Learning-based encoder algorithms for VVC in the context of the optimized VVenC implementation

Tech, Gerhard, George, Valeri, Pfaff, Jonathan, Wieckowski, Adam, Bross, Benjamin, et al.

Gerhard Tech, Valeri George, Jonathan Pfaff, Adam Wieckowski, Benjamin Bross, Heiko Schwarz, Detlev Marpe, Thomas Wiegand, "Learning-based encoder algorithms for VVC in the context of the optimized VVenC implementation," Proc. SPIE 11842, Applications of Digital Image Processing XLIV, 1184207 (1 August 2021); doi: 10.1117/12.2597228

SPIE.

Event: SPIE Optical Engineering + Applications, 2021, San Diego, California, United States

Learning-based encoder algorithms for VVC in the context of the optimized VVenC implementation

Gerhard Tech[a], Valeri George[a], Jonathan Pfaff[a], Adam Wieckowski[a], Benjamin Bross[a], Heiko Schwarz[a], Detlev Marpe[a], and Thomas Wiegand[ab]

[a]Department of Video Communication and Applications, Fraunhofer Institute for Telecommunications—Heinrich Hertz Institute, Berlin, Germany
[b]Department of Telecommunication Systems, Technical University of Berlin, Berlin, Germany

ABSTRACT

Versatile Video Coding (VVC) is the most recent and efficient video-compression standard of ITU-T and ISO/IEC. It follows the principle of a hybrid, block-based video codec and offers a high flexibility to select a coded representation of a video. While encoders can exploit this flexibility for compression efficiency, designing algorithms for fast encoding becomes a challenging problem. This problem has recently been attacked with data-driven methods that train suitable neural networks to steer the encoder decisions. On the other hand, an optimized and fast VVC software implementation is provided by Fraunhofer's Versatile Video Encoder VVenC. The goal of this paper is to investigate whether these two approaches can be combined. To this end, we exemplarily incorporate a recent CNN-based approach that showed its efficiency for intra-picture coding in the VVC reference software VTM to VVenC. The CNN estimates parameters that restrict the multi-type tree (MTT) partitioning modes that are tested in rate-distortion optimization. To train the CNN, the approach considers the Lagrangian rate-distortion-time cost caused by the parameters. For performance evaluation, we compare the five operational points reachable with the VVenC presets to operational points that we reach by using the CNN jointly with the presets. Results show that the combination of both approaches is efficient and that there is room for further improvements.

Keywords: VVC, VVenC, Fast Encoding, CNN, Partitioning

1. INTRODUCTION

The latest video compression standard finalized by ITU-T and ISO/IEC—Versatile Video Coding[1] (VVC)—specifies a great variety of coding modes. These modes achieve a bit rate reduction about 50%[2] compared to VVC's predecessor High Efficiency Video Coding[3] (HEVC). A requirement for these reductions is an encoder that chooses efficient encoding modes, i.e. modes that decrease the bit rate while maintaining the distortion of the coded video. Conventionally, this is done by rate-distortion optimization (RDO). RDO encodes a block with different mode combinations and selects the combination that minimizes the Lagrangian rate-distortion (RD) cost. This process is, however, time consuming and the encoding time usually limited.

A solution to this problem is to optimize the encoder implementation, for example algorithmically or by exploiting dedicated hardware features or parallelism. Another solution is to skip testing encoding modes in RDO, which, however, can lead to a reduced rate-distortion performance. Both approaches are used by Fraunhofer's VVenC software:[4,5] Based on an efficient implementation, VVenC allows encoding with 5 presets, which define different encoding configurations and thus different sets of VVC coding modes that are tested in RDO. Beyond this general selection of modes, VVenC comprises several methods that skip testing encoding modes adaptively on a block basis using heuristically derived rules.

Not yet applied for VVenC are learning-based approaches. In particular, CNN-based approaches for fast partitioning of intra-predicted pictures have recently become popular. These approaches often determine a block's partitioning mode directly or restrict which partitioning modes are tested in RDO. Most of them have

Further author information: (Send correspondence to Gerhard Tech)
E-mail: firstname.lastname@hhi.fraunhofer.de

Applications of Digital Image Processing XLIV, edited by Andrew G. Tescher, Touradj Ebrahimi, Proc. of SPIE
Vol. 11842, 1184207 · © 2021 SPIE · CCC code: 0277-786X/21/$21 · doi: 10.1117/12.2597228

Table 1. Tool settings for the five VVenC presets; *Red.* indicates that VVenC only tests a reduced number of modes; A description of the coding tools can be found in Ref. 10.

Tool	Abb.	*Slower*	*Slow*	*Medium*	*Fast*	*Faster*
Coding Tree Unit (CTU) Size		128	128	128	64	64
Minimum QT Size		8	8	8	4	4
Maximum MTT Depth		3	3	2	1	0
QT-BTT-Extra-Fast		1	2	2	2	2
Content-Based-Fast-QTBT		Off	Off	On	On	On
Intra Sub-Partitioning	ISP	On	Red.	Red.	Off	Off
Matrix-based Intra Prediction	MIP	On	Red.	Red.	Off	Off
Multi-Reference Line	MRL	On	On	On	Off	Off
Cross Component Linear Model	CCLM	On	On	On	On	On
Multiple Transform Selection	MTS	On	Red.	Red.	Red.	Red.
Low-Freq. Non-Separable Transform	LFNST	On	On	On	On	Off
Transform Skip	TS	On	On	On	On	On
Dependent Quantization		On	On	On	Off	Off
Joint Chroma Coding	JCCR	On	On	On	Off	Off
Adaptive Loop Filter	ALF	On	Red.	Red.	Red.	Off
Deblocking Filter Optimization		On	On	On	Off	Off
Luma Mapping/Chroma Scaling	LMCS	On	On	On	Off	Off

shown their efficiency in the reference or test model encoders of HEVC, the Joint Exploration Model[6] (JEM), or VVC. However, since these encoders are optimized only to a minor degree, the question is left open whether they are also efficient in the context of an encoder like VVenC. This is the motivation for this paper, which exemplarily evaluates one of the best performing approaches in the context of VVenC. To this end, this paper discusses VVenC in Section 2, the learning-based approach in Section 3, and their combination in Section 4. Then, Section 5 shows how the combined approach performs in encoding before Section 6 and Section 7 provide an outlook and a conclusion.

2. VVENC

The VVenC[5] software is originally based on the VVC reference software VTM.[7] Motivation to develop VVenC was to establish lightweight encoder with additional optimizations. By careful analysis of time-heavy critical parts and removing of redundant code, it was already possible to speed-up the original software. The additional algorithmic optimizations, including reducing of memory bandwidth, clever exploitation of inter-tool dependencies and extensive use of SIMD operations, significantly reduced the run-time to about 70% without RD performance losses. On top of this, VVenC offers a multi-threading mode.

Based on the work in Ref. 8 and Ref. 9, VVenC defines five presets that represent different operation points regarding the trade-off between encoding-speed and compression: *Slower*, *Slow*, *Medium*, *Fast*, and *Faster*. They are shown in Table 1. The *Slower* preset is nearly the same as the default setup of VTM. The other four presets disable encoding tools and enable tool specific speed-ups successively, such that the encoding time and compression efficiency decrease incrementally.

Furthermore, VVenC comprises the following tool specific speed-ups relevant for intra-picture coding (see also Ref. 11):

- *ISP*: A single split direction or the entire ISP estimation is early terminated depending on the results of the previously tested intra modes like angular/MIP/MRL or the first tested ISP mode.
- *MIP*: MIP is only tested for blocks that have a ratio of width and height that is below a threshold. Additionally, the number of the MIP mode candidates can be reduced.

- *QT-BTT-Extra-Fast*:[12] This speed-up estimates the future cost of a split mode and compares it to a threshold. This way, expensive partitioning configurations can be omitted. Additionally, splits depend on the remaining available partitioning depth and on the preceding splits on the same partitioning level.
- *Block Search History Cache*:[11] Because of different partitioning strategies, the encoder would test some blocks with the same position and size multiple times. In some cases, the multiple testing would produce identical coding results. For such kind of blocks, the encoder can use a search history cache: It performs the test only once and re-uses the cached result. This speed-up is already available in VTM.
- *Content-Based-Fast-QTBT*:[11] This speed-up exploits the spatial characteristics of the original signal through an evaluation of the directional gradient. It calculates horizontal, vertical and diagonal gradients. Based on them and some additional thresholds, the encoder skips particular split directions. This speed-up is already available in VTM.
- *Quad tree (QT) depending on MTT*:[11] If the binary splits do not reduce the coding cost, the QT partitioning can be skipped
- *MTT depending on QT*:[11] Depending on neighbor block partitioning heuristics, the encoder tests the QT before the MTT. Testing of particular MTT modes then depends on the results of the QT evaluation.

In summary, VVenC allows to operate at different points in the RD performance-time space, which have been empirically found by selecting general encoder settings, speed-up methods and encoding algorithms for specific tools. The actual performance of VVenC will be shown and discussed in Section 5.2.

3. LEARNING-BASED ENCODER ALGORITHM

In the last four years, several methods that control block partitioning with a CNN have been proposed. Based on original picture samples and other input data, they infer either partitioning modes directly or derive a set of modes for RDO. There are methods for the reference or test model encoders of HEVC, JEM, and VVC.

HEVC-based methods often use the CNN to determine the QT partitioning. The methods of Shi[13] and Li[14] decide locally whether to split a QT node further. In contrast, the idea of Xu[15] is to determine the whole QT structure for a CTU at once. Kuanar's method[16] does not decide on splits directly, but selects for each CTU between five presets of QT depths and prediction modes.

For JEM, methods often restrict the joint quad- and binary tree (QTBT). To do this efficiently, the CNN of Galpin[17] estimates split probabilities at lines in a 4×4 raster. Then, to decide whether to test a particular partitioning mode in RDO, his method aggregates the probabilities over the mode's parting lines and performs thresholding. Jin's approach[18] defines five quad- and binary-tree depth ranges. In encoding, the CNN selects one of these ranges per 32×32 block. In inter-coding, Wang's CNN[19] estimates a joint quad- and binary-tree depth for 64×64 blocks.

VVC supports partitioning with a joint QT and MTT, which allows a great variety of block sizes. To select modes efficiently, Tissier[20] uses an adapted version of Galpin's JEM method[17] (as described above). The CNN of Li[21] selects a splitting mode using different sub-nets to adapt to different block sizes. Tang's method[22] resolves the problem of different block sizes by sub-sampling and decides whether splitting is continued.

For training, the described methods use e.g., the cross entropy or the squared error with respect to an optimal mode selection. Some of them also consider RD performance decreases.[21] However, as the final target is minimizing the encoding time while maintaining the RD performance, we presented an approach in Ref. 23 that uses actually recorded Lagrangian rate-distortion-time (RDT) cost as training loss. The approach estimates parameters that restrict the tested VVC partitioning modes. In Ref. 24, we suggested and evaluated two variants of the approach: In the first one, the CNN estimates two further partitioning parameters; in the second one, it estimates additionally prediction and transform parameters. Moreover, we presented an approach in Ref. 25 that allows to restrict partitioning modes with higher flexibility using 42 parameters. However, to enable this, we could not longer use the actual RDT cost in training, but we had to mimic RDO while training to derive estimates from the recorded data.

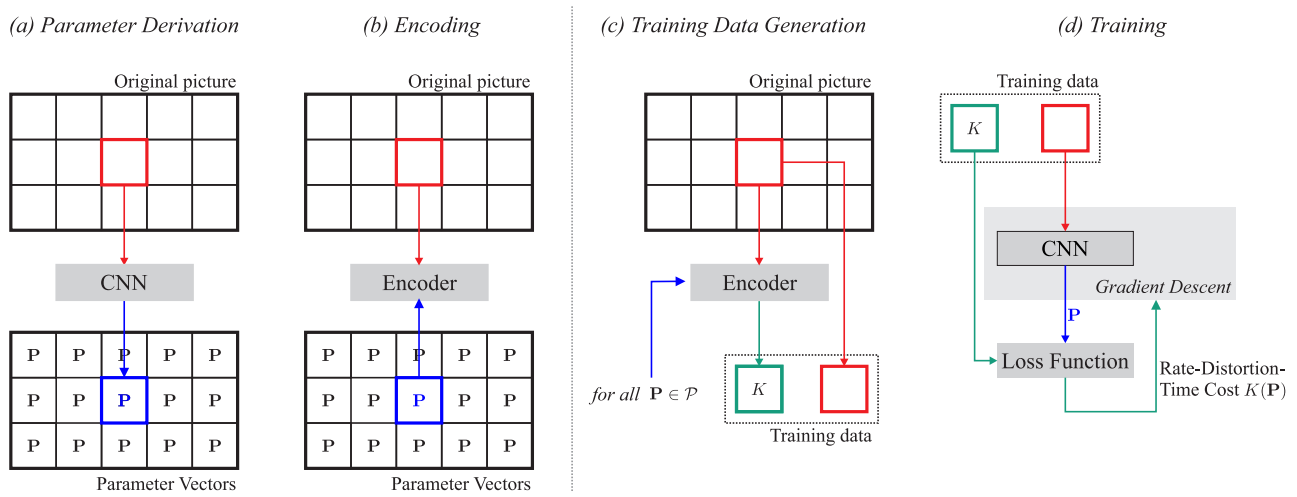

Figure 1. Overview of the CNN-based method: (a) and (b): Data-flow in encoding; (c) and (d): Steps of the training approach.

Using the VTM 10.2[7] encoder as basis, our RDT cost-driven approaches outperform the other VVC-based methods discussed above. And among them, the first variant from Ref. 24 and the method from Ref. 25 have about the same performance. As the former is easier to implement, we use it in this paper to exemplarily study whether CNN-based encoder optimization are also effective in the context of VVenC. For this, we recall the main steps of the method from Ref. 24 and our major ideas of the training approach from Ref. 23 in the remainder of this section.

3.1 Overview

Figure 1 shows an overview of our approach from Ref. 24. Before encoding a picture (Figure 1 a), the approach inputs the picture's original samples in a granularity of 32×32 blocks to a CNN. For each block, the CNN then estimates a parameter vector $\mathbf{P}$. In encoding, (b), when the encoder reaches a 32×32 block, it selects the partitioning modes that are tested by RDO based on the parameter vector $\mathbf{P}$ derived for the block. This way, the parameter derivation process and the encoding process are decoupled, so that they can be parallelized, for example on a picture level. Furthermore, also the parameters of different blocks can be derived in parallel, for example by inputting them as tensor to a GPU.

Figure 1 (c) and (d) show our training approach. It consists of two steps. The first step is training data generation (c), which encodes patches of several training sequences to explore and record the rate-distortion-time cost K related to different parameter vectors $\mathbf{P}$ from an available set $\mathcal{P}$. The second step (d) is the actual CNN training, which uses a gradient descent approach. The training uses the recorded cost data to determine the gradient of the training loss $K(\mathbf{P})$ caused by parameter vectors $\mathbf{P}$ provided by the CNN. This way, the CNN can learn to select parameters that minimize the encoding time while maintaining the rate-distortion performance.

In the following, Section 3.2 discusses how the parameter vectors $\mathbf{P}$ restrict the encoding. Then, Section 3.3 and Section 3.4 present the actual training approach and the CNN.

3.2 Partitioning Restrictions

VVC supports a quad tree (QT), which can start, for example, at 128×128 blocks and allows recursive quad splits (see Figure 2) down to a minimum leaf node size of typically 4×4. Quad tree leaf nodes not smaller than a minimum size and not larger than a maximum size can have a nested multi-type tree (MTT).[26] The MTTs allow binary and ternary splits in horizontal or vertical direction as also shown in Figure 2. Conventional settings for the minimum and maximum sizes are 8×8 and 32×32, respectively. With these settings, a 32×32 block can have an MTT at its single 32×32 QT node, its four 16×16 QT nodes, and its 16 8×8 QT nodes. Consequently, an encoder needs to evaluate up to 21 MTTs.

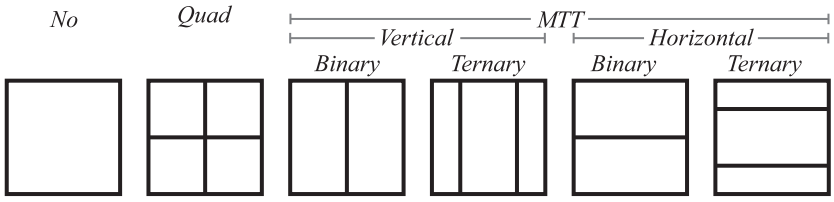

Figure 2. VVC partitioning modes in the QT and the MTT

From this it becomes clear that there is a great number of partitionings for a 32×32 block. The idea is now to skip partitioning modes that are sub-optimal for a given block B without testing in RDO. To this end, the CNN adaptively estimates the parameter vector $\mathbf{P}$, which restricts the tested MTT splits for B. As we suggested in Ref 24, $\mathbf{P}$ consist of four parameters $\mathbf{P} = [p_a, p_i, p_w, p_h]^T$.

Parameters p_a and p_i restrict the maximum and minimum size of QT nodes that can have an MTT further. More specifically, a QT node can only have an MTT when its size is not smaller than $32/2^{p_i}$ and not greater than $32/2^{p_a}$. Given the general minimum size of 8×8, p_a and p_i can be in the range $\{0, 1, 2\}$ and p_i must greater than or equal to p_a. Consequently, p_a and p_i jointly allow to select between six ranges of QT node sizes in which MTTs are allowed: $\{32\times32\}$, $\{32\times32, 16\times16\}$, $\{32\times32, 16\times16, 8\times8\}$, $\{16\times16\}$, $\{16\times16, 8\times8\}$, and $\{8\times8\}$,

Parameters p_w and p_h restrict the minimum width and height of blocks in the MTTs, however, not absolutely, but relatively to the size of their QT leaf nodes. This means that for a QT node size of $S\times S$, the minimum width and height are given by $S/2^{p_w}$ and $S/2^{p_h}$. MTT partitioning modes that would lead to sub-blocks with a smaller width or height are skipped in RDO. As the general minimum block size is 4, p_w and p_h can be in the range of $\{0, 1, 2, 3\}$, $\{0, 1, 2\}$, $\{0, 1\}$, for S equal to 32, 16, and 8, respectively.

In summary, combining the four parameters p_a, p_i, p_w, and p_h leads to value space with 70 valid combinations as shown in Figure 3. Selecting a combination allows to control from which QT sizes MTTs can start and at which minimum sub-block width and height they can end. Theoretically, it would be possible to control QT partitioning with even higher flexibility, e.g. by adding further parameters for different QT node positions. However, the number of parameter combinations increases exponentially with the number of parameters and, since our training data generation process (as discussed in Section 3.3.1) requires to evaluate all combinations, so does the time for generating the training data. Consequently, only parametrizations with a small number of parameters are feasible.[24]

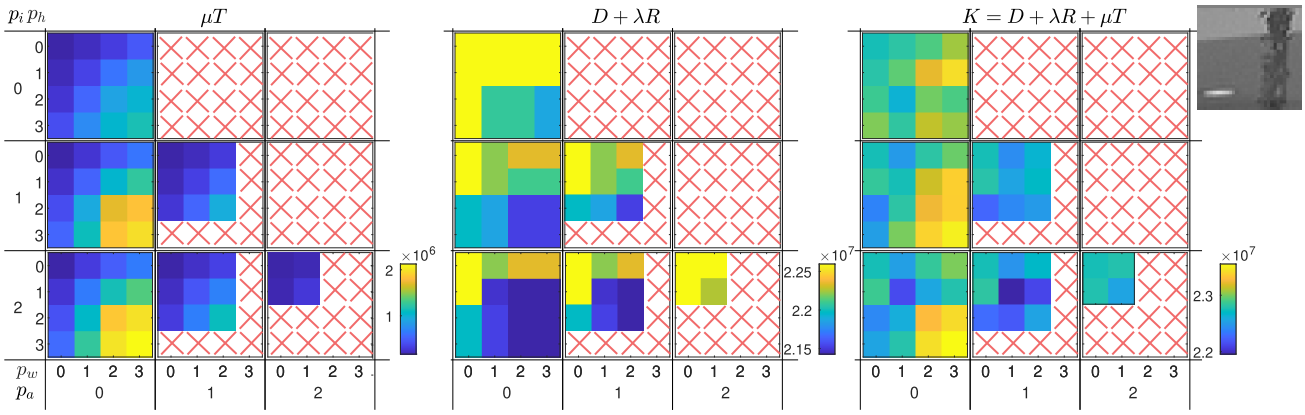

Figure 3. Example for the training data. The time cost μT, the RD cost $J = D + \lambda R$ and the rate-distortion-time cost $K = J + \mu T$ in dependence of different parameter vectors $\mathbf{P} = [p_a, p_i, p_w, p_h]^T$. Recorded for the patch shown top right. Combinations marked with x are invalid.

Algorithm 1 *Parameter Search* at a 32×32 block

1: **SAVE**(initial encoder state)
2: **for all P** valid for the 32×32 block **do**
3: **LOAD**(initial encoder state)
4: $J(\mathbf{P}), T(\mathbf{P}) = \mathbf{RDO}(\mathbf{P})$
5: $K(\mathbf{P}) = J(\mathbf{P}) + \mu \cdot T(\mathbf{P})$
6: Store $K(\mathbf{P})$ as training data
7: **SAVE**(encoder state of $\mathbf{P}$)
8: $\mathbf{P}^* = \arg\min K(\mathbf{P})$
9: **LOAD**(encoder state of $\mathbf{P}^*$)

3.3 CNN Training

We follow our idea from Ref. 23 and train the CNN such that it estimates the parameter vectors $\mathbf{P}$ that optimize the encoder's rate-distortion-time performance. These are the parameters with the least rate-distortion performance loss at a constant encoding time reduction. Theoretically, the optimal parameters can be found using the generalized Lagrangian multiplier method.[27] Given that several different parameters $\mathbf{P}$ are available for a current block B, the optimal choice is the parameter that minimizes the Lagrangian rate-distortion-time cost

$$K(\mathbf{P}) = D(\mathbf{P}) + \lambda \cdot R(\mathbf{P}) + \mu \cdot T(\mathbf{P}) = J(\mathbf{P}) + \mu \cdot T(\mathbf{P}) \tag{1}$$

with D, R, and T denoting the distortion, bits, and encoding time for encoding block B with $\mathbf{P}$; λ and μ denoting the Lagrangian multipliers; and J denoting the rate-distortion cost. Under the assumption that the RDT costs for different blocks B are independent, such a per-block choice also minimizes the overall rate-distortion-time cost.[27]

Consequently, when we train the CNN using Equation (1) as training loss, it will provide an estimate of the parameters that maximize the rate-distortion-time performance. As already shown in Figure 1, aspects of this approach are the training data generation process and the actual loss calculation in training, which we discuss in Section 3.3.1 and Section 3.3.2, respectively.

3.3.1 Training Data Generation

For training data generation, we encode several training sequences using a modified encoding process, called *parameter search*, which we introduced in Ref. 23. The *parameter search* corresponds to normal encoding process, however, when it reaches a 32×32 block B, it operates as shown in Algorithm 1: For each valid parameter combination $\mathbf{P}$ (line 2), it repeats the RDO process for block B (line 4) and stores the related RDT cost (line 6). Finally, it continues with the results of the parameter $\mathbf{P}^*$ that provides the minimal RDT cost (line 8). Further modifications maintain the correct encoder state: To ensure, that the parameters $\mathbf{P}$ are always evaluated based on the same state, the *parameter search* stores the initial state before testing the first parameter (line 1) and reloads it before testing the a next parameter vector $\mathbf{P}$ (line 3). Furthermore, it also stores the state after encoding with a parameter $\mathbf{P}$ (line 7) and finally reloads the state of the best performing parameter $\mathbf{P}^*$. Consequently, the *parameter search* evaluates further blocks under the condition that optimal parameters have been selected for the blocks encoded before.

In summary, the *parameter search* explorers and records the exact RDT cost for all valid parameter combinations $\mathbf{P}$ to generate training data (exemplarily shown in Figure 3). Moreover, it determines per block of the picture the optimal parameter vector $\mathbf{P}^*$. Thus, when we perform an additional encoder run using the optimal parameter vector $\mathbf{P}^*$ per block without the search, then we also reveal the highest possible rate-distortion-time performance.

Furthermore, what should be noted is that results of the *parameter search* depend on the selection of the Lagrange multiplier μ (see line 5). Considering Equation (1), this means that increasing μ will lead to a selection optimal parameters that produce a lower encoding time. Moreover, as the recorded RDT cost are used in training, this will lead to a CNN that estimates parameters that produce a lower encoding time. Or, in other words, we can use the multiplier μ to adjust the encoding time reduction associated with the trained CNN.

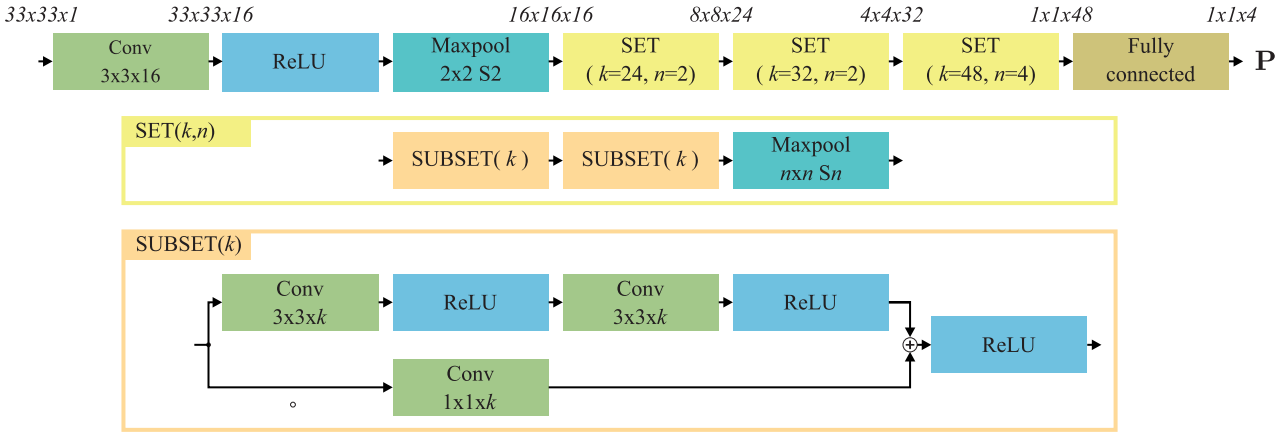

Figure 4. The CNN: It has been proposed by Galpin.[17] Conv 3x3xk indicates a 3×3 convolution with k output channels. Maxpool nxn Sn indicates maximum pooling with a kernel size of $n \times n$ and a stride of n.

3.3.2 Loss Function in Training

As we suggested in Ref. 23, we train the CNN such that it estimates parameters that minimize the Lagrangian rate-distortion time cost given by Equation (1). For training, we apply a gradient descent approach. Thus, when the CNN outputs a parameter vector $\mathbf{P}$ in training, we require the related derivatives of the RDT cost $\partial K/\partial p_j$ with respect to the elements p_j with $j \in \{i, a, w, h\}$ of $\mathbf{P}$.

We derive them from the recorded training data for the block B in two steps.[24] The first step augments the recorded training data with RDT cost values for invalid parameter combinations, i.e. for combinations for which no data has been recorded (marked with red crosses in Figure 3. More specifically, it extrapolates RDT cost values for the invalid combinations comprised by the Cartesian product of the ranges $\{0, 1, 2\}$ for p_a and p_i, and $\{0, 1, 2, 3\}$ for p_w and p_h in multiple iterations. In each iteration, invalid combinations that are neighbored by recorded or already extrapolated combinations are set to their neighbor's maximum, where an increasing offset is added. Consequently, the farther away a position is from a valid position the higher its RDT cost. The second step then fits piecewise cubic hermite splines[28] to the augmented data and determines $\partial K/\partial p_j$ as derivatives of the splines.

In summary, this procedure resolves the problem that the recorded data are sampled at discrete points of a non-differentiable function by assuming that the function is differentiable and using interpolation. It should be noted that the first step, i.e. the extrapolation of increasing values for invalid positions, ensures that the gradient descent method converges to the area of valid parameter combinations. To achieve this also for parameter values outside the area comprised by the augmented or recorded combinations (i.e. p_i or p_a less than 0 or greater than 2; or p_w or p_h less than 0 or greater than 3), we use an artificial gradient there that points back to this area.

3.4 The CNN

Figure 4 shows the CNN that we apply for our method. It has been proposed by Galpin[17] and has also been used by Tissier.[20] It is a residual CNN. To use it for our approach, we changed the input block size from 64×64 to 32×32 (both plus a top and left margin of one sample) and the number of outputs from 480 to four. It comprises three sets of layers, each consisting of two subsets of layers. Each subset contains two convolutional layer and a skip connection. The final layer of the CNN is fully connected.

4. LEARNING-BASED ENCODING WITH VVENC

This section describes how we integrated the CNN-based method to VVenC in Section 4.1, discusses how we generated CNNs for different VVenC presets in Section 4.2, and summarizes the training conditions in Section 4.3.

4.1 CNN Integration

To estimate the partitioning parameters $\mathbf{P}$, we added a new module to VVenC that implements the CNN using the Dlib software library.[29] As shown in Figure 1, the module infers the parameters for the whole picture at once before encoding is started. In principle, the parameter inference could also happen in parallel to the encoding process. In encoding, we use the parameters only to restrict MTTs of the luma component. Furthermore, when using a CNN with one of the presets, which are described in Section 2, we set the maximum luma MTT depth always to three regardless of the value defined in Table 1. This way, the CNN can adaptively select from the full range of the 70 parameters as described in Section 3.2. Consequently, when using a CNN with a particular preset having originally an MTT depth less than three, the CNN could trade-off encoding speed against RD performance, such that the encoding time becomes higher than the particular preset's encoding time.

4.2 CNN Training for Different Presets

On the one hand, VVenC offers different presets to adjust the encoding time. On the other hand, our CNN-based approach allows to adapt the encoding time by selecting a Lagrange multiplier μ as described in Section 3. Thus, given a desired encoding time reduction point, the question is which combination of VVenC preset and Lagrange multiplier provides the highest RD performance and how this performance compares to using VVenC without the CNN. To explore this in Section 5, we generate CNNs with different combinations of VVenC presets, QPs, and Lagrangian parameters μ.

In total, we generate 120 CNNs, one for each of the combinations of the five VVenC presets s, the four QPs $q \in \{22, 27, 32, 37\}$, and the six target encoding time points $t^*(s)$, which we select for each VVenC preset s individually. To enforce that the encoding time reduction is the same at each QP, we derive for each CNN an individual Lagrange multiplier $\mu(s, q, t^*(s))$ that we then use to generate the CNN's training data.[23] For derivation, we consider the following measures:

- $T(v, s, q, \mu)$ is the time for encoding a sequence v with the VVenC preset s, the QP q, and the optimal parameters for the Lagrange parameter μ, which we obtain using the *parameter search* as described in Section 3.3.1.
- $T_{Ref}(v, q)$ is the reference encoding time, which we obtain by encoding sequence v with the unmodified VVenC, the *Slower* preset, and the QP q.
- $t(s, q, \mu)$ is the actual reduced encoding time and given by averaging $T(v, s, q, \mu)/T_{Ref}(v, q)$ over all sequences v.

We use these measures to find a particular Lagrangian multiplier $\mu(s, q, t^*(s))$ as follows: First, we encode one frame of each sequence v of our test set (described in Section 5.1) with the *parameter search* to determine $T(v, s, q, \mu)$ for different values of μ. Then, we set $\mu(s, q, t^*(s))$ to the value of μ that produces the encoding time $t(s, q, \mu)$ that is closest to the target encoding time $t^*(s)$.

In summary, this approach selects the Lagrangian multipliers μ such that encoding with their optimal parameters leads to approximately the same encoding time reduction at each QP. Consequently, it also enforces that the encoding time reduction is roughly the same when using the CNNs trained with the RDT cost data obtained with these Lagrangian multipliers. This is important, as we will evaluate the rate-distortion-time performance of our approach in Section 5.2 considering the average bit rate reduction obtained with the four QPs, as well as the average encoding time reduction achieved with them. Such an assessment would not be very conclusive when the encoding time reductions would vary strongly at the four QPs.

4.3 Training Conditions

For training, we use a framework based on the Dlib software library.[29] We train each of the 120 CNNs with about 22 million patches and their related RDT cost data. To obtain the RDT cost, we employ training data generation, as described in Section 3.3.1, using sequences of the RAISE training set.[30] In the training process, we present all patches 24 times to the CNN and use the Adam optimizer with learning rate of 10^{-4} and a mini-batch size of 512.

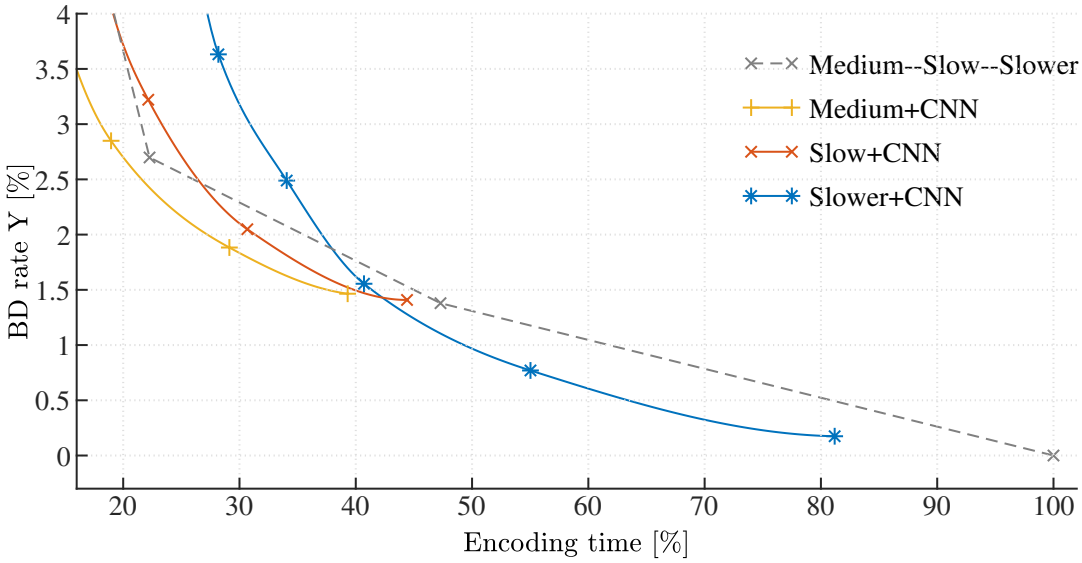

Figure 5. Slow operational range: BD rate increase in dependence of the encoding time.

5. EVALUATION

After summarizing test conditions in Section 5.1, this section discusses how encoding with the CNN compares to encoding with the unmodified VVenC software in Section 5.2.

5.1 Test Conditions

For evaluation, we used VVenC version 1.0.0[31] and followed the common test conditions[32] of the Joint Video Experts Team (JVET). We present results in terms of the average bit rate reduction using the Bjøntegaard bit rate delta[33] (with respect to the luma PSNR) and the average encoding time reduction. We average over the results for sequences of the JVET test set classes A1, A2, B, C, D, and E.[32] Anchor is encoding with the VVenC *Slower* preset. Since the CNN can be evaluated in parallel to the encoding, we report the encoding times without considering the CNN's inference time (which is about 2% of the anchor's encoding time). For encoding, we used Supermicro FatTwin nodes with Intel Xeon E5-2697A v4 processors and Scientific Linux 7.

5.2 Experimental Results

To evaluate the performance of VVenC using the CNN, we consider two different operational ranges of RD-time performance tradeoffs. In the first range, the encoding time is high and the RD performance loss is modest, while in the second range the encoding time is low and the RD performance loss is high. Results for the ranges are shown in Figures 5 and 6.

Both figures show points obtained with the unmodified VVenC software using different configuration presets. These points are connected by dashed lines which are linearly interpolated. The dashed lines approximately show points that could be reached by 1) using a particular preset for n% of all CTUs; 2) using the preset following the particular preset for the remaining $(100 - n)$% CTUs, and 3) varying n from 0 to 100. In the following, we use them as reference for the performance evaluation. Furthermore, solid curves show results for VVenC using the CNN. Here, we used polynomials for interpolation. This is justified, since these points can be reached by varying the Lagrange multiplier μ in training data generation as described in Section 3.3.1.

The slow operational range, shown in Figure 5, comprises the VVenC presets, *Slower*, *Slow*, and *Medium*. When using *Slow* instead of *Slower*, the BD rate increases by about 1.4% while the encoding time decreases to 47%. It can be seen in Table 1, that this is caused by a faster setting for the QT-BTT-Extra-Fast method and reduced testing for ISP, MIP, MTS, and ALF. The curve *Slower+CNN* shows that using the CNN together with *Slower* preset outperforms these configuration changes. For example at an encoding time of about 60%, the BD rate decreases by about 0.4% or at 0.6% BD rate the encoding time decreases about 16%.

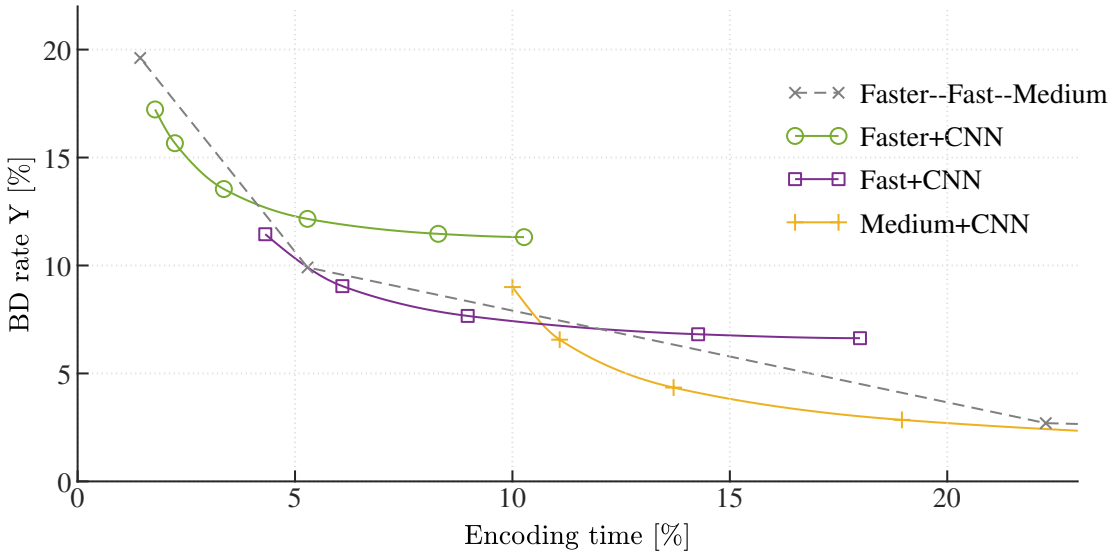

Figure 6. Fast operational range: BD rate increase in dependence of the encoding time.

Using the *Medium* preset instead of the *Slow* preset enables Content-Based-Fast-QTBT and decreases the MTT depth from 3 to 2. Consequently, the BD rate increases further from 1.4% to 2.7% and the encoding time decreases from 47% to 22%. As the curves in Figure 5 show, it is more beneficial to use the CNN in this encoding time range: For *Slow+CNN* only in a part of the range, for *Medium+CNN* in the whole range. As described in Section 4.2, the *Medium+CNN* setup can slow the *Medium* preset down, as it allows a higher luma MTT depth, but restricts the tested MTT partitionings adaptively using the CNN. With this approach, we decrease the rate about 0.5% at 30% encoding time, or the encoding time by about 8% at a BD rate of 1.8%.

Figure 6 shows the fast operational range, which comprises the VVenC presets *Medium*, *Fast*, and *Faster*. Considering the change from *Medium* to *Fast* the encoding time drops from 22% to 6% while the BD rate increases from 2.7% to 9.9%. This is caused by the following: Using a smaller CTU and minimum CU size; reducing the MTT depth to 1; and disabling ISP, MIP, MRL, Dependent Quantization, JCCR, LMCS, and the deblocking filter optimization entirely. It can be seen that *Medium+CNN* outperforms these changes until an encoding time of 11% is reached. Moreover, *Fast+CNN* performs better in the range from 12% to 5%. In summary, both approaches—decreasing and increasing the encoding time with the CNN based on the *Medium* and the *Fast* preset, respectively—are efficient and reduce the BD rate loss by about 0.8% and 2% at encoding times of 8% and 14%, respectively.

When changing the preset from *Fast* to *Faster*, the MTT, LFNST and ALF are disabled entirely. Consequently, the BD rate increase changes from 9.9% to 19.6% and the encoding time drops from to 5.2% to 1.4%. In this range, especially *Faster+CNN* is efficient and achieves, compared to the preset change, a bit rate reduction of about 2% at an encoding time of 2.5% .

6. OUTLOOK

As described in Section 3.4, our method currently uses only a very basic CNN, which can run in parallel to the encoding process. Moreover, the CNN run-time is currently only about 2% of the run time of the *Slower* preset and further speed-ups are possible e.g. by using a GPU or parallel evaluation of the 32×32 blocks. Consequently, it would be feasible to use more complex and deeper CNNs to improve the estimation accuracy. Furthermore, also additional CNNs for other or more parameters could be employed.

Another aspect that might be interesting for further research concerns the configuration that we use in combination with the CNN. In this paper, we evaluated the CNN-based method using the five pre-defined VVenC presets as starting point. Here, using other configurations might lead to further performance gains.

7. CONCLUSION

In this paper, we evaluated how learning-based encoder algorithms perform in the context of VVenC. Exemplarily, we incorporated a method that controls the partitioning of the MTT with a CNN to VVenC. The CNN estimates four parameters for each 32×32 block. The parameters then restrict, which MTT partitionings are tested by RDO. We trained the CNN considering the rate-distortion-time cost.

We compared the method to the unmodified VVenC at several operational points in the RD performance-time space. To vary the encoding time of VVenC, we successively disabled an increasing number of tools with five presets. For VVenC using the CNN, we also used the presets, but additionally employed CNNs trained for different encoding time reductions or increases. The evaluations showed that using a current preset with the CNN can outperform using the next preset without the CNN. Or in other words, restricting the MTT modes for RDO adaptively with the CNN can outperform disabling or restricting different prediction, transform, or filtering modes globally. This is in particular distinct at higher encoding times.

In conclusion, the results exemplarily show that learning-based algorithms can efficiently be combined with the optimized VVenC implementation to increase the rate-distortion-time performance. Further research makes sense since there is room for further improvements considering, e.g., more advanced CNNs, partitioning restrictions that are more effective at higher encoding speeds, or other encoder presets.

REFERENCES

[1] ITU-T and ISO/IEC JTC 1, "Versatile video coding (ITU-T Rec. H.266 and ISO/IEC 23090-3)," (Aug. 2020).

[2] Wien, M. and Baroncini, V., "Report on VVC compression performance verification testing in the SDR UHD Random Access Category (JVET-D0117)," *Joint Video Experts Team (JVET)* (Oct. 2020).

[3] ITU-T and ISO/IEC JTC 1, "High efficiency video coding (ITU-T Rec. H.265 and ISO/IEC 23008-2)," (Apr. 2013).

[4] Wieckowski, A., Brandenburg, J., Hinz, T., Bartnik, C., George, V., Hege, G., Helmrich, C., Henkel, A., Lehmann, C., Stoffers, C., Zupancic, I., Bross, B., and Marpe, D., "VVenC: An open and optimized VVC encoder implementation," in [*IEEE Int. Conf. on Multimedia Expo Workshops (ICMEW)*], 1–2 (2021).

[5] Brandenburg, J., Wieckowski, A., Hinz, T., and Bross, B., "VVenC Fraunhofer versatile video encoder." `https://www.hhi.fraunhofer.de/fileadmin/Departments/VCA/MC/VVC/vvenc-v1.0.0-v1.pdf` (May 2021).

[6] Joint Video Experts Team (JVET), "Joint exploration model." `https://jvet.hhi.fraunhofer.de/svn/svn_HMJEMSoftware/` (Sept. 2018).

[7] Joint Video Experts Team (JVET), "VVC reference software, VTM-10.2." `https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/` (November 2020).

[8] Brandenburg, J., Wieckowski, A., Hinz, T., Henkel, A., George, V., Zupancic, I., Stoffers, C., Bross, B., Schwarz, H., and Marpe, D., "Towards fast and efficient VVC encoding," in [*IEEE Int. Workshop on Multimedia Signal Process. (MMSP)*], 1–6 (2020).

[9] Wieckowski, A., Stoffers, C., Bross, B., and Marpe, D., "VVenC, an open optimized VVC encoder, in versatile application scenarios," in [*To appear in Applications of Digital Image Processing XLIV*], International Society for Optics and Photonics, SPIE (2021).

[10] Bross, B., Wang, Y.-K., Ye, Y., Liu, S., Sullivan, G., and Ohm, J.-R., "Overview of the versatile video coding (VVC) standard and its applications," *To appear in IEEE Trans. on Circuits and Syst. for Video Tech.* (Dec. 2020).

[11] Wieckowski, A., Ma, J., Schwarz, H., Marpe, D., and Wiegand, T., "Fast partitioning decision strategies for the upcoming versatile video coding (VVC) standard," in [*IEEE Int. Conf. on Image Process. (ICIP)*], 4130–4134 (2019).

[12] Wieckowski, A., Bross, B., and Marpe, D., "Fast partitioning strategies for VVC and their implementation in an open optimized encoder," in [*IEEE Picture Coding Symp. (PCS)*], (2021).

[13] Shi, J., Gao, C., and Chen, Z., "Asymmetric-kernel CNN based fast CTU partition for HEVC intra coding," in [*IEEE Int. Symp. on Circuits. and Syst. (ISCAS)*], 1–5 (2019).

[14] Li, T., Xu, M., and Deng, X., "A deep convolutional neural network approach for complexity reduction on intra-mode HEVC," in [*IEEE Int. Conf. on Multimedia and Expo*], 1255–1260 (2017).

[15] Xu, M., Li, T., Wang, Z., Deng, X., Yang, R., and Guan, Z., "Reducing complexity of HEVC: A deep learning approach," *IEEE Trans. on Image Process.* **27**(10), 5044–5059 (2018).

[16] Kuanar, S., Rao, K. R., and Conly, C., "Fast mode decision in HEVC intra prediction, using region wise CNN feature classification," in [*IEEE Int. Conf. on Multimedia Expo Workshops (ICMEW)*], 1–4 (2018).

[17] Galpin, F., Racapé, F., Jaiswal, S., Bordes, P., Le Léannec, F., and François, E., "CNN-based driving of block partitioning for intra slices encoding," in [*IEEE Data Compression Conf.*], 162–171 (2019).

[18] Jin, Z., An, P., Shen, L., and Yang, C., "CNN oriented fast QTBT partition algorithm for JVET intra coding," in [*IEEE Vis. Commun. and Image Process.*], 1–4 (2017).

[19] Wang, Z., Wang, S., Zhang, X., Wang, S., and Ma, S., "Fast QTBT partitioning decision for interframe coding with convolution neural network," in [*IEEE Int. Conf. on Image Process. (ICIP)*], 2550–2554 (2018).

[20] Tissier, A., Hamidouche, W., Vanney, J., Galpin, F., and Menard, D., "CNN oriented complexity reduction of VVC intra encoder," in [*IEEE Int. Conf. on Image Process. (ICIP)*], 3139–3143 (2020).

[21] Li, T., Xu, M., Tang, R., Chen, Y., and Xing, Q., "DeepQTMT: a deep learning approach for fast QTMT-based CU partition of intra-mode VVC," *arXiv:2006.13125* (2020).

[22] Tang, G., Jing, M., Zeng, X., and Fan, Y., "Adaptive CU split decision with pooling-variable CNN for VVC intra encoding," in [*IEEE Vis. Commun. and Image Process.*], 1–4 (2019).

[23] Tech, G., Pfaff, J., Schwarz, H., Helle, P., Wieckowski, A., Marpe, D., and Wiegand, T., "Fast partitioning for VVC intra-picture encoding with a CNN minimizing the rate-distortion-time cost," in [*IEEE Data Compression Conf.*], 3–12 (2021).

[24] Tech, G., Pfaff, J., Schwarz, H., Helle, P., Wieckowski, A., Marpe, D., and Wiegand, T., "CNN-based parameter selection for fast VVC intra-picture encoding," in [*To appear in IEEE Int. Conf. on Image Process. (ICIP)*], (2021).

[25] Tech, G., Pfaff, J., Schwarz, H., Helle, P., Wieckowski, A., Marpe, D., and Wiegand, T., "Rate-distortion-time cost aware CNN training for fast VVC intra-picture partitioning decisions," in [*IEEE Picture Coding Symp. (PCS)*], (2021).

[26] Li, X., Chuang, H., Chen, J., Karczewicz, M., Zhang, L., Zhao, X., and Said, A., "Multi-Type-Tree (JVET-D0117)," *Joint Video Experts Team (JVET)* (Oct. 2016).

[27] Everett, H., "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources," *Operations Res.* **11**(3), 399–417 (1963).

[28] Fritsch, F. N. and Carlson, R. E., "Monotone piecewise cubic interpolation," *SIAM J. on Numer. Anal.* **17**(2), 238–246 (1980).

[29] King, D. E., "Dlib-ml: A machine learning toolkit," *J. of Mach. Learn. Res.* **10**, 1755–1758 (2009).

[30] Dang-Nguyen, D.-T., Pasquini, C., Conotter, V., and Boato, G., "RAISE: A raw images dataset for digital image forensics," in [*Proc. of the 6th ACM Multimedia Syst. Conf.*], 219–224 (2015).

[31] Fraunhofer Heinrich Hertz Institute (HHI), "VVenC, version VVenC-1.0.0." `https://github.com/fraunhoferhhi/vvenc/releases/tag/v1.0.0` (Aug. 2021).

[32] Bossen, F., Boyce, J., Li, X., Seregin, V., and Sühring, K., "VTM common test conditions and software reference configurations for SDR video (JVET-T2010)," *Joint Video Experts Team* (Oct. 2020).

[33] Bjøntegaard, G., "Calculation of average PSNR difference between RD curves," *VCEG-M33* (Oct. 2001).