neural Network (DNN)-aided coding tools.

Recently, DNN-based coding tools have become a topic of interest, and experimental results superior to the conventional coding tools in terms of rate-distortion (RD) performance have started to appear in the literature. Some of these have also been discussed in JVET meetings as part of normative tools [4]. For example, DNN-based quality improvement for in-loop filtering and post-filtering have been actively researched in both academia and the standardization community [5]–[17]. Other aspects of video coding where DNNs have been suggested are intra and inter prediction [18]–[28], reduction of coding complexity [29]–[31], modeling the RD relationship [32], and replacing the entire coding pipeline by a DNN [33]. Given the variety of coding tools where DNNs have already made inroads, they appear to be an inevitable technological trend in video compression.

H. Choi and I. V. Bajić are with the School of Engineering Science, Simon Fraser University, BC, V5A 1S6, Canada. E-mail: chyomin@sfu.ca, ibajic@ensc.sfu.ca

For intra prediction, [18], [21] employed a Convolutional Neural Network (CNN) with neighbouring blocks (instead of reference lines) as inputs, to generate a predicted block. Moreover, [20] included a Recurrent Neural Network (RNN) into their predictor for improved accuracy. A fully connected network architecture (Multi-Layer Perceptron, MLP) is adopted in [19], [22] for intra prediction, and the input samples are also from the neighboring region. Specifically, [22] proposed a Discrete Cosine Transform (DCT)-domain predictor, where the output of the MLP is subject to inverse DCT to obtain the predicted pixels.

However, most of the coding gain in video compression comes from inter prediction. Recent works on inter prediction using DNNs include interpolation filters [23], [24], enhanced motion compensation [25]–[27], and texture prediction, which directly generates the predicted pixel values using reference samples/frames as inputs to the trained network [28] without transferring motion information. In this paper, we propose a DNN-based frame prediction architecture that is able to support both uni- and bi-directional prediction. Our work is inspired by recent progress on frame prediction (Section II), and is the first, to our knowledge, to develop a single DNN that can support both P and B frame coding.

The paper is organized as follows. Section II reviews recent related work on DNN-based inter prediction, and identifies the contributions of this paper. In Section III, presents the architecture of the proposed DNN and describes how the proposed DNN-based prediction can be used in video coding. Section V describes DNN training and examines the coding gain for various configurations. Finally, the paper is concluded in Section VI.

II. Neural network-aided inter prediction

Inter coding plays a crucial role in achieving high efficiency in video compression. Fractional-pel motion compensation requires a frame to be interpolated in between existing pixels to compensate for continuous motion of objects. In the HEVC, DCT-based interpolation filters provide quarter-pel precision, but their coefficients are non-adaptive, which may limit their effectiveness [34]. In [23], [24], CNNs have been used for content-adaptive interpolation. Specifically, Zhang *et al.* [23] proposed a half-pel interpolation filter based on a super-resolution network [35], while Yan *et al.* [24] proposed a different CNN-based interpolation filter.

Other studies that have looked at the use of neural networks in motion compensation include [25]–[27]. Huo *et al.* [25] proposed a CNN-based motion compensation refinement algorithm. The suggested network has the motion-compensated

block and its neighboring coded area as inputs, and it generates the refined prediction block. Considering the spatial correlation between adjacent pixels, especially for small blocks, this approach is able to reduce some artifacts along the block boundaries. The approach is less suitable for larger blocks, but it is applicable to both uni- and bi-prediction. In [26], [27], Zhao *et al.* suggested CNN-based bi-directional motion compensation. Conventional bi-prediction averages two predictions unless additional weights are transmitted, while the proposed CNN combines two predictions adaptively, based on the content, and produces more accurate predicted blocks. Nonetheless, this method is only applicable to bi-directional prediction.

Prior work that is most relevant to the present study is [28], [36]. In [36], Niklaus *et al.* present a CNN that takes two frames as input and produces a set of 1D filters that can be used to synthesize an intermediate frame. The primary application in [36] is frame-rate up conversion (FRUC), but Zhao *et al.* [28] adopted that CNN (along with pre-trained weights provided by Niklaus *et al.*) for bi-directional prediction in HEVC-based video coding. Considerable coding gains are reported in [28] for high quantization parameter (QP) values. However, the CNN model developed in [36] and used in [28] cannot perform uni-directional prediction, so it cannot be used in low-delay coding scenarios.

In this paper, we extend the ideas presented in [36] to develop a single neural network model that is able to perform both uni- and bi-directional prediction. The proposed model is able to operate in all inter coding scenarios and provides a unified frame prediction tool for video coding.

III. Proposed frame prediction

The proposed method for frame prediction operates as follows. Two frames from the decoded picture buffer (DPB), along with their temporal index, are fed to the CNN. The CNN produces filter coefficients that are used to synthesize patches of a new frame. Once the frame is synthesized, it is used as a predictor for the current frame without any further motion information. Therefore, with this form of inter-prediction, no additional motion information needs to be coded, only the prediction residual and block-based flags indicating the use of the proposed method. At the decoder, the same procedure is performed to synthesize the predictor frame and then the residual is added to produce the final decoded frame.

A. Proposed network architecture

The proposed DNN architecture for frame prediction is shown in Fig. 1. It is inspired by the work of Niklaus *et al.* [36]. However, the network in [36] supports only bi-directional prediction, whereas our proposed DNN is able to support both uni- and bi-directional prediction. In video coding, it is important to be able to support both forms of prediction because, depending on the settings, some frames might not be available for prediction. This is true in low-delay delay cases, and more generally, in cases where the coding order does not produce an available decoded frame on both sides of the currently coded frame.

Unlike the network in [36], which has a single input path, the proposed DNN (Fig. 1) has two input paths which are then merged inside the network. The two inputs are fed with two patch tensors ($\widetilde{\mathbf{P}}_{t_1}$ and $\widetilde{\mathbf{P}}_{t_2}$) of size $N \times M \times 3$ (three color channels), from which prediction of an $N \times M \times 3$ patch in the current frame is made. For training the DNN we used $N = M = 128$, but in testing, different values are used. Since the DNN does not contain any fully-connected layers, the output scales with input size. Indices t_1 and t_2 (with $t_1 < t_2$) represent time index relative to the current frame index t. The tilde character (~) indicates that these patches come from previously coded and decoded frames. If needed, the patches are converted from YUV420 to YUV444 to make the resolution of all color channels the same, so that conventional convolutional layers can process the input. The final frame is converted back to YUV420.

In addition to the color channels, the input tensor contains an additional temporal index channel. Specifically, to the tensor $\widetilde{\mathbf{P}}_{t_i}, i \in \{1, 2\}$, we append $\mathbf{T}_{t_i} = c_i \cdot \mathbf{1}_{N \times M}$, where $\mathbf{1}_{N \times M}$ is a $N \times M$ matrix of all ones, and constants c_i are chosen as

$$(c_1, c_2) = \begin{cases} (-10, 10), & \text{if } t_1 < t < t_2, \\ (-20, -10), & \text{if } t_1 < t_2 < t. \end{cases} \tag{1}$$

The sign of c_i indicates whether the corresponding patch comes from a previous or subsequent frame, and its magnitude indicates the relative distance to the current frame. Using indices as tensor channels is inspired by the work in [37] where the authors have used spatial coordinates as additional tensor channels to achieve spatially-variant processing. Here, we use this concept for temporal indices of reference frames to enable temporally-variant processing that depends on the relative positions of the reference frames and the frame to be synthesized.

Comparing the proposed DNN in Fig. 1 with that in [36], we see that the network in [36] contains a single input path with both reference frames stacked as channels of the input tensor. This is appropriate for the case studied in [36], where the frame to be synthesized is always halfway between the two reference frames. In our case, however, the two reference frames may have different positions relative to the frame to be synthesized, so two input paths are used to allow different processing initially. Later, the paths are merged, and the inner portion of our DNN resembles U-Net [38], which is essentially the same as in [36]. One additional difference between our DNN and [36] are the skip connections from the merge point to each of the outputs, which do not exist in [36]. These skip connections offer further support to temporally-variant processing by allowing outputs to depend on the temporal index of the reference frames.

Processing within the network is accomplished via various processing blocks, which are indicated in Fig. 1. In the figure legend, the following abbreviations are used: “conv” stands for the convolutional layer (with 3×3 filters), “ReLU” stands for Rectified Linear Unit activation, and “bilinear” stands for bilinear upsampling. Input/output dimensions of various processing blocks are indicated in Table I.

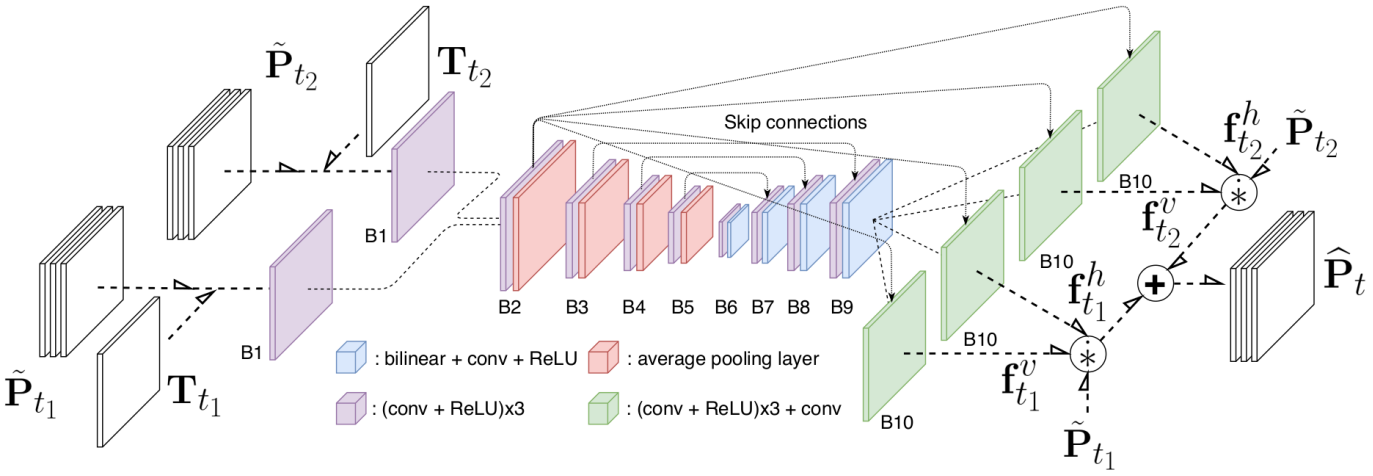


Fig. 1. Architecture of the proposed neural network, which is an extension of the one presented in [36]. Four-channel tensors derived from two reference patches and their temporal indices are applied at the two inputs. Processing is performed in ten blocks whose structure is indicated in the figure and whose input/output dimensions are shown in Table I. The outputs produce spatially-varying filters that are used to synthesize the predicted patch.

TABLE I
INPUT/OUTPUT DIMENSIONS OF VARIOUS BLOCKS (B1–B10) IN THE PROPOSED NETWORK

	B1 (two paths)	B2	B3	B4	B5	B6	B7	B8	B9	B10 (four paths)
Input	$N \times M \times 4$	$N \times M \times 32$	$\frac{N}{2} \times \frac{M}{2} \times 64$	$\frac{N}{4} \times \frac{M}{4} \times 128$	$\frac{N}{8} \times \frac{M}{8} \times 256$	$\frac{N}{16} \times \frac{N}{16} \times 512$	$\frac{N}{8} \times \frac{N}{8} \times 512$	$\frac{N}{4} \times \frac{M}{4} \times 256$	$\frac{N}{2} \times \frac{M}{2} \times 128$	$N \times M \times 64$
Output	$N \times M \times 16$	$\frac{N}{2} \times \frac{M}{2} \times 64$	$\frac{N}{4} \times \frac{M}{4} \times 128$	$\frac{N}{8} \times \frac{M}{8} \times 256$	$\frac{N}{16} \times \frac{N}{16} \times 512$	$\frac{N}{8} \times \frac{N}{8} \times 512$	$\frac{N}{4} \times \frac{M}{4} \times 256$	$\frac{N}{2} \times \frac{M}{2} \times 128$	$N \times M \times 64$	$N \times M \times C$

The network produces four output tensors: $\mathbf{F}^d_{t_i} \in \mathbb{R}^{N \times M \times C}$, where $i \in \{1,2\}, d \in \{h,v\}$. Each of these tensors contains 1-D filter coefficients for spatially-varying convolution, and C is the filter length (we used $C = 51$). For example, $\mathbf{F}^h_{t_i}$ contains horizontal (h) filters and $\mathbf{F}^v_{t_i}$ contains vertical (v) filters for processing $\widetilde{\mathbf{P}}_{t_i}$, $i \in \{1,2\}$. To apply spatially-varying convolution to pixel (x,y) in $\widetilde{\mathbf{P}}_{t_i}$, we extract (channel-wise) vectors from the corresponding locations in $\mathbf{F}^h_{t_i}$ and $\mathbf{F}^v_{t_i}$:

$$\mathbf{f}^h_{t_i} = \mathbf{F}^h_{t_i}(x,y,:), \qquad \mathbf{f}^v_{t_i} = \mathbf{F}^v_{t_i}(x,y,:), \tag{2}$$

and then perform the outer product to create a $C \times C$ 2-D filter kernel

$$\mathbf{K}^{(x,y)}_{t_i} = \mathbf{f}^v_{t_i}(\mathbf{f}^h_{t_i})^T. \tag{3}$$

Finally, pixel value at location (x,y) in color channel $c \in \{1,2,3\}$ in the predicted patch $\widehat{\mathbf{P}}_t$ is obtained as

$$\begin{aligned}\widehat{\mathbf{P}}_t(x,y,c) = &\sum \mathbf{K}^{(x,y)}_{t_1} \circ \widetilde{\mathbf{P}}^{(x,y)}_{t_1}(:,:,c) + \\ &\sum \mathbf{K}^{(x,y)}_{t_2} \circ \widetilde{\mathbf{P}}^{(x,y)}_{t_2}(:,:,c),\end{aligned} \tag{4}$$

where $\circ$ represents the Hadamard product (element-wise multiplication) of the 2-D kernel and a $C \times C$ region of the corresponding input patch centered at (x,y) in color channel c. The sum ($\sum$) represents summation of all elements in the Hadamard product. In [36], the operation of performing the Hadamard product and summing up the result is called "local convolution" and is denoted by symbol $\dot{*}$. We use the same symbol in Fig. 1. From (2)-(4) it is easy to see that "local convolution" is in fact spatially-varying convolution, because the kernels $\mathbf{K}^{(x,y)}_{t_i}$ depend on (x,y). When performing the Hadamard product in (4), samples are taken from the neighborhood of the patch in the corresponding frame, if needed. Near the frame boundary, nearest-neighbor padding is used to fill up the $C \times C$ matrix $\widetilde{\mathbf{P}}^{(x,y)}_{t_i}(:,:,c)$.

B. Loss function

The network is trained to predict the original patch $\mathbf{P}_t$. To accomplish this, a loss function with several terms is minimized. The first term is the Mean Squared Error (MSE) between the predicted patch $\widehat{\mathbf{P}}_t$ and the original patch $\mathbf{P}_t$:

$$\mathcal{L}_N = \frac{1}{N \cdot M \cdot 3} \, || \, \widehat{\mathbf{P}}_t - \mathbf{P}_t \, ||^2_F \tag{5}$$

The second loss term is borrowed from [36] and is based on the feature reconstruction loss introduced in [39]. According to [36], this term helps the model improve its prediction of details along the edges, which important for perceptual quality of the predicted patch. This loss term is defined as

$$\mathcal{L}_F = || \, \phi(\widehat{\mathbf{P}}_t) - \phi(\mathbf{P}_t) \, ||^2_2 \tag{6}$$

where $\phi(\cdot)$ is a feature extraction function. Like [36], we use the `relu4_4` layer of the VGG-19 network [40] as our feature extraction function.

The last part of the loss function is based on geometric features, specifically the horizontal and vertical gradients computed as the differences of neighboring pixels. The Mean

Absolute Difference (MAD) of the gradients in the predicted and the original patch form the corresponding loss terms:

$$\mathcal{L}_{G_x} = \frac{1}{N \cdot M \cdot 3} \sum \left| \frac{\partial}{\partial x} \widehat{\mathbf{P}}_t - \frac{\partial}{\partial x} \mathbf{P}_t \right| \\ \mathcal{L}_{G_y} = \frac{1}{N \cdot M \cdot 3} \sum \left| \frac{\partial}{\partial y} \widehat{\mathbf{P}}_t - \frac{\partial}{\partial y} \mathbf{P}_t \right| \tag{7}$$

and the summations are carried out over all three color channels.

Finally, the overall loss is the weighted combination of the loss terms introduced above:

$$\mathcal{L} = \lambda_N \cdot \mathcal{L}_N + \lambda_F \cdot \mathcal{L}_F + \lambda_G \cdot \left(\mathcal{L}_{G_x} + \mathcal{L}_{G_y} \right), \tag{8}$$

where we empirically selected $\lambda_N = \lambda_F = 2$ and $\lambda_G = 1$ for training. Further details of training will be discussed in Section V.

C. Conventional vs. proposed inter prediction

The proposed DNN is an alternative to conventional inter prediction – it also uses previously coded frames to predict the currently coded frame. Therefore, the proposed approach will compete with conventional prediction in the inter-coded frames in terms of rate distortion (RD) cost. To contrast the two approaches, we first recall the components of the RD cost of conventional inter prediction. Suppose a block $\mathbf{B}_t$ in the current frame t is to be inter-coded. Depending on whether uni- or bi-directional prediction is used, the motion-compensated prediction η has the form

$$\eta(\mathcal{T}, \Delta) = \begin{cases} \overline{\mathbf{B}}_{t_1}(x + \Delta x_{t_1}, y + \Delta y_{t_1}), & \text{uni-} \\ \alpha \cdot \overline{\mathbf{B}}_{t_1}(x + \Delta x_{t_1}, y + \Delta y_{t_1}) + & \\ (1 - \alpha) \cdot \overline{\mathbf{B}}_{t_2}(x + \Delta x_{t_2}, y + \Delta y_{t_2}), & \text{bi-} \end{cases} \tag{9}$$

where $\overline{\mathbf{B}}_{t_i}$ is the (possibly filtered) motion-compensated prediction from frame t_i, $(\Delta x_{t_i}, \Delta y_{t_i})$ is the corresponding motion vector (MV), and α is a weighting parameter, which is set to 0.5 by default, unless high-level syntax specifies another value [41]. $\mathcal{T}$ is the list of time indices of reference frames, which contains only t_1 for uni-directional prediction, otherwise both t_1 and t_2, and Δ is the list of corresponding MVs, containing one or two MVs, depending on whether prediction is uni- or bi-directional. Note that in bi-directional prediction, the two reference frames are on the opposite sides of the current frame, i.e., $t_1 < t < t_2$.

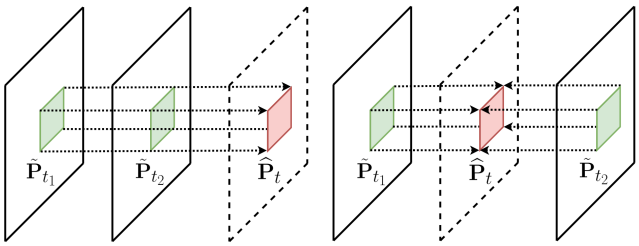
The distortion associated with inter prediction is conventionally taken as the Sum of Absolute Differences (SAD) between the current block and its prediction:

$$D_{\text{Inter}}(\mathcal{T}, \Delta) = \sum |\mathbf{B}_t - \eta(\mathcal{T}, \Delta)| \tag{10}$$

where the summation is over all pixels in the block. Finally, the RD cost of conventional inter prediction is given by

$$J_{\text{Inter}}(\mathcal{T}, \Delta) = D_{\text{Inter}}(\mathcal{T}, \Delta) + \lambda \cdot R(\mathcal{T}, \Delta) \tag{11}$$

where R denotes a rate function that measures the number of bits required to encode residuals and other necessary parameters, such as MVs, and λ is the Lagrage multiplier chosen according to the quantization parameter (QP). RD optimization finds the parameters $(\mathcal{T}, \Delta)$ that lead to the minimum RD cost. According to [42], the number of bits needed to represent the optimal inter-prediction parameters usually accounts for about 10-30% of the total bits spent on an inter frame.



(a) Uni-direction (b) Bi-direction

Fig. 2. Uni- and bi-directional prediction in the proposed method.

On the other hand, the proposed inter prediction operates as shown in Fig. 2. For uni-directional prediction, co-located patches from two preceding frames are taken, and for bi-directional prediction, co-located patches from opposite sides of the current frame are used. Since the patches are co-located, no motion vectors need to be transmitted. Motion information, as well as any possible weighting of contributions from the two reference frames, is effectively encoded in the spatially-varying kernels $\mathbf{K}_{t_i}^{(x,y)}$, which are derived by the DNN from the reference frames, at both encoder and decoder. Apart from the residual, only the flag indicating that the proposed inter prediction is used and the temporal indices of the reference frames $\mathcal{T} = (t_1, t_2)$ need to be transmitted.

The distortion associated with the proposed inter prediction is denoted D_{DNN} and computed as:

$$D_{\text{DNN}}(\mathcal{T}) = \sum \left| \mathbf{P}_t - \widehat{\mathbf{P}}_t \right| \tag{12}$$

where $\widehat{\mathbf{P}}_t$ is computed as in (4) and the sum is over all pixels in the patch. Compared with conventional inter prediction (9), the proposed method always exploits two reference patches even for the uni-prediction. The RD cost for the proposed method is denoted J_{DNN} and is computed as

$$J_{\text{DNN}}(\mathcal{T}) = D_{\text{DNN}}(\mathcal{T}) + \lambda \cdot R(\mathcal{T}) \tag{13}$$

where the rate function measures the number of bits needed to encode the residual and the flag indicating that the proposed method is used.

In preliminary testing, we found (unsurprisingly) that bi-directional prediction usually gives better results than uni-directional prediction. So, in looking for reference frames, the priority is given to bi-directional prediction, and previously coded frames closest to the current frame on both sides are selected as references (up to distance 2) if they are available, given the hierarchical-B coding order [43]. If bi-directional prediction is not possible due to unavailability of future coded frames up to distance 2, the two closest previously coded frames preceding the current frame are selected as references.

Finally, RD opimization selects the smallest RD cost J^* among the intra, inter, and the proposed DNN-based prediction, as the best coding mode for the current block:

$$J^* = \min(J_{\text{Intra}}, J_{\text{Inter}}, J_{\text{DNN}}). \quad (14)$$

Note that the blocks coded by the proposed method do not have any MVs associated with them. While this saves bits compared to conventional inter coding, it may inadvertently increase the bits used for other, conventionally inter-coded blocks in the same frame, because their MVs may have fewer neighboring MVs from which MV prediction can be made. In order to mitigate this, we examined using motion estimation (at both encoder and decoder, without sending any extra information) for DNN-predicted blocks in order to assign MVs to them for the purpose of MV prediction of other, conventionally coded MVs. However, this did not lead to any rate saving, which leads us to conclude that MVs estimated for DNN-predicted blocks were rarely, if ever, selected as MV predictors. Hence, we abandoned this idea, and the final codec did not incorporate this feature.

IV. DNN TRAINING

In this section, we detail the process of our DNN training. Since the performance of the learned network directly affects the coding gain of the proposed frame prediction, choosing the effective learning strategy is very important. Some of the training choices, such as the optimizer and learning rate, are based on the exploration by Niklaus *et al.* [36]. Accordingly, we employed AdaMax [44] with default parameter values, and started training from scratch with the learning rate of 0.001. We also set the mini-batch size to 16. However, there are also some key differences relative to [36]. Since the focus application of our DNN is video coding, we used YUV sequences, rather than RGB. Also, our training samples are collected from [45], which contains diverse raw sequences at various frame rates. We found raw video sequences with 25, 30, 50, and 60 frames per second (fps) and various resolutions from 352×240 (SIF) to 1920×1080 (FullHD). Considering the diversity of resolutions and frame rates in our data, we chose a two-part training strategy consisting of pre-training and fine tuning.

A. Pre-training

The purpose of pre-training is to find, on a relatively small dataset, reasonably good network weights from which large-scale fine tuning can start. For pre-training, we employ a total of 27 sequences: 25 for training and another 2 for validation. Resolutions of these 27 sequences were either 352×240 (SIF) or 352×288 (CIF). In order to train a model that is able to operate at various QP values, we adopted compression augmentation [46]. First, the chosen sequences were encoded at various QPs: 20, 22, 24, ..., 44, for a total of 13 QP values. For each QP value, three different coding configurations were used: Low delay, Low delay P, and Random access. During training, either a raw sequence or one of the three coding configurations was chosen randomly, and if the choice was a coded sequence, then one QP was selected randomly from the 13 QP values available.

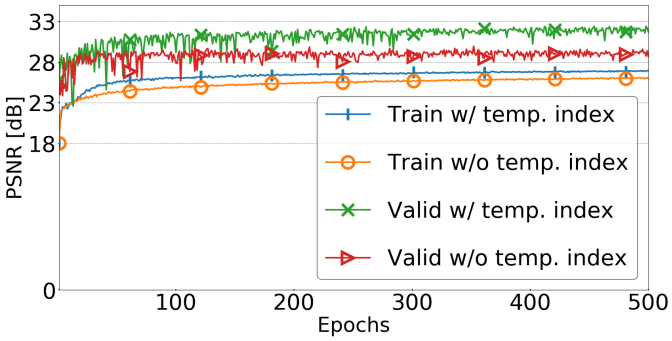


Fig. 3. The effect of adding the temporal index channel to the DNN input.

A training sample consists of three collocated patches of size 128×128 randomly chosen from a triplet of frames within a given sequence. One epoch covers a total of 6,158 training frame triplets and 698 validation triplets. In addition to compression augmentation, which was done offline, we used several forms of online augmentation, namely horizontal and vertical flipping, and reversing the frame order, so that the network gets exposed to a variety of motions. The network was trained for 500 epochs on NVIDIA GeForce GTX 1080 GPU with 11 GB RAM.

In order to demonstrate the impact of the temporal index channel $\mathbf{T}_{t_i}$, we trained another DNN with the same architecture and on the same data, but with $\mathbf{T}_{t_i}$ set to a dummy constant, same for both input patches. Fig. 3 shows the results in the form of predicted patch Peak Signal to Noise Ratio (PSNR) vs. epoch, on both training and validation data. As seen in the results, correct temporal index results in predicted patches having 1–2 dB higher PSNR.

B. Fine tuning

Fine tuning starts from the network weights obtained in pre-training. In this stage, a lager dataset and a more sophisticated training strategy was used. First, patch triplets were drawn randomly from the training sequences with resolutions ranging from SIF to FullHD. Following [36], patches of size 150×150 were initially drawn, and from these patches, collocated windows of 128×128 were cropped. Motion augmentation [36] in the form of shifting the reference windows was used to increase the diversity of motions seen by the network. To further enrich the diversity of training samples, we eliminated "monotonous" triplets – those whose entropy in any patch was less than 3.5 bits per pixel, and those that did not exhibit any pixel value change between any pair of patches. We also estimated optical flow between the (temporally) first and last patch in the triplet using the Coarse2Fine method [47] and eliminated patch triplets with zero variance in optical flow magnitudes. We also excluded triplets with extreme motion, where the largest optical flow magnitude was larger than the patch diagonal. Finally, the training set was formed with 27,360 triplets, and the validation set had 3,040 triplets. Training was done for 500 epochs starting from the weights

TABLE II
EXPERIMENTAL ENVIRONMENT

Item	Specification
Processor	Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz
GPU	GeForce GTX 1080 with 11 GB RAM
Operating system	Ubuntu-16.04
HEVC version	HM-16.20
DNN framework	Pytorch-gpu-0.4.1-py36 with CUDA 9.0

TABLE III
AVERAGE Y-PSNR OF PREDICTED FRAMES

Sequence	fps	Uni-directional (dB)			Bi-directional (dB)		
		[36]	Sep.	Comb.	[36]	Sep.	Comb.
BQSquare	60	22.62	**33.84**	33.46	**37.19**	36.59	36.89
BasketballPass	50	21.15	**30.40**	30.25	31.57	**33.21**	32.80
RaceHorses	30	18.96	**27.43**	27.28	29.35	**30.20**	29.99
ParkScene	24	26.55	**36.01**	35.79	36.74	**38.26**	38.22

obtained in pre-training, and using the same data augmentation strategy as in pre-training.

In addition to the DNN that supports both uni-and bi-directional prediction, we trained two other DNNs with the same architecture: one specifically for uni-directional prediction, the other specifically for bi-directional prediction. The same data and the same strategy – pre-training followed by fine tuning – was used for these two DNNs as well. These two DNNs will be used in the next section for comparison purposes.

V. EXPERIMENTAL RESULTS

The setup used for experimental evaluation of the proposed approach is shown in Table II. Two groups of experiments were carried out: evaluation of frame prediction performance in Section V-A and evaluation of video coding performance in Section V-B.

A. Frame prediction performance

We compare the frame prediction performance of our three networks – one for combined uni-/bi-directional prediction and two for separate uni- and bi-directional prediction – with the reference network [36], which was also used in [28]. The comparison is carried out on the four HEVC test sequences [48] listed in Table III. Sequences that were larger than 416×240 (≈WQVGA) were cropped down to this resolution. For uni-directional prediction, the temporal indices of reference frames were $(t_1, t_2) = (t-2, t-1)$ and for the bi-directional case they were $(t_1, t_2) = (t-1, t+1)$.

Table III presents the average PSNR of the Y-component of predicted frames for each model. Our DNN that performs combined uni-/bi-directional prediction is denoted "Comb." in the table, while the DNNs for separate uni- and bi-directional prediction are denoted "Sep." The uni-directional performance of [36] is quite low, understandably, because the network was not trained for this case. But our models outperform [36] in bi-directional prediction in three out of four sequences. The only exception is BQSquare, with very low motion, where all models do fairly well. Among our models, while separately trained models are clearly better than the combined model, the difference is not large, less than 0.5 dB. Most importantly, the combined model outperforms [36] in most bi-directional cases, even though it can cupport both uni- and bi-directional prediction, whereas [36] supports only bi-directional prediction.

Several visual examples of bi-direction prediction are shown in Fig. 4. The BQSquare sequence in the first row is characterized by camera panning at a distance, with high frame rate. Thus the difference between consecutive frames is very small, and the the predicted frame is fairly accurate (around 37 dB) for all models. Only subtle differences can be found in the zoomed-in face region shown in the bottom right of each frame. In the second row, the BasketballPass sequence has much more complicated motion, and the differences in predicted frames are easily noticeable. The basketball is poorly reconstructed by the network from [36] in the second column, while our separate model in the third column and the combined model in the last column do a much better job. Of the three predictions, the separate model in the third column gives the best result. With the RaceHorses sequence in the third row, all models struggle to reconstruct the horse's leg, but our separate model again provides the best result. Finally, with the ParkScene sequence in the last row, the proposed models again provide better prediction than [36], which suffers from the noticeable warp on the pillar along the the arm.

For bi-directional prediction to be used in hierarchical B coding in the Random access configuration, prediction is sometimes required from more distant frames, not just the immediately preceding and succeeding frame. Therefore, in Fig. 5, we evaluate bi-directional prediction accuracy from a previous and a future frame at various (symmetric) distances from the current frame. The figure shows average Y-PSNR (in dB) of the predicted frame vs. frame distance, averaged over all frames in the corresponding sequences where such bi-directional prediction is possible. The proposed DNNs, both the separately trained bi-directional DNN and the combined uni-/bi-directional DNN, outperform the network from [36] in all cases except BQSquare at frame distance of 1. At the same time, except in BQSquare at frame distance of 1, our separately trained bi-directional DNN is slightly better than our combined uni-/bi-directional DNN, most notably in ParkScene at larger frame distances.

B. Video coding performance

In this section, we evaluate the performance of the proposed frame prediction in coding units in video coding. The benchmark is the latest video coding standard, High Efficiency Video Coding (HEVC) [41]. The proposed method competes with HEVC inter and intra prediction in inter frames in terms of the RD cost (14), where the proposed method is selected when $J_{\text{DNN}} < J_{\text{Inter}}, J_{\text{Intra}}$. As shown in Table II, the proposed prediction method is implemented based on HM-16.20 and the DNN is implemented in Pytorch. Python Embedding Library[1]

[1]https://docs.python.org/3/extending/

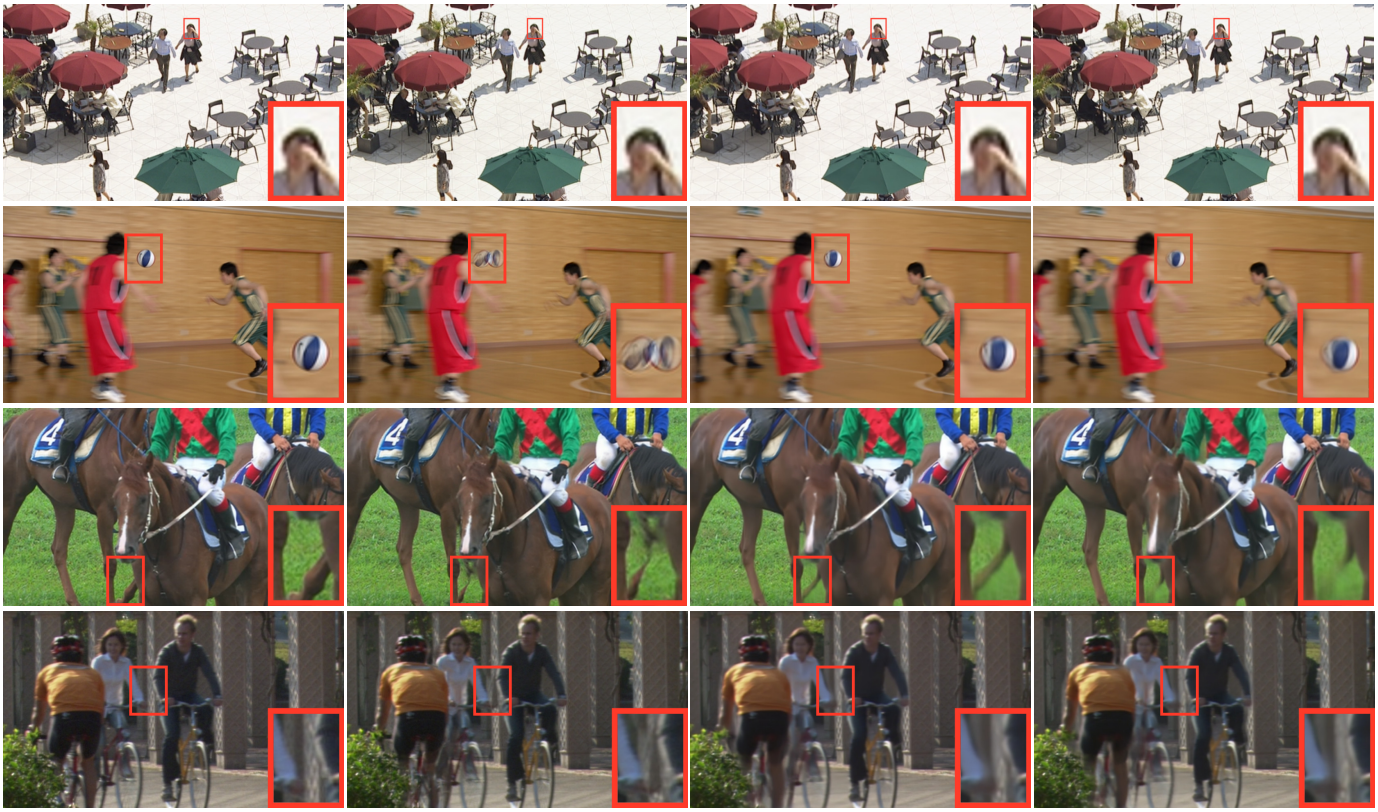


Fig. 4. Visual comparison of bi-directionally predicted frames: (a) original, (b) frame predicted by [36], (c) frame predicted by our separately trained bi-directional DNN, (d) frame predicted by our combined uni-/bi-directional DNN.

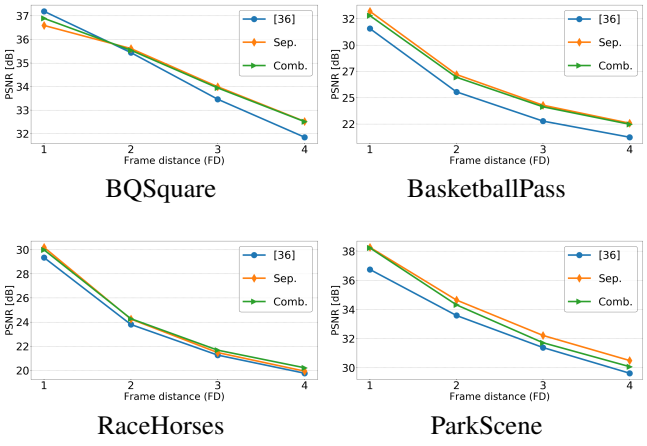


Fig. 5. Bi-directional prediction with varying frame distance.

is used to embed the DNN into HM-16.20. During video coding, forward operation of the DNN to perform frame prediction is executed on the GPU (Table II). However, due to the memory limitations of the GPU, it is impossible to generate the prediction frame larger than 832×480 (≈WVGA) resolution. Therefore, larger frames are split into multiple tiles with a maximum size of WVGA. Then tile-wise prediction ins performed and the predicted frame is assembled from predicted tiles.

For evaluation, three coding configurations (Low delay, Low delay P, and Random access) that allow inter-frame coding under the common test conditions (CTC) with the common test sequences (CTS) [48] are employed with four quantization parameters QP $\in \{22, 27, 32, 37\}$. In the Low delay and Low delay P configurations, only uni-directional prediction is available using two previously coded frames before updating the reference picture set (RPS) [49] of the current frame. For the Random access configuration, bi-directional prediction is performed when the coded frames with a maximum difference of ± 2 from the POC of the current frame are available before updating the RPS. Larger distances can be allowed, but the computational complexity increases while prediction performance reduces at larger distances, so we felt the POC difference of up to 2 is a good compromise.

Table IV shows the coding performance compared to the HEVC for various configurations. We measure the coding performance using BD-Bitrate [50]. We show the performance of both the separate uni- and bi-directional models ("Sep." in the table), as well as the combined uni-/bi-directional model ("Comb." in the table). The proposed method reduces the bitrate for the luminance component in all test cases. There is some increase for the chrominance components of a couple of sequences, but considering that the chrominance components are much smaller than luminance, this increase is negligible compared to the overall savings. And on average, the bitrate of chrominance components across all sequences is reduced. For the Low delay P configuration, the proposed method achieves

the largest bits savings of up to 9.7% and 7.9% with the separate model and the combined model, respectively, for the sequence FourPeople. In general, significant bit reductions are shown in larger resolution sequences in Classes A, B and E. Since the proposed method utilizes two coded frames using adaptive filters derived by the DNN, it seems to compensate the case when the predictors with P frame coding have limited accuracy because of a single reference frame.

Compared with the Low delay P configuration, the coding gains are somewhat lower in the Low delay configuration, especially in Class B sequences. However, gains in Class E (teleconference) sequences are still strong. Similarly, in the Random access configuration, the coding gains are somewhat less than in Low delay P, but comparable to the coding gains in the Low delay configuration. The largest luminance coding gain in this case is 4.4%, achieved on the sequence PeopleOnStreet. Moreover, significant bit savings were shown in Class C and D. Overall, the combined uni-/bi-directional DNN offers somewhat lower gains than separate uni- and bi-directional DNNs by about 0.4%-0.8%, but it supports both forms of prediction so it allows a simpler video coding system design.

The last two rows of the table show the difference in encoding and decoding time, defined as

$$\Delta T_p = \frac{T_{Proposed,p}}{T_{HM,p}} \times 100\% \tag{15}$$

where $p \in \{Enc, Dec\}$, and $T_{Proposed,p}$ and $T_{HM,p}$ represent the the total elapsed time for the proposed method and the original HM-16.20, respectively. Due to the complexity of running a deep model, the encoding run time increases by 45-65%, depending on the configuration. However, the decoding time increases much more significantly. This is because decoding time is small to start with, and the complexity of frame prediction with the proposed method is symmetric (i.e., it takes the same amount of time) at the encoder and the decoder, so compared to an initially small decoding time, running a deep model adds significant complexity. As with other envisioned applications of deep models, run-time complexity is one of the bottlenecks, and clever solutions will need to be found to get this technology into end-user products.

In Fig. 6 we show the percentage area of the frame where a particular coding mode is selected. Fig. 6(a) shows the case of the FourPeople sequence in the Low delay configuration, while Fig. 6(b) shows the case of BQMall in the Random access configuration. For each QP value, three bars are shown: one for HM-16.20, one for the separate uni- and bi-directional DNNs ("Sep."), and one for the joint uni-/bi-directional DNN ("Comb."). From the graphs, we see that the proposed frame prediction takes over from inter and skip modes in both cases, but more so in the Random access configuration in Fig. 6(b). As expected, separate uni- and bi-directional DNNs are better at frame prediction than the combined uni-/bi-prediction DNN, but even the combined model provides the best coding mode for up to a third of the frame at higher QP values Fig. 6(b). In the FourPeople sequence in Fig. 6(a), large static background causes skip mode to become more and more efficient as QP increases, but in the BQMall sequence (Fig. 6(b)) with camera motion, our predictors tend to win over larger area as the QP increases.

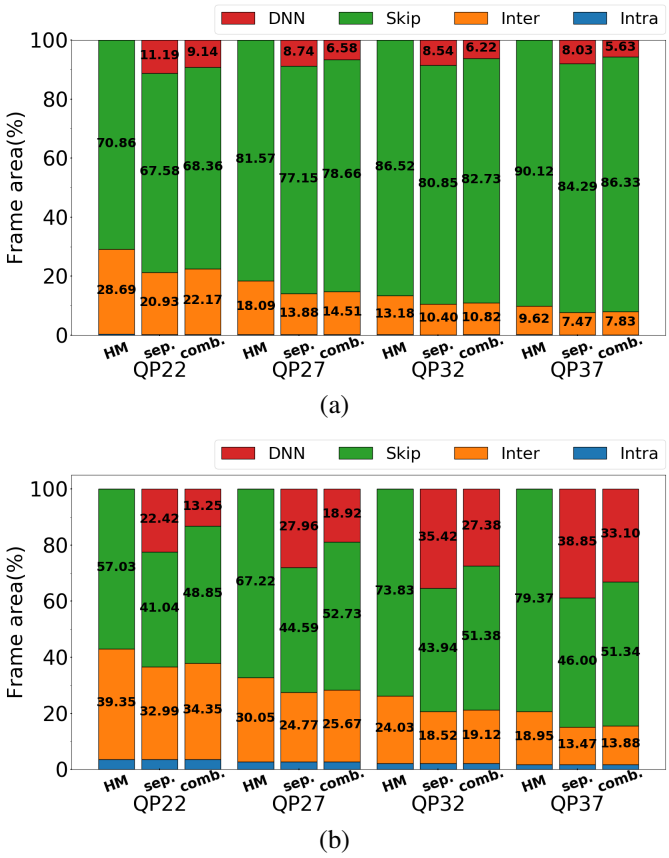


Fig. 6. Selected coding modes for various QP values. (a) FourPeople in Low delay configuration; (b) BQMall in Random access configuration.

Finally, we compare the coding efficiency of the proposed approach to that of Lei *et al.* [28]. Since [28] directly uses the network from [36], it only supports bi-directional prediction. Hence, we could only carry out this comparison on the Random access configuration. Further, [28] is implemented in HM-16.6, so we also re-implemented our method in HM-16.6, and evaluated the coding performance on the same experimental setup as in [28], on sequences in Class B, C and D for 2 seconds with QP $\in \{27, 32, 37, 42\}$. Table V shows BD-Bitrate of each method against HM-16.6. Overall, our separately trained DNN model for bi-directional prediction ("Sep.") is the best overall, with the highest average reduction in bitrate (3.5%) and providing the best performance in 9 out of 13 sequences in this test. Lei *et al.* [28] results are the second best overall, with the average bit rate reduction of 3.2% and top performance in 4 out of 13 sequences, while our combined uni-/bi-directional DNN ("Comb.") comes in third with a slightly lower overall bit rate reduction of 3.1%. However, even our combined DNN provides better coding gain than [28] in 8 out of 13 sequences. Furthermore, despite the fact that DNN-based uni-directional prediction was not used in this test, it is worth remembering that our combined DNN is the only one of the three models that supports both uni- and bi-directional prediction, and still is competitive with the other two models.

TABLE IV
BD-BITRATE RELATIVE TO HM-16.20 OVER THREE COMMON TEST CONDITIONS

Class	Sequence	fps	Low delay P						Low delay						Random access					
			Sep. (%)			Comb. (%)			Sep. (%)			Comb. (%)			Sep. (%)			Comb. (%)		
			Y	U	V	Y	U	V	Y	U	V	Y	U	V	Y	U	V	Y	U	V
A	PeopleOnStreet	30	–5.8	–4.8	–1.6	–4.9	–2.0	–2.6	–4.3	–3.0	–0.1	–3.4	–0.4	–1.0	–4.4	–5.4	–5.4	–3.7	–4.8	–5.5
	Traffic		–3.9	–4.7	–3.7	–3.1	–3.5	–1.5	–2.6	–3.2	–3.1	–1.8	–2.6	–1.4	–2.4	–2.7	–2.2	–2.0	–2.3	–1.9
B	BQTerrace	60	–4.4	–3.3	–0.9	–3.3	0.7	9.3	–1.3	–1.2	–1.0	–1.0	–0.4	2.7	–1.7	0.0	0.3	–1.5	0.3	0.5
	BasketballDrive	50	–2.8	–5.9	–4.0	–2.8	–4.8	–3.0	–1.1	–2.7	–1.9	–0.9	–2.1	–1.1	–1.2	–2.0	–1.4	–1.1	–1.4	–0.9
	Cactus		–6.6	–11.4	–7.1	–5.7	–8.9	–3.2	–3.5	–7.9	–4.9	–2.8	–6.3	–2.9	–3.2	–5.5	–3.3	–2.7	–5.0	–2.3
	Kimono	24	–5.6	–8.8	–4.1	–5.1	–7.1	–2.1	–2.2	–5.2	–2.9	–1.7	–4.0	–1.7	–1.1	–1.9	–1.0	–1.0	–1.2	–0.5
	ParkScene		–3.5	–5.9	–2.9	–3.0	–3.4	–1.2	–2.2	–4.3	–2.2	–1.8	–2.0	–0.8	–1.8	–2.3	–1.6	–1.7	–1.6	–1.1
C	BQMall	60	–6.1	–9.8	–8.0	–5.0	–6.5	–4.3	–4.5	–7.8	–6.4	–3.6	–5.7	–4.0	–4.3	–5.3	–4.9	–3.5	–3.7	–3.3
	BasketballDrill	50	–3.1	–5.5	–3.5	–2.7	–5.6	–3.2	–1.6	–3.2	–2.0	–1.4	–3.2	–1.7	–1.7	–3.1	–2.8	–1.4	–2.2	–2.0
	PartyScene		–2.8	–4.2	–4.6	–2.6	–3.4	–1.4	–1.8	–3.2	–3.2	–1.6	–2.7	–1.5	–3.9	–4.1	–3.5	–3.5	–2.7	–2.2
	RaceHorsesC	30	–1.2	–1.8	–2.6	–1.0	–1.2	–1.7	–0.8	–1.0	–1.7	–0.7	–0.8	–0.9	–0.7	–1.2	–1.5	–0.5	–0.8	–0.9
D	BQSquare	60	–2.4	–3.3	–1.0	–1.8	2.4	4.4	–1.3	–2.8	–2.1	–0.9	–0.3	0.7	–2.4	–0.3	0.0	–2.6	–0.1	0.3
	BasketballPass	50	–4.6	–6.9	–5.6	–3.9	–5.3	–3.2	–3.4	–5.0	–4.3	–2.7	–3.7	–2.0	–4.1	–5.5	–4.2	–3.4	–4.0	–2.7
	BlowingBubbles		–3.8	–5.4	–5.0	–2.9	–4.2	–3.9	–2.5	–4.3	–4.2	–1.8	–3.7	–3.5	–3.9	–3.9	–3.5	–3.3	–2.7	–2.3
	RaceHorses	30	–1.9	–2.7	–3.3	–1.5	–2.2	–2.2	–1.4	–2.1	–2.9	–1.0	–1.2	–2.1	–1.6	–2.7	–2.6	–1.4	–2.0	–2.1
E	FourPeople	60	–9.7	–12.6	–12.2	–7.9	–7.0	–4.9	–6.8	–10.1	–9.9	–5.2	–5.7	–4.2		-			-	
	Johnny		–8.0	–6.4	–7.3	–6.4	–2.1	–1.8	–4.5	–5.9	–4.8	–3.0	–2.7	–4.3		-			-	
	KristenAndSara		–8.6	–10.6	–7.9	–6.8	–6.0	–3.5	–5.4	–8.1	–5.8	–3.6	–3.4	–2.3		-			-	
	Average		–4.7	–6.3	–4.7	–3.9	–3.7	–1.7	–2.8	–4.5	–3.5	–2.2	–2.8	–1.8	–2.6	–3.1	–2.5	–2.2	–2.3	–1.8
	ΔT_{Enc}		163%			164%			146%			145%			150%			149%		
	ΔT_{Dec}		16,563%			16,562%			15,415%			15,410%			11,389%			11,488%		

TABLE V
BD-BITRATE RELATIVE TO HM-16.6 WITH THE RANDOM ACCESS CONFIGURATION

Class	Sequence	fps	Lei *et al.* [28]		Proposed			
					Sep.		Comb.	
			Y (%)	Avg. (%)	Y (%)	Avg. (%)	Y (%)	Avg. (%)
B	BQTerrace	60	–0.2	–2.0	**–0.8**	–2.0	–0.6	–1.9
	BasketballDrive	50	–1.1		**–1.5**		–1.4	
	Cactus		**–4.6**		–3.7		–3.2	
	Kimono	24	**–1.7**		–1.0		–1.2	
	ParkScene		–2.6		**–3.0**		–2.9	
C	BQMall	60	–6.0	–3.2	**–7.1**	–3.9	–6.3	–3.4
	BasketballDrill	50	**–3.2**		–2.7		–2.3	
	PartyScene		–3.0		**–4.2**		–3.7	
	RaceHorsesC	30	–0.8		**–1.4**		–1.2	
D	BQSquare	60	**–7.1**	–4.7	–4.0	–4.5	–4.0	–4.1
	BasketballPass	50	–5.4		**–6.6**		–5.8	
	BlowingBubbles		–4.1		**–4.2**		–3.7	
	RaceHorses	30	–2.2		**–3.2**		–2.8	
	Average		–3.2		**–3.5**		–3.1	

VI. CONCLUSION

We presented a deep neural network (DNN) for frame prediction that can be used to improve video coding efficiency. The DNN operates at both encoder and decoder, and uses previously decoded frames to predict the current frame. This form of prediction is signalled as a separate prediction mode, and can be used within RD optimization to compete with other prediction modes. Although it is a form of inter prediction, it does not require any motion vectors to be transmitted. Three DNNs were trained for this purpose: two for separate uni- and bi-directional prediction, and one that supports both uni- and bi-directional prediction, the first of its kind, to our knowledge. The DNNs were evaluated on common test sequences and various coding configurations, and demonstrated to bring significant coding gains relative to HEVC. Furthermore, advantages over the recent work on DNN-based frame prediction for video coding was also demonstrated.

REFERENCES

[1] "Cisco visual networking index: Forecast and methodology, 2016-2021," https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/, Accessed: 2018-10-22.

[2] T. Huang, "Surveillance video: The biggest big data," *Computing Now*, vol. 7, no. 2, pp. 82–91, 2014.

[3] V. Barocini, J.-R. Ohm, and G.J. Sullivan, "Report of results from the call for proposals on video compression with capability beyond HEVC," in *ISO/IEC JTC1/SC29 WG11, JVET-J1003*, Sep. 2018.

[4] S. Liu, B. Choi, K. Kawamura, Y. Li, L. Wang, P. Wu, and H. Yang, "JVET AHG report: neural networks in video coding," in *ISO/IEC JTC1/SC29 WG11, JVET-L0009*, Oct. 2018.

[5] R. Yang, M. Xu, and Z. Wang, "Decoder-side hevc quality enhancement with scalable convolutional neural network," in *Proc. IEEE ICME'17*. IEEE, 2017.

[6] Z. Jin, P. An, C. Yang, and L. Shen, "Quality enhancement for intra frame coding via cnns: An adversarial approach," in *Proc. IEEE ICASSP'18*. IEEE, 2018.

[7] C. Li, L. Song, R. Xie, and W. Zhang, "Cnn based post-processing to improve hevc," in *Proc. IEEE ICIP'17*. IEEE, 2017.

[8] X. He, Q. Hu, X. Zhang, C. Zhang, W. Lin, and X. Han, "Enhancing HEVC compressed videos with a partition-masked convolutional neural network," in *Proc. IEEE ICIP'18*. IEEE, 2018.

[9] J. Kang, S. Kim, and K. M. Lee, "Multi-modal/multi-scale convolutional neural network based in-loop filter design for next generation video codec," in *Proc. IEEE ICIP'17*. IEEE, 2017.

[10] C. Jia, S. Wang, X. Zhang, S. Wang, and S. Ma, "Spatial-temporal residue network based in-loop filter for video coding," in *Proc. IEEE VCIP'17*. IEEE, 2017.

[11] L. Zhou, X. Song, J. Yao, L. Wang, and F. Chen, "Convolutional neural network filter for inra frame," in *ISO/IEC JTC1/SC29 WG11, JVET-I0022*, Jan. 2018.

[12] J. Yao, X. Song, S. Fang, and L. Wang, "AHG9: convolutional neural network filter for inter frame," in *ISO/IEC JTC1/SC29 WG11, JVET-J0043*, Apr. 2018.

[13] T. Hashimoto and E. Sasaki T. Ikai, "AHG9: Separable convolutional neural network filter with squeeze-and-excitation block," in *ISO/IEC JTC1/SC29 WG11, JVET-K0158*, Jul. 2018.

[14] Y.-L. Hsiao, C.-Y. Chen, T.-D. Chuang, C.-W. Hsu, Y.-W Huang, and S.-M Lei, "AHG9: Convolution neural network loop filter," in *ISO/IEC JTC1/SC29 WG11, JVET-K0222*, Jul. 2018.

[15] Y. Wang, Z. Chen, and Y. Li, "AHG9: Dense residual convolutional neural network based in-loop filter," in *ISO/IEC JTC1/SC29 WG11, JVET-K0391*, Jul. 2018.

[16] Y. Wang, Z. Chen, and Y. Li, "AHG9: Dense residual convolutional neural network based in-loop filter," in *ISO/IEC JTC1/SC29 WG11, JVET-L0242*, Oct. 2018.

[17] K. Kawamura, Y. Kidani, and S. Naito, "AHG9: Convolution neural network filter," in *ISO/IEC JTC1/SC29 WG11, JVET-L0383*, Oct. 2018.

[18] W. Cui, T. Zhang, S. Zhang, F. Jiang, W. Zuo, Z. Wan, and D. Zhao, "Convolutional neural networks based intra prediction for HEVC," in *Proc. IEEE DCC'17*. IEEE, 2017.

[19] Jiahao Li, Bin Li, Jizheng Xu, and Ruiqin Xiong, "Intra prediction using fully connected network for video coding," in *Proc. IEEE ICIP'17*. IEEE, 2017.

[20] Y. Hu, W. Yang, S. Xia, W.-H. Cheng, and J. Liu, "Enhanced intra prediction with recurrent neural network in video coding," in *Proc. IEEE DCC'17*. IEEE, 2018, pp. 413–413.

[21] Y. Hu, W. Yang, M. Li, and J. Liu, "Progressive spatial recurrent neural network for intra prediction," *arXiv preprint arXiv:1807.02232*, 2018.

[22] J. Pfaff, P. Helle, D. Maniry, S. Kaltenstadler, B. Stallenberger, P. Merkle, M. Siekmann, H. Schwarz, D. Marpe, and T. Wiegand, "Intra prediction modes based on neural networks," in *ISO/IEC JTC1/SC29 WG11, JVET-J0037*, Apr. 2018.

[23] H. Zhang, L. Song, Z. Luo, and X. Yang, "Learning a convolutional neural network for fractional interpolation in HEVC inter coding," in *Proc. IEEE VCIP'17*. IEEE, 2017.

[24] N. Yan, D. Liu, H. Li, T. Xu, F. Wu, and B. Li, "Convolutional neural network-based invertible half-pixel interpolation filter for video coding," in *Proc. IEEE ICIP'18*. IEEE, 2018.

[25] S. Huo, D. Liu, F. Wu, and H. Li, "Convolutional neural network-based motion compensation refinement for video coding," in *Proc. IEEE ISCAS'18*. IEEE, 2018.

[26] Z. Zhao, S. Wang, S. Wang, X. Zhang, S. Ma, and J. Yang, "CNN-based bi-directional motion compensation for high efficiency video coding," in *Proc. IEEE ISCAS'18*. IEEE, 2018.

[27] Z. Zhao, S. Wang, S. Wang, X. Zhang, S. Ma, and J. Yang, "Enhanced bi-prediction with convolutional neural network for high efficiency video coding," *IEEE Trans. Circuits Syst. Video Technol.*, 2018.

[28] L. Zhao, S. Wang, X. Zhang, S. Wang, S. Ma, and W. Gao, "Enhanced CTU-level inter prediction with deep frame rate up-conversion for high efficiency video coding," in *Proc. IEEE ICIP'18*, 2018.

[29] Z. Jin, P. An, L. Shen, and C. Yang, "CNN oriented fast QTBT partition algorithm for JVET intra coding," in *Proc. IEEE VCIP'17*. IEEE, 2017.

[30] M. Xu, T. Li, Z. Wang, X. Deng, R. Yang, and Z. Guan, "Reducing complexity of hevc: A deep learning approach," *IEEE Transactions on Image Processing*, 2018.

[31] Z. Wang, S. Wang, X. Zhang, S. Wang, and S. Ma, "Fast qtbt partitioning decision for interframe coding with convolution neural network," in *Proc. IEEE ICIP'18*. IEEE, 2018.

[32] B. Xu, X. Pan, Y. Zhou, Y. Li, D. Yang, and Z. Chen, "Cnn-based rate-distortion modeling for h. 265/hevc," in *Proc. IEEE VCIP'17*. IEEE, 2017.

[33] T. Chen, H. Liu, Q. Shen, T. Yue, X. Cao, and Z. Ma, "Deepcoder: A deep neural network based video compression," in *Proc. IEEE VCIP'17*. IEEE, 2017.

[34] K. Ugur, A. Alshin, E. Alshina, F. Bossen, W.-J. Han, J.-H. Park, and J. Lainema, "Motion compensated prediction and interpolation filter design in H.265/HEVC," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 6, pp. 946–956, 2013.

[35] J. Kim, J. Lee, and K. Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proc. IEEE CVPR'16*, 2016, pp. 1646–1654.

[36] S. Niklaus, L. Mai, and F. Liu, "Video frame interpolation via adaptive separable convolution," in *Proc. IEEE ICCV'17*, 2017.

[37] R. Liu, J. Lehman, P. Molino, F. P. Such, E. Frank, A. Sergeev, and J. Yosinski, "An intriguing failing of convolutional neural networks and the coordconv solution," *arXiv preprint arXiv:1807.03247*, 2018.

[38] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.

[39] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. ECCV*, 2016, pp. 694–711.

[40] A. Zisserman K. Simonyan, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[41] Int. Telecommun. Union-Telecommun. (ITU-T) and Int. Standards Org./Int/Electrotech. Commun. (ISO/IEC JTC 1), "High efficiency video coding," Rec. ITU-T H.265 and ISO/IEC 23008-2, 2013.

[42] J. Stankowski, D. Karwowski, T. Grajek, K. Wegner, J. Siast, K. Klimaszewski, O. Stankiewicz, and M. Domański, "Analysis of compressed data stream content in hevc video encoder," *International Journal of Electronics and Telecommunications*, vol. 61, no. 2, pp. 121–127, 2015.

[43] V. Sze and M. Budagavi, *High Efficiency Video Coding (Algorithms and Architectures)*, Springer, 2014.

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

[45] Xiph, "Xiph.org video test media," https://media.xiph.org/video/derf/, Accessed: 2018-11-01.

[46] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *Proc. IEEE ICIP'18*, 2018.

[47] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, "Learning features by watching objects move," in *Proc. IEEE CVPR'17*, 2017.

[48] F. Bossen, "Common HM test conditions and software reference configurations," in *ISO/IEC JTC1/SC29 WG11, JCTVC-L1100*, Jan. 2013.

[49] M. Wien, *High Efficiency Video Coding (Coding Tools and Specification)*, Springer, 2015.

[50] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," Apr. 2001, VCEG-M33.