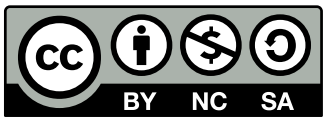
Aalto University
School of Science

Master's programme in ICT-Innovation

Two-Stage Overfitting of Neural Network-Based Video Coding In-Loop Filter

József-Hunor Jánosi

Master's Thesis
2023



Aalto University
School of Science

Author József-Hunor Jánosi

Title Two-Stage Overfitting of Neural Network-Based Video Coding In-Loop Filter

Degree programme ICT-Innovation

Major Data Science

Supervisor Prof. Juho Kannala

Advisor Francesco Cricri, Ph.D.

Collaborative partner Nokia

Date 25 September 2023 **Number of pages** 52 **Language** English

Abstract

Modern video coding standards like the Versatile Video Coding (VVC) produce compression artefacts, due to their block-based, lossy compression techniques. These artefacts are mitigated to an extent by the in-loop filters inside the coding process. Neural Network (NN) based in-loop filters are being explored for the denoising tasks, and in recent studies, these NN-based loop filters are overfitted on test content to achieve a content-adaptive nature, and further enhance the visual quality of the video frames, while balancing the trade-off between quality and bitrate. This loop filter is a relatively low-complexity Convolutional Neural Network (CNN) that is pretrained on a general video dataset and then fine-tuned on the video that needs to be encoded. Only a set of parameters inside the CNN architecture, named multipliers, are fine-tuned, thus the bitrate overhead, that is signalled to the decoder, is minimized. The created weight update is compressed using the Neural Network Compression and Representation (NNR) standard. In this project, an exploration of high-performing hyperparameters was conducted, and the two-stage training process was employed to, potentially, further increase the coding efficiency of the in-loop filter. A first-stage model was overfitted on the test video sequence, it explored on which patches of the dataset it could improve the quality of the unfiltered video data, and then the second-stage model was overfitted only on these patches that provided a gain. The model with best-found hyperparameters saved on average 1.01% (Y), 4.28% (Cb), and 3.61% (Cr) Bjontegaard Delta rate (BD-rate) compared to the Versatile Video Coding (VVC) Test Model (VTM) 11.0 NN-based Video Coding (NNVC) 5.0, Random Access (RA) Common Test Conditions (CTC). The second-stage model, although exceeded the VTM, it underperformed with about 0.20% (Y), 0.23% (Cb), and 0.18% (Cr) BD-rate with regards to the first-stage model, due to the high bitrate overhead created by the second-stage model.

Keywords neural video coding, in-loop filter, overfitting, two-stage training, content-adaptation, hyperparameter-tuning

Acknowledgement

This master's thesis and degree program has been a journey, one that I couldn't have completed without significant guidance and support. Immense thanks to my industrial advisor, Francesco Cricri for your invaluable insights. Gratitude to my colleagues at Nokia: María, Ruiying, Honglei, and Zhijie for your collaborative spirit. Gratitude goes towards Juho Kannala, who acted as my supervisor. My deepest appreciation goes to my loving parents because they always believed in me. Additionally, I would like to thank my university peers from the University of Twente, and Aalto University; our shared experiences have been integral to this academic journey.

Thank you all for making this possible.

Otaniemi, 25 September 2023

József-Hunor Jánosi

Contents

Symbols and abbreviations

Symbols

σ	neural network activation function
$*$	convolutional operation
$\mu_{\mathbf{x}}$	mean of vector $\mathbf{x}$
$\sigma^2_{\mathbf{x}}$	variance of vector $\mathbf{x}$
$\sigma_{\mathbf{xy}}$	covariance of vectors $\mathbf{x}$ and $\mathbf{y}$

Abbreviations

AI	artificial intelligence
ALF	adaptive loop filter
B-frame	bidirectional predicted frame
BD-rate	Bjøntegaard's delta-rate
BS	boundary strength
CABAC	context-based adaptive binary arithmetic coding
CC-ALF	cross-componenet ALF
CNN	convolutional neural network
CP-decomposition	CANDECOMP/PARAFAC decomposition
CTC	common test conditions
CTU	coding tree unit
CU	coding unit
Cb, Cr	blue-, red-difference chroma, chrominance
DBLK	deblocking filter
DCT	discrete cosine transform
dPSNR	dPSNR
fps	frames per second
GOP	group of pictures
H.264/AVC	Advanced Video Coding
H.265/HEVC	High Efficiency Video Coding
H.266/VVC	Versatile Video Coding
HDR	high dynamic range
I-frame	intra-coded frame
JVET	Joint Video Experts Team
LMCS	luma mapping with chroma scaling
LSTM	long short-term memory NN
LR	learning rate
MAC	multiply-accumulate
MAE	mean absolute error
ML	machine learning
MPEG	Moving Picture Experts Group
MSE	mean squared error
MSSIM	multiscale structural similarity index

NNR	neural network coding and representation
NNVC	NN-based video coding
P-frame	predicted frame
POC	picture order count
PSNR	peak signal-to-noise ratio
PU	prediction unit
QP	quantization parameter
QStep	quantization parameter step
QoE	quality of experience
RA	random access
RDO	rate-distortion optimization
RHCNN	residual highway CNN
RNN	recurrent neural networks
ReLU	rectified linear unit
SADL	small ad-hoc deep-learning library
SAO	sample adaptive offset
SDR	standard dynamic range
SEI	supplemental enhancement information
SPS	sequence parameter set
UHD	ultra-high definition
VTM	VVC test model
WWW	world wide web
Y	luma, luminance

1 Introduction

The amount of people connected to the World Wide Web (WWW) is growing by large margins every year, and the volume and demand for online content is also expanding. It was predicted that in 2022, the internet traffic's 82% will consist of video content, and also the quality of these videos is improving rapidly. Cisco has anticipated that by 2023, 4K or Ultra High Definition (UHD) videos will make up for 66% of connected flat-panel TV sets [1, 2]. With high-resolution video demand, the internet bandwidth is not growing at the same pace, thus compression algorithms are being exploited and developed to accommodate the quality of videos. Since research into deep learning applications is on the rise, their usage is also employed by new video compression techniques [3].

Compression can either be lossless, if the decoded data is bit-exact to the original information, or lossy, if, during compression, some information is lost. Lossy compression is used to decrease the bitrate needed to represent the data, which is advantageous for storing purposes, however, the quality of the reconstructed data is hindered. The goal of video coding is to maximize the video quality while minimizing the bitrate needed to store the content. This oftentimes comes off as a trade-off between the two. In our fast-growing world of video streaming, it is essential to represent the video data in as less bits as possible, and even if the modern video compression algorithms are only able to outperform the preceding standardized techniques by a small margin (a couple of percentage points), in the vast amount of data that is processed every day, those small margins account to big storing and processing impact.

Modern video coding standards, such as the High Efficiency Video Coding (H.265/HEVC) [4], and the Versatile Video Coding (H.266/VVC) [5], are block-based compression frameworks, which, despite having advanced coding efficiency, introduce blocking, ringing, and blurring artefacts. The purpose of the in-loop filters is to mitigate these artefacts with various techniques. For example, in the modern VVC standard, there are successive in-loop filters, such as Deblocking Filter (DBLK), Sample Adaptive Offset (SAO), and Adaptive Loop Filter (ALF). Although these filters can alleviate the compression flaws and increase the visual quality, they do so only to an extent, due to them being handcrafted with the assumption of stationary signals in signal processing theory. However, natural video sequences are regularly non-stationary, thus the efficiency of the implemented in-loop filters is limited [6].

Neural Networks (NN) are employed to learn on video data and act as a filter next to, or in place of other filters inside the existing video coding standards. These NN-based filters can improve visual quality by dealing with multiple artefacts simultaneously. NN-based filters can be post-processing or in-loop filters that are used in hybrid video coding techniques. A frame processed with an in-loop filter will stay as a reference to encode other frames, while post-processing filters are applied after the decoding and before outputting. Most NN-based filters are trained on large video datasets, their architecture is sophisticated, and they count on favourable generalization instead of being content-specific. While investigations have been made with content-adaptation in video coding, research in this field is still scarce, but growing [7–9].

In recent studies, Convolutional Neural Networks (CNN) have been used as loop

filters, which have low complexity and can provide good coding gains. The content-adaptive fine-tuning in CNNs can be done by first pretraining the network on a large general dataset, and then overfitting it on one video sequence. The overfitted network's weights are then signalled to the decoder side, where it reconstructs the overfitted CNN loop filter. To minimize the signalling bitrate, only a subset of weights are fine-tuned, such as the biases of the network, or introduced variables, called multipliers, that multiply the feature tensor output with a convolutional layer [10, 11].

1.1 Objectives of the study

In this work, optimal hyperparameter search was conducted on a low-complexity, and content-adaptive CNN in-loop filter, to maximize the coding efficiency, while also proposing two-stage overfitting as content-adaptive NN training for video coding. For the two-stage training, a first-stage model was chosen that measured its performance on the elements of the dataset, and the blocks that had negative Peak Signal-to-Noise Ratio-gain (dPSNR) over the unfiltered video patches, were removed, forming a subset of the video data. The second-stage model overfits on this subset of data, to gain even greater performance on the subset of the dataset. With this two-stage training, better Bjøntegaard Delta rate (BD-rate) [12] is anticipated, compared to the first-stage model's, because overfitting is usually accomplished on a smaller sample size [13].

With this research, insights are awaited on the different hyperparameters and their effects, and on the achievements and limitations of two-stage overfitting.

1.2 Structure of the thesis

The background work, and literature review, where the key concepts of video coding, and neural video coding are presented is written in Section 2. In Section 3, the methodology of the experiments is presented, as well as an explanation of the architecture, and setting used to produce the analysis. The results of the experiments, divided into two subsections of hyperparameter tuning, and two-stage overfitting, are shown in Section 4. The breakdown and analysis of these results and metrics are discussed in Section 5, where next to the interpretation, also prospects of future research are considered. A conclusion is done over the findings and the accomplishments of the thesis are written in Section 6.

2 Literature review

2.1 Video coding

Video coding typically refers to the process of compressing image/frame/video sequences into binary code, i.e., bitstreams to reduce storage and transmission needs [3]. The terms *lossy* or *lossless* refer to the reconstruction of the video if it is a perfect or imperfect one. For natural images or videos, most often lossy coding is used, where the optimization task lies mainly in two metrics: the compression efficiency, i.e., the size of the image or video, measured in the number of bits, where the smaller size, the shorter bitstream is more preferable; while the second aspect is the quality of the decoded frames, that is compared to the original image/video data.

The first effective and used video coding standard created was the H.261, which was developed by BT, Hitachi, NTT, PictureTel, and Toshiba, among others. Afterwards, various standards have been developed too [14]:

- H.262/MPEG-2 Part 2, that was used in DVDs and early digital televisions.
- H.263, which was used in low-bit-rate communication.
- H.264/AVC (Advanced Video Coding), which became the standard for many applications for mobile video and for high-definition streaming that are still the most common ones our days. It was introduced in 2003 [15].
- H.265/HEVC (High Efficiency Video Coding), which was developed after the H.264 standard, and it offers approximately double the compression efficiency.
- H.266/VVC (Versatile Video Coding), which was developed by the Joint Video Experts Team (JVET) [16] and Moving Picture Experts Group (MPEG) in 2020. It supports resolutions ranging from very low resolution up to 4K and 16K. VVC promises up to 50% greater compression efficiency compared to its predecessor, H.265/HEVC, without compromising on video quality [5, 17].

Video coding algorithms, in essence, work on the task of reducing redundancies present in video data, which can be spatial (within frames) or temporal (between frames). These algorithms are made up of different components that handle various aspects of the process, such as motion estimation, transformation, quantization, and entropy coding [14, 18].

2.1.1 Motion estimation and compensation

Through motion estimation and compensation, the algorithm analyzes the movement of objects in a video and uses that information to predict the next frame on the reference frames and the estimated motion vectors, which are compared to the actual frame, and only the differences between the two are transmitted or stored, also called *residuals*. This process reduces greatly the amount of transmitted data. In VVC, several new techniques are used to improve the process of motion estimation and compensation,

such as affine motion compensation, perspective affine motion compensation, and deep affine motion compensation network [19].

2.1.2 Transformation

In this step, spatial domain data is converted into the frequency domain to reveal insights about the image or video frame, most often using Discrete Cosine Transform (DCT). It works by converting spatial domain representation, where pixel values are correlated, into a set of uncorrelated coefficients that highlight the inherent patterns and structures within the video frame. By focusing on compressing the energy into fewer coefficients, DCT facilitates a significant reduction of data without substantially compromising the visual quality [20, 21].

2.1.3 Quantization

The quantization step reduces the precision of the transformed coefficients. This process, while inducing loss of information, significantly reduces the data size. By mapping a range of input values to a single output value, it simplifies the data representation and thus compresses the video's size [22]. Modern video coding standards implement adaptive quantization techniques that aim to improve subjective coding performance. These techniques consider temporal characteristics to produce a more visually friendly quantization parameter offset distribution [23].

The **quantization parameter (QP)** is the value that determines the level of quantization, and it controls the trade-off between the percentage of compression and the quality of the reconstructed video. A higher QP value means higher amounts of quantization and compression [24].

2.1.4 Entropy coding

In this step, the data is converted into the compressed bitstream, in a lossless manner, in which residual correlation in the coded data is exploited. Utilizing statistical coding techniques, it aims to assign shorter bit sequences to frequently occurring symbols and longer sequences to less frequent symbols, thereby minimizing the average code length and consequently the file size [25].

Several entropy coding algorithms are used in video coding, including Huffman coding [26], and Context-Based Adaptive Binary Arithmetic Coding (CABAC). CABAC was introduced in H.264/AVC, and is based on arithmetic coding [27], in which it encodes binary symbols, that allow probability modelling for frequent bits of any symbol. Selection of these probability models is done adaptively based on local conditions [22].

2.1.5 Loop filtering

Loop filtering is used to reduce artefacts introduced by block-based coding and quantization, and to smooth the transitions between blocks, enhancing the visual quality of the decompressed video [28] (more about loop filters in Subsection 2.2).

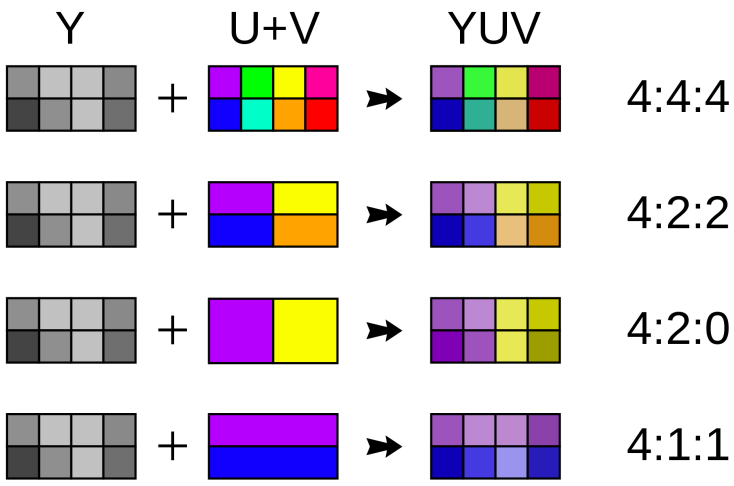


Figure 1: Colour subsampling visualization [31]

2.1.6 Colour subsampling

Colour subsampling is a technique which reduces the amount of data needed for the representation of the video sequence, without worsening the perceived visual quality in a major way. It exploits the human visual system's higher sensitivity to brightness (luminance), compared to colour (chrominance) information [29]. This can be done by converting from RGB colour space into YUV colour components, where Y denotes luminance (luma), and U and V: chrominance (chroma), where U is the blue projection, also noted as Cb, and V is the red projection, also noted as Cr [30]. Chroma components U and V may be reduced (subsampled) to half of the pixels in both horizontal and vertical directions (4:2:0), or only in horizontal direction (4:2:2). In the 4:2:0 setting, the data size is reduced by 50%, and in 4:2:2 by 30%. An example visualization of colour subsampling can be seen in Figure 1.

2.1.7 Temporal layers

Temporal layers enable adaptive bitrate streaming, where different layers can be selectively transmitted based on network conditions. They refer to the hierarchy of the video frames in a video sequence. They are organized in different layers based on their temporal distance from their key frame, which is an Intra-coded picture (I-frame). These I-frames are complete images like JPG or BMP image files, whereas the predicted images (P-frame) that are after the key frame only store the differences from the I-frame. For example, the encoder isn't required to store the background pixels in a P-frame if it is unchanged from the previous frame, so it can save space. A Bidirectional predicted picture (B-frame) can save even more space due to it referring to both preceding and following frames to determine its content [32, 33].

In Figure 2 we can see an example of how each type of frame is referring to the other ones. I-frames are called intra-frames, while P and B-frames are inter-frames. A group of pictures, also called GOP structure refers to the order in which intra- and

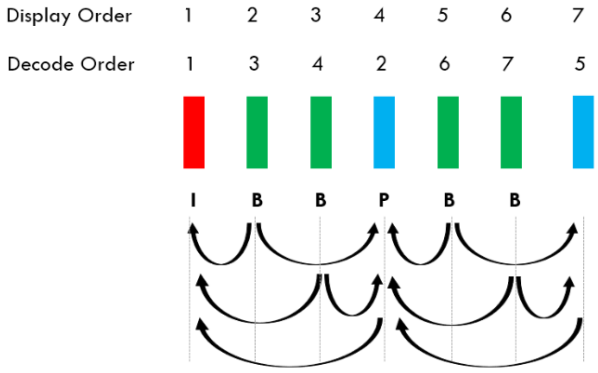


Figure 2: I, P, and B-frames in temporal layers example [33]

inter-frames are arranged. The encoded video streams consist of successive GOPs, which are independent of each other, i.e., encountering a new GOP in an encoded video stream allows one to decode it without access to other GOPs. Each GOP begins with an I-frame [4].

During *inter prediction*, the prediction of the frame is done from one or more frames that have been previously decoded. While in *intra prediction*, the decoding of the frame is based on the spatial redundancy found within the image. Frames that contain many smooth areas with the same colour or its gradients, can be decoded by predicting the pixel values based on the values of the surrounding pixels. Here, instead of relying on previously decoded frames, it relies on previously decoded parts of the same frame. This technique is similar to image coding [34, 35].

2.1.8 Random access

Random access in VVC allows us to consume the video content from any position instead of always at the start. Due to modern video coding standards mostly using inter-picture prediction as their main means to compress video images that rely on other frames, this option becomes complicated. To mitigate this, the encoded bitstream is constructed in a way that the order of decoding is assembled in the order in which the frames depend on each other, rather than the output order, i.e., the timely order in which the video is played [5].

2.1.9 Partitioning

Flexible partitioning schemes are an advancement in modern video coding algorithms, and it makes it possible for VVC to have a 50% higher efficiency over HEVC. A Coding Tree Unit (CTU) is a data structure that represents a region of a video frame, that the encoder is partitioning into smaller blocks or coding units (CU), which enables compression on a smaller scale instead of whole frames. This hierarchical structure allows the encoder to select adaptively the optimal block size based on the features of the region that is being encoded [36–38].

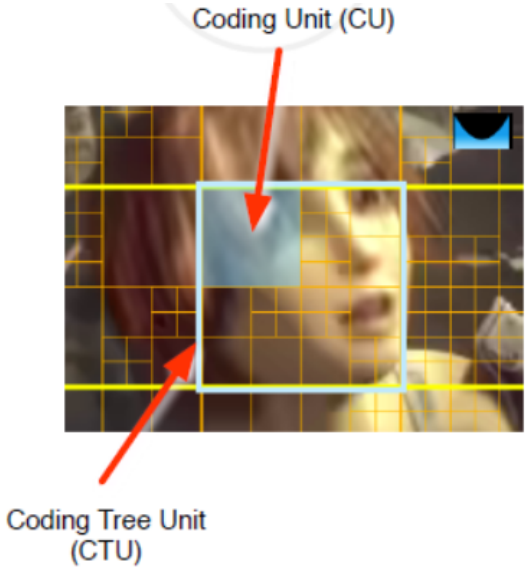


Figure 3: Frame partitioning into Coding Tree Units (CTU) and Coding Units (CU) example [39]

Each frame in the video sequence is split up into fixed-size blocks or CTUs, which are determined by the coding standard or settings. They can also be partitioned into square CUs, using a quadtree structure. Recursively, depending on the content of the CTU, these regions can be further split up, which gives different levels of complexity and detail [39]. An example of CTUs and CUs can be seen in Figure 3. These CUs are going to be the blocks/patches, or Prediction Units (PU) that, instead of the whole frame, are going to be encoded.

2.1.10 Supplemental enhancement information

Supplemental Enhancement Information (SEI) is considered as additional data that is sent alongside the main video stream and is stored in the encoded bitstream, adding a bit-rate overhead. SEI messages are features that were used from H.264/AVC onwards. These messages convey various types of information, including the video's display characteristics, content properties, or instructions for the decoding process [36, 40].

2.2 Loop filters

Loop filters are used on reconstructed video images in the HEVC and VVC standards, and they minimize block artefacts and noise that often occur after the quantization and transformation steps and also enhance the perceptual quality of the image before storing them. Multiple loop filters are used in video coding techniques. The in-loop filters are applied both during the encoding and decoding processes, before outputting the frames. HEVC uses two of these loop filters, the deblocking filter, and the sample adaptive offset, while VVC uses a few more on top of these, which are described in the

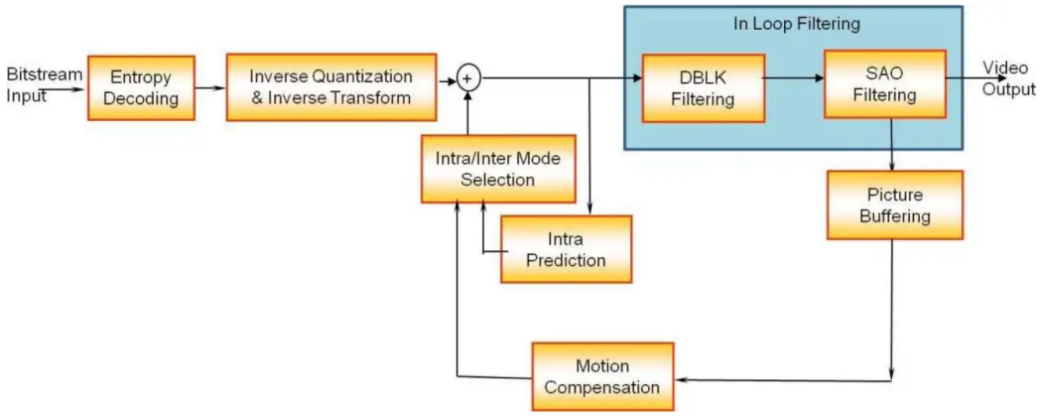


Figure 4: Structure of HEVC decoder, and placement of loop filters [41]

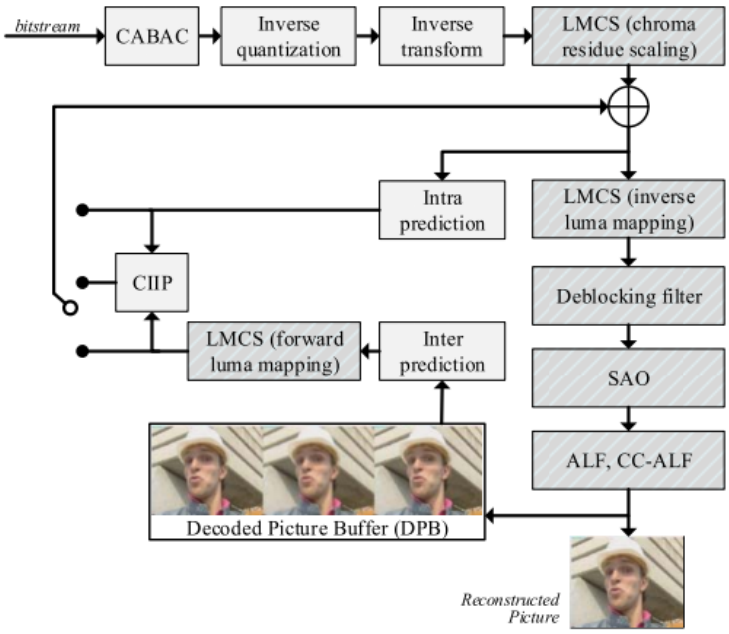


Figure 5: Simplified structure of VVC decoder, and placement of loop filters [28]

following subsections [4, 5]. In Figures 4 and 5 the structure of the decoder of HEVC and VVC can be seen, along with the placements of the in-loop filters.

2.2.1 Luma mapping with chroma scaling

The Luma Mapping with Chroma Scaling (LMCS) process alters the input signal's dynamic range by applying a luma inverse mapping function on the video frames, at the start of the loop filters. This function is used on reference images, while, in the same way, a forward mapping function is used by the encoder on the inter-picture prediction signal. Furthermore, based on the luma mapping, chroma residual scaling adjusts the

chroma signal, in order to balance the bitrate of luma and chroma components. Using a piecewise linear model, the LMCS's luma mapping process is implemented at the pixel sample level. Meanwhile, the chroma scaling process is carried out at the chroma block level, utilizing a scaling factor obtained from the reconstructed neighbouring luma samples of that chroma block. [5, 42].

2.2.2 Deblocking filter

The Deblocking filter (DBLK) removes the block-based compression artefacts which are mostly shown as blocky noise. The filter operates at block edges, smoothing the transition between adjacent blocks. It functions by identifying and reducing sharp intensity transitions from one block to another, which occur most often after the quantization step [43].

2.2.3 Luma-adaptive deblocking

The Luma-Adaptive Deblocking tool adjusts the strength of the deblocking filter based on the frame's mean luma level. In the Sequence Parameter Set (SPS), at most four luma level threshold values are signalled with their associated offset values for intensified deblocking. This method is particularly beneficial for High Dynamic Range (HDR) content, which possesses different non-linear transfer characteristics compared to Standard Dynamic Range (SDR) video. When displayed, a corresponding non-linear process transforms the decoded video signal into linear light, making distortions from quantization more noticeable in areas with high or low brightness. By allowing for stronger deblocking, the luma-adaptive deblocking effectively mitigates visible distortions in these areas [5, 44–46].

2.2.4 Sample adaptive offset

Sample Adaptive Offset (SAO) is a nonlinear filtering tool, that additionally refines the reconstructed signal, and it enhances the signal representation in smooth regions and near the edges. It was designed to reduce banding and ringing artefacts. It applies an offset to the pixel values in the decoded picture, that are calculated based on edge direction or shape and pixel level [4, 41].

2.2.5 Adaptive loop filter

Adaptive Loop Filter (ALF) is applied to enhance the reconstructed video signal through spatial filtering. It operates differently for luma and chroma components, utilizing a 7×7 diamond-shaped region for luma and a 5×5 region for chroma. This filter applies non-linear clipping on the differences between a current sample and its neighbouring samples, facilitating the encoding process to consider the value similarity between them by choosing suitable clipping parameters that get signalled as well. On the decoding side, a local classification of a 4x4 block is used to choose the luma filter. This classification, which divides the block into 25 categories, is based on the directionality and the 2D Laplacian activity. This process, which signals an index

for one of the 25 luma filters and one of the 8 chroma filters at the Coding Tree Unit (CTU) level, allows for significant local adaptivity. The encoder establishes the filter coefficients and clipping parameters and can signal various sets for every encoded video through an Adaptive Parameter Set (APS) [5, 47].

2.2.6 Cross-component adaptive loop filter

Cross-Component ALF (CC-ALF) exploits correlations between the luma and chroma channels, where it uses the luma channel to identify the chroma residuals which are added to an earlier reconstructed chroma signal. It applies a 3x4 diamond-shaped high pass filter for every chroma component to the luma input samples of ALF. For the chroma components, the encoder defines up to four sets of CC-ALF filter coefficients. This enhances local adaptivity by signalling one of the four filter sets for each component at the Coding Tree Unit (CTU) level. The ALF parameters and the CC-ALF filter sets are jointly utilized in an APS [5, 47, 48].

2.3 Neural network-based video coding

Techniques powered by Artificial Intelligence (AI) have demonstrated significant promise in enhancing video coding efficiency, aiming to optimize the Quality of Experience (QoE) for end-users within a constrained bit rate allocation. Traditional video coding algorithms like AVC, and HEVC, rely on different processing steps such as prediction, transformation, and quantization, while neural network-based video coding uses the feature representation abilities of deep neural networks to enhance various stages of the video coding pipeline. For instance, neural networks (NN) can be used in the reconstruction stage to enhance the quality of the video frames, by reducing the artefacts introduced in the compression phase. These advancements are done with, next to other architectures, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), which are capable of capturing spatial and temporal dependencies in video data. Motion compensation is one example where Machine Learning (ML) architectures based on deep CNNs can improve greatly. However, a downside to it is that it increases the runtime and memory requirements needed to compress them, making them inadequate for real-time deployment. It is a currently researched topic, where we can expect to have breakthroughs with the help of AI [3, 49–54].

2.3.1 Overfitting

In Machine Learning (ML), overfitting is an undesirable occurrence when a model excels on training data but struggles with new, unseen data [55]. However, in image and video coding, intentionally overfitting a model during inference (or compression simulation) for specific media elements within an image or video is actually a beneficial strategy. This approach enhances the efficiency of codecs that rely on learning for image and video processing [56–59].

A decoder-side neural network model overfitting technique was introduced in [57] This method is applied to a neural network-based post-processing filter for the Versatile Video Coding (VVC) codec. It involves compressing weight updates by encouraging sparsity and lower absolute values during the overfitting of the pre-trained model on the image or video frame. Despite the initial method resulting in a significant overhead due to the large number of parameters in neural networks, authors of [58] suggest focusing on overfitting only the bias parameters of convolutional layers. This approach, which impacts a smaller portion of the parameters but has a substantial effect on performance, significantly reduces weight update overhead and achieves a notable Bjøntegaard Delta rate (BD-rate) gain [60].

2.3.2 Neural network-based in-loop filter

Integrating CNN-based filters into the compression loop presents challenges as the filtered image influences other coding tools. Various studies have explored different approaches to this integration. Park and Kim [61] developed a CNN in-loop filter with 3-layers for HEVC, with specific models for different QP spectrums and conditions for application based on quality improvement or Picture Order Count (POC). Meng *et al.* [62] introduced an LSTM-based in-loop filter that is applied before the SAO in HEVC, utilizing both Multiscale Structural Similarity Index (MSSIM) and Mean Absolute Error (MAE) loss during training. Zhang *et al.* [63] proposed a residual highway CNN (RHCNN) as an in-loop filter, with distinct models for I-, P-, and B-frames and separate training for various QP ranges. Dai et al. and Jia *et al.* [7] both developed deep CNNs as in-loop filters in HEVC, incorporating different strategies for I and P/B-frames and utilizing CTU- and CU-level controls, with the latter also introducing a content analysis network to optimize CNN model selection. These approaches signify the ongoing efforts to optimize CNN-based in-loop filtering in video coding [3].

Utilizing neural networks facilitates the creation of adaptive filters capable of dynamically adjusting their parameters during the encoding process, to optimize filtering operations based on the intrinsic particularities of the video data. The existing in-loop filters in VVC, namely deblocking (DBLK), sample adaptive offset (SAO), and adaptive loop filter (ALF), have been found to be insufficient in handling complex compression artefacts. To combat this problem, NNs have been used to remove compression artefacts either as in-loop filters [7, 8, 64–70] or post-filters [9, 11, 57, 58, 71–77]. Most of these frameworks use a single offline trained neural network which is used as a filter, or they propose a set of NNs from which one the best suited is used based on, e.g., rate-distortion optimization (RDO) [10].

Content adaptation has been achieved by overfitting neural networks, and in many cases, they resulted in higher coding efficiency [11, 57, 58, 76, 78, 79]. In these techniques, a pretrained NN-based in-loop, or post-processing filter is trained at the encoder side to overfit on the desired video data. In the supplemental enhancement information (SEI) message the weight update, i.e., the difference between the pretrained NN's weights and the overfitted NN's weights, is signalled next to the video bitstream to the decoder side, where the overfitted NN is reconstructed with the help of this

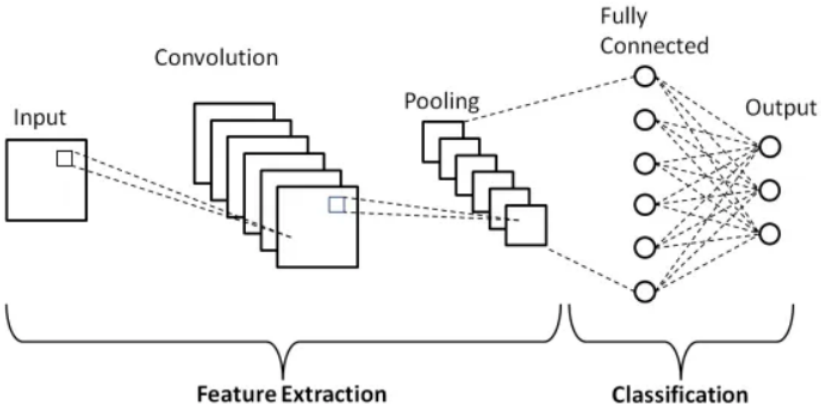


Figure 6: Convolutional Neural Network (CNN) outline [83]

weight update. With this restored NN the filtering can be done at decoding time. To have the least amount of information sent along the bitstream, only the biases are overfitted instead of all the NN's weights. Santamaría *et al.* [10] introduced a new learnable parameters inside the CNN filter, called *multipliers*, that are overfitted instead of the biases. This multiplier is a vector that multiplies the feature tensor output by a convolutional layer so that each element in the vector multiplies a single channel of a feature tensor.

These NN-based in-loop filters can be placed in many positions next to the other in-loop filters (DBLK, SAO, ALF). Common positioning schemes include placing them before the Deblocking filter (DBLK), or between the DBLK and the Sample Adaptive Offset (SAO), or by replacing any of these two. It is also common practice in modern algorithms, to use it in parallel with either the DBLK or the SAO, in the way that the datastream goes through both the deblocking filter and the trained NN filter, and their outputs are combined.

2.3.3 Convolutional neural networks

Convolutional Neural Networks (CNN) are a subtype of artificial neural networks that are categorized in the classification problems in Machine Learning (ML). They are extensively used on image- and matrix-type data, such as videos, in the scopes of object detection, image and video recognition, recommender systems, and image generation [80, 81].

They are a type of feed-forward neural network, that learns feature engineering through the optimization of filters or kernels. CNNs primarily consist of three layers: convolutional, pooling, and fully connected layers. Each layer transforms the input data with the aim of gradually extracting higher-level features. In the initial layers, the model identifies basic attributes like colours and edges. As it advances to deeper layers, it discerns more intricate patterns or shapes [82]. In Figure 6 the outline of the CNN's structure is shown.

- **Convolutional layer** is an essential building block of a CNN, which creates a feature map by applying a filter or kernel to the input data. This kernel is a matrix

that slides over the input datapoint, performing element-wise multiplication and summation to produce a single value out of the feature map. This process is repeated on the entire input data to create a complete feature map. Multiple feature maps are created, where each focuses on different features of the image.

- **Pooling layer** is applied to reduce the spatial dimensions of the data while maintaining useful information. It helps to decrease the computational complexity and to avoid overfitting. The most common types of pooling are max pooling, and average pooling, where either the maximum or average value is aggregated from non-overlapping filters on the data.
- **Fully connected layer** is used to generate predictions from the features that have been extracted. It takes the output from the last layer and flattens it into a one-dimensional vector, which is then fed into a standard feed-forward NN to make the prediction.

As for the activation function, Rectified Linear Unit (ReLU) is often used which is linear for positive values, but 0 for negative ones. Leaky ReLU is based on ReLU, but the difference lies in the negative values where the slope is a non-zero small slope, allowing small non-zero gradients through the layer [84]. The mathematical formulation for Leaky ReLU is given by:

$$\sigma = \begin{cases} x & \text{if } x \geq 0 \\ a \cdot x & \text{otherwise} \end{cases} \tag{1}$$

where a is the slope coefficient that is determined before training.

2.3.4 Neural network coding and representation

Neural Network Coding and Representation (NNR) was developed by the ISO/IEC Moving Picture Experts Group (MPEG) [85] as the first international standard for efficient neural network (NN) compression. It is created as a toolbox of compression methods, that can function as a standalone coding framework with its bitstream format or can be used with other neural network formats and frameworks. Offering maximum flexibility, it operates on a per-parameter tensor basis, ensuring accurate decoding even without structural information. Incorporating advanced encoding and decoding technologies like deep context-adaptive binary arithmetic coding (DeepCABAC), it also facilitates neural network parameter optimization through various methods like sparsification, pruning, low-rank decomposition, unification, local scaling and batch norm folding. Notably, NNR maintains over 97% compression efficiency without affecting classification quality in transparent coding scenarios [86]. In Figure 7 the overview of the NNR is presented.

2.3.5 Neural network quantization

Neural networks are mostly trained with high-precision numerical formats like 32-bit floating-point (float32), although, relying on floating-point arithmetic can lead to dis-

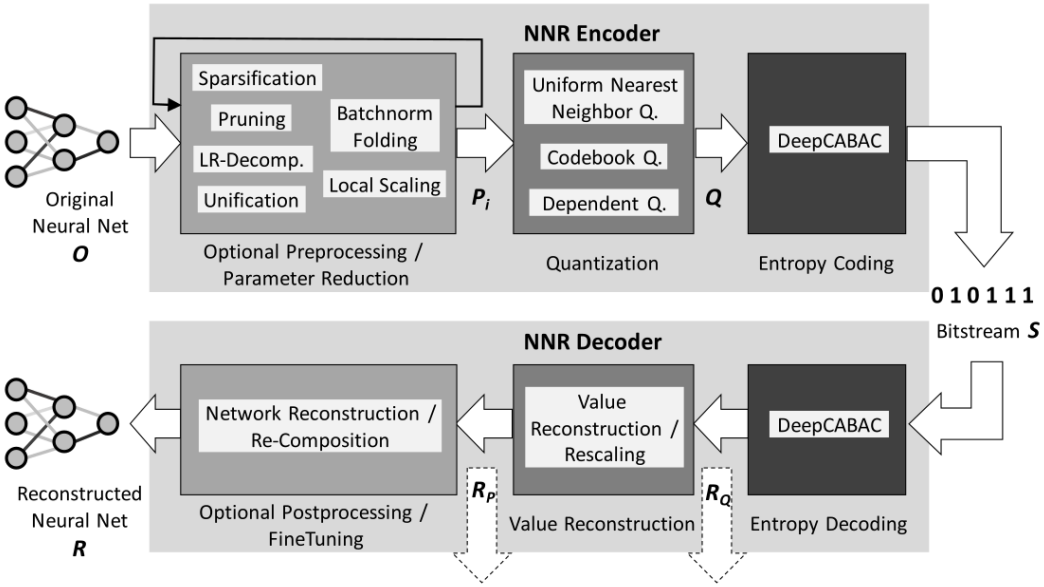


Figure 7: Neural Network Coding and Representation (NNR) overview [86]

crepancies across different platforms, potentially resulting in corrupted reconstructions during the decoding process due to data mismatch between the encoder and decoder. Yang *et al.* [87] suggest that one way to mitigate this issue is by representing the network parameters with fixed-point precision, which not only addresses the mismatch problem but also offers benefits such as reduced power and memory usage, decreased network delay, and smaller costs for individual Multiply-ACcumulate (MAC) operations. The authors suggest using quantisation from float32 to 32-bit or 16-bit fixed-point values (int32, or int16).

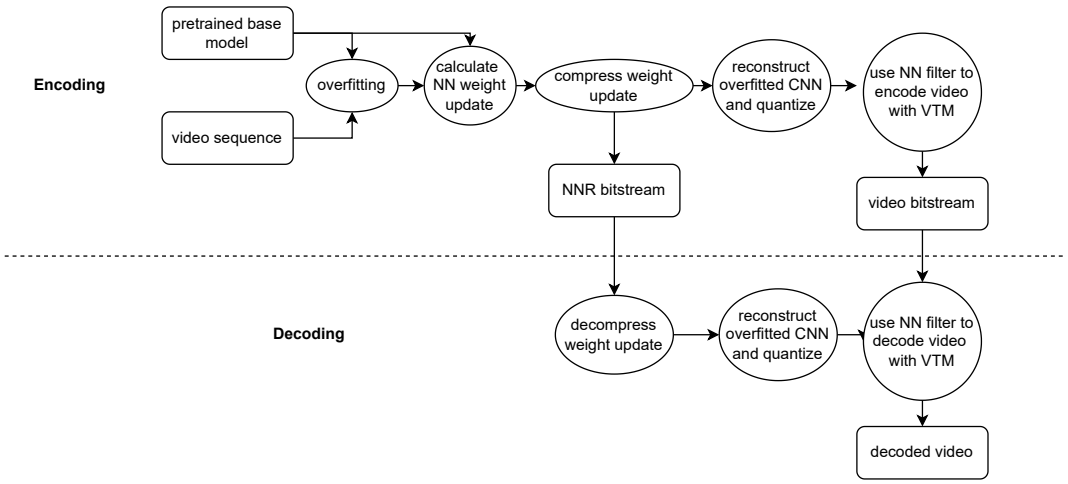


Figure 8: Overfitting pipline

3 Methodology

3.1 Overview

The goal of the research is to answer how well does two-stage overfitting of the NN-based in-loop filter performs in video coding efficiency, where in the second stage, the model overfits on only patches that give a positive Peak Signal-to-Noise Ratio-gain. To test it, first, the hyperparameters were searched to obtain decent results without two-stage overfitting.

In the first part of the hyperparameter tuning, an optimal learning rate was searched. Afterwards, experiments were done on the loss functions, different epoch counts, and loss weighting for colour components. Subsequently, during two-stage overfitting, two models were tested: one that started overfitting from the pretrained model, and another that continued the overfitting from the first-stage model that filtered out the patches that gained negative delta-PSNR (dPSNR).

The overfitting was done on the multipliers inside the CNN in-loop filter. Loop filter models were created by overfitting on each test sequence, and for multiple quantization parameters (QP) $(22, 27, 32, 37, 42)$, giving us $4 \cdot 5 = 20$ models per experiment. In one experiment the models' performances are aggregated into one set of metric values.

Four different base in-loop filter models were trained on a general video dataset (BVI-DVC), while the overfitting was performed on one of the base models on the test sequences at the encoder side. Here, the resulting weight update of the model is compressed and signalled with the encoded bitstream, which is then decoded at the decoder side to reconstruct the overfitted loop filter.

3.1.1 Pipeline

A simplified encoding and decoding pipeline is shown in Figure 8.

The process of overfitting the loop filter, and encoding the video sequence is described in the following:

1. One pretrained base NN filter model is selected (out of 4 existing ones) to be overfitted based on which model gives the highest PSNR on the given video sequence.
2. The model is overfitted on one video sequence with one QP.
3. The weight update of the neural network (NN) is calculated as: *overfitted parameters - pretrained parameters*.
4. The weight update is compressed using the NNR standard.
5. The decompressed weight update is used to reconstruct the overfitted NN filter.
6. The NN is quantized to fixed point precision (int).
7. The overfitted NN filter is integrated into the VVC Test Model (VTM) that simulates the decoding.
8. The video sequence that the model overfitted on, gets encoded into a bitstream, including the weight update of the NN. The VTM uses the VVC standard to encode but uses the overfitted loop filter in its in-loop filtering process (if it enhances the quality, thus improving the PSNR).

After the video and NNR bitstream are created, the decoding can take place as follows:

1. The encoded NNR bitstream is decoded.
2. Using the weight update, the overfitted NN loop filter is reconstructed.
3. The network is quantized.
4. Integration into VTM.
5. The video bitstream is decoded using the NN in-loop filter, where the position of the overfitted filter is in parallel with the Deblocking Filter (DBLK), and both their outputs are combined.

After the decoding process, the final metrics and BD-rates are obtained.

The process of two-stage overfitting is shown on Figure 9, and listed here:

1. Overfit a model like before, on every patch from the dataset. This will be the first-stage model.
2. Based on this model, compute the PSNR values over the patches of the video sequence. Compare this value to the PSNR of the unfiltered data that the CNN learns from.

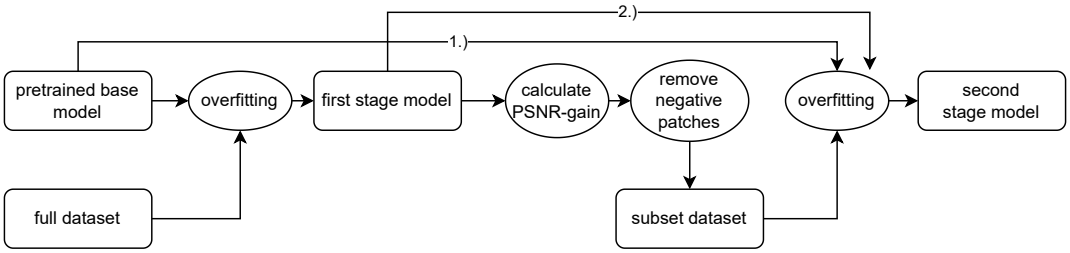


Figure 9: Two-stage overfitting pipline

3. Remove those patches from the dataset that are getting negative delta-PSNR (dPSNR), i.e., $\text{PSNR}_{\text{first-stage model}} - \text{PSNR}_{\text{unfiltered data}} < 0$, meaning that the overfitted first-stage NN didn't improve the quality of the video patches during 200 epochs.

4. Overfit a model over this subset of the dataset. This overfitting can be started 1.) from the base model, as usual, or 2.) from the first-stage model, so that it has already learnt from the video sequence, but now it has to focus on only a subset of all the patches from the video data.

3.2 Common test conditions

The assessment is conducted utilizing the VVC Test Model (VTM) [88], the benchmark software for the H.266/Versatile Video Coding (VVC) standard. This research utilized version 11.0 NNVC 5.0 [89], a tool that has been used by the standardization specialists, the Joint Video Experts Team (JVET) [16], to test and assess NN tools in the context of hybrid video coding. The codec setup used is the JVET Random Access (RA) Common Test Conditions (CTC) for NN-based Video Coding (NNVC) [90], in the RA configuration, both intra-frame and inter-frame predictions are used. This software and the corresponding configurations are used to produce the reconstructed data and additional supplementary details necessary for the creation of training datasets.

From the CTC test sequences, four sequences from class D were considered during inference time in this research, with the attributes shown in Table 2. Results were compared to the anchor model's results, that is the NNVC 5.0. These test sequences were in YUV 4:2:0 format instead of RGB format, facilitating the possibility of calculating PSNR on the luma and chroma channels, and also tweaking the weighting of colour distributions of Y (luma), Cb (blue-difference chroma) and Cr (red-difference chroma).

3.3 Pretraining

The base in-loop filter CNNs were pretrained on the BVI-DVC dataset which is an extensive and representative video database, designed to train CNN-based video compression systems. This database is particularly focused on facilitating the development of machine learning tools that improve traditional compression frameworks. This

Sequence name	Frame count	Resolution	Frame rate	Bit depth
RaceHorses	300	416x240	30	8
BQSquare	600	416x240	60	8
BlowingBubbles	500	416x240	50	8
BasketballPass	500	416x240	50	8

Table 2: Test sequences in class D from the JVET Common Test Conditions

includes aspects such as upscaling of spatial resolution and bit depth, post-processing, and in-loop filtering. The BVI-DVC incorporates 800 sequences with spatial resolutions ranging from 270p up to 2160p, including video content of a wide array of natural scenes and objects, featuring diverse textures and motion types, and various camera movements and activities involving humans, animals, and other objects [91, 92].

There are 4 in-loop filter base models that were pretrained on the BVI-DVC dataset. Each of these base models is used in the decoding process, but only one is used for one frame or block. The loop-filter model used is decided based on Rate-Distortion Optimization (RDO). One of these base models is overfitted and used instead of the original base model. The model selection is based on the highest PSNR gain on the whole sequence.

3.4 Overfitting

3.4.1 Preprocessing

As a preprocessing step, the YUV 4:2:0 format images of size 144x144 were converted into 6 blocks/patches of size 72x72. They were also normalized into the range of $[0, 1]$. These patches from the frames are then randomized and put into batches of size 64. If it was needed, the patches were padded, in the same way as the VTM does it.

The bit-depth of the video sequences shall be 10-bit, as restricted by the CTC [90], and for this reason, the 8-bit sequences are converted into 10-bit.

Each video sequence has been coded with VTM 11.0, RA configuration, with different Quantization Parameters (QPs) of 22, 27, 32, 37, and 42.

3.4.2 Architecture

The content-adaptive in-loop filter is a CNN with 46 layers, based on [67] and [93]. The network architecture is shown in Figure 10, with the difference that the multiplier layers are added after each convolutional block. The output of a multiplier layer is defined as:

$$\mathbf{y} = \sigma\left((\mathbf{K} * \mathbf{x} + \mathbf{b}) \cdot \mathbf{m}\right) \tag{2}$$

where $\mathbf{K}$ is the kernel, $\mathbf{x}$ is the input to the layer, $*$ is the convolution operator, $\mathbf{b}$ is the bias, $\mathbf{m}$ is the multiplier, and σ is the activation function [10, 87].

The CNN is made up of 13 filter blocks, out of which 11 filter blocks are used as hidden layers. These hidden blocks consist of two 1x1 convolution layers with a Leaky ReLU activation layer between them, and a 3x3 convolution layer, where $M =$

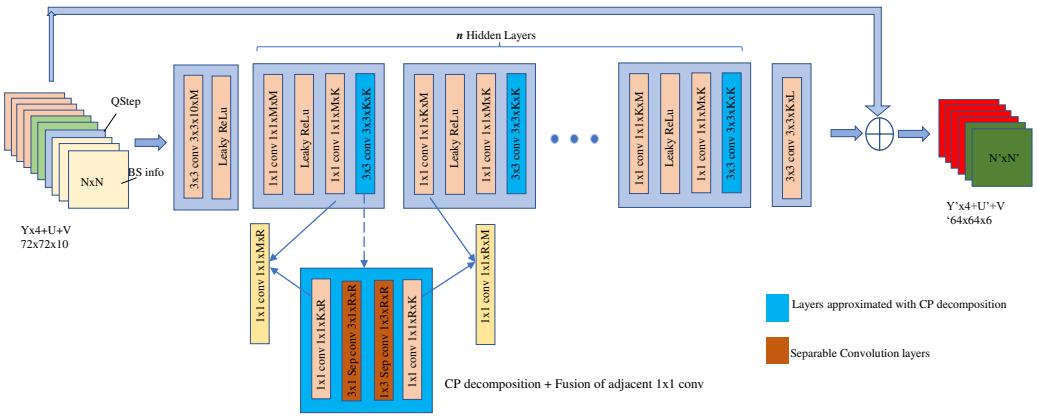


Figure 10: CNN loop filter architecture [79]

72, $K = 24$, $L = 6$, and rank $R = 24$, such that $M > K$. CANDECOMP/PARAFAC decomposition (CP) [94] blocks are added to each hidden layer that decomposes the 3x3 convolutional layer, and fuse the adjacent 1x1 pointwise convolutions. These 3x3 convolutions are decomposed into 4 layers with rank R followed by fusion of adjacent 1x1 convolution:

- 1st layer: 1x1xKxR pointwise convolution
- 2nd layer: 3x1xRxR separable convolution
- 3rd layer: 1x3xRxR separable convolution
- 4th layer: 1x1xRxK pointwise convolution

The input to the CNN loop filter are:

- the reconstruction Coding Tree Unit (CTU) and 8 neighbouring samples to each side of the CTU, which is derived from video compression following the VVC standard, i.e., the original video is first fed through the VVC Test Model (VTM) and the network will learn to enhance the quality of this reconstruction, to get closer to the original video
- 4 luma (Y) and 2 chroma (U, V) tensors with size 72x72. The luma samples from the patch are interleaved into four blocks, as shown in Figure 11
- frame-level normalized Quantization Parameter (QP) Step
- Boundary Strength (BS), which is a map that contains the boundaries/edges of a frame relative to the CTU partitioning

The output is a filtered CTU with size 64x64x6, including 4 luma channels that were interleaved as at the input, and 2 chroma channels [67, 78, 79, 93].

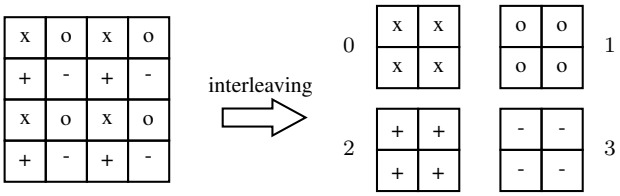


Figure 11: Interleaving the luma samples into four blocks [10]

3.4.3 Process

The encoder will try to overfit the pretrained base CNN loop filter on the particular [video sequence, QP] pair by optimizing over the loss function which is the weighted Mean Squared Error (MSE). The weights used for the loss function were 4:1:1, and 12:1:1 for the Y:Cb:Cr separate losses. ADAM [95] is used as a stochastic optimizer, with a fixed learning rate that is set and tuned as a hyperparameter. The overfitting is done in 100, 200, or 400 epochs, which each consist of batches of size 64. Experiments were done on the number of epochs as well, thus the varying numbers. The specifications of the overfitting are shown in Table 3.

Framework	TensorFlow 2.8.0 [96]
Epoch	100/200/400
Batch size	64
Loss function	Weighted-MSE/MAE/SSIM
Colour weights for loss function	4:1:1/12:1:1
Patch size	72x72
Learning rate	1e-4/1e-3/1e-2
Learning rate update strategy	Constant learning rate
Optimizer	ADAM
Total Convolutional Layers	46
Total Fully Connected Layers	0
Number of Parameters	53724
Number of Trainable Parameters	1158
Quantization parameters (QPs)	22, 27, 32, 37, 42

Table 3: Overfitting specifications and technicalities

3.4.4 Two-stage overfitting

After the best hyperparameters were obtained, we used the model with those hyperparameters as a first-stage model. On the whole dataset of patches taken from the video sequence, the PSNR metric is calculated between the model's reconstruction (output of video sequence filtered with the model) and the ground truth (the original video). This PSNR is compared with the PSNR of the unfiltered data, i.e., the video reconstruction from the VTM (original video fed through the VVC coding), which

serves as the input data to the loop filter neural network. Throughout the overfitting the goal is to obtain a gain over the unfiltered data, i.e., to obtain a positive PSNR-gain (dPSNR). This goal is achieved, however not on all patches of the video data, but some patches are left with negative dPSNR, i.e., the model is performing worse in those instances. The purpose of the two-stage overfitting is to remove those patches that have negative dPSNR from the dataset and to overfit the model on only a subset of the video data. If some patches show no gain, it means the model can't learn the characteristics of them, thus by removing them, it is believed that it will boost the performance on the patches that have already shown that the model is capable of obtaining positive dPSNR.

Two types of two-stage overfitting were experimented on, depending on where the overfitting starts from in the second-stage model:

1.) base model. The second-stage model is overfitted like the other models, but on the subset of the dataset, that has positive PSNR-gain from the first-stage model.

2.) first-stage model. The first-stage model continues the overfitting, but on the subset of the dataset, instead of the whole video sequence, like before.

The second-stage models are further overfitted for 200 epochs. The model at point 2.) has been overfitted on the whole dataset for 200 epochs, then on the subset for another 200 epochs, totalling 400 epochs. To compare the results, this model has to be compared to another model that is trained for 400 epochs on the whole dataset. The model at point 1.) can be compared with the first-stage model which has been overfitted for 200 epochs.

Since these models are optimized for only a subset of the video data, not for all the patches in the sequence, the rest of the sequence will yield worse performance. However, due to the VTM's ability to choose whether it uses the NN in-loop filter or not, for those patches that have negative dPSNR, it will either way choose to not use the overfitted model. Thus, only optimizing for the positive dPSNR patches should yield higher BD-rates overall.

3.5 Inference

To obtain the final results and measurements, the compression pipeline has to be simulated. The simulation, also called inference, was done with the help of the Small Ad-hoc Deep-Learning Library (SADL) and the VVC Test Model software (VTM), which performed the encoding of the CTC video sequences with the compressed NN loop filter model, and then the decoding as well.

Before the encoding happens, the coded weight update is decoded and used to recreate the overfitted loop filter by adding the multiplier update to the pretrained CNN filter. This NN filter is integrated into the VTM 11.0 with NNVC 5.0, which, during encoding, can choose to use the overfitted NN loop filter on a CTU partition, or not. This decision is based on Rate-Distortion Optimisation (RDO), where also the bitrate overhead created by the NN model is considered. For this reason, there exists a flag that determines (i) if the filter is not used on all the CTUs within a frame, (ii) if

the filter is used for all CTUs, or (iii) if the filter has been used partially on the frame CTUs, in which case another CTU-level flag signals if the NN is used on the particular CTU or not [11, 76]. After the encoding, the decoding can take place, using the VTM, and the final BD-rates and other metrics can be calculated, and compared against the performance of VTM 11.0 NNVC 5.0 in RA configuration (random access).

SADL was used with int16 precision, meaning that the NN model's parameters and activations were quantized to int16, instead of float32, as in [87].

SADL (Small Ad-hoc Deep-Learning Library) is a C++ source code library designed for neural network inference (compression simulation), which was created to provide a simple framework, that is compatible with JVET software development policy. Furthermore, it can be integrated with VTM and it can function without any dependencies [97].

3.6 Metrics

During the overfitting of the filter models, Mean Squared Error (MSE), and Peak Signal-to-Noise Ratio (PSNR) loss functions were used. Also experimented with Mean Absolute Error (MAE), and Structural Similarity Index (SSIM) loss functions. Traditionally, MSE and PSNR have been the most commonly used loss functions in video coding due to their simplicity and ease of computation. However, they often do not align well with human perception of quality, as well as SSIM and Multiscale SSIM (MSSIM).

The quality was assessed using the PSNR and MSSIM metrics, and also the Bjøntegaard-Delta rate (BD-rate) was calculated for each metric. The bitrate of the weight update, which was transmitted outside of the video bitstream, but was still included in the BD-rate calculation. Thus this metric takes into account both visual quality and compression efficiency (bitrate). It is calculated for the luma, and the two chroma components separately [90].

The bitrate of the weight-update r is calculated with the following equation:

$$r = \frac{b \cdot f}{1000 \cdot n} \tag{3}$$

where b is the bits of the weight-update, f is the frame rate of the video sequence measured in frames per second (fps), and n is the number of frames in the video sequence [76].

Bjøntegaard Delta-rate

3.6.1 Peak signal-to-noise ratio

Mean Squared Error (MSE) is a widely used loss function that calculates the average of the squares of the differences between the actual and predicted values [98, 99]. The formula of MSE is as follows:

$$\frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2 \tag{4}$$

Peak Signal-to-Noise Ratio (PSNR) measures the quality of reconstructed images or video frames compared to the original ground-truth, where the with the quality increase the PSNR value also increases, i.e., a lesser amount of distortion or noise introduced during the compression or reconstruction process. PSNR denotes the proportion between the upper boundary of a signal and the power of the interfering noise that impacts its representation quality. Given that many signals possess a wide dynamic range (the disparity between the maximum and minimum potential values of a variable quantity), the PSNR is typically expressed in the logarithmic decibel scale [100]. The PSNR is calculated using the Mean Squared Error (MSE) between the original and the compressed or reconstructed image/video. The formula for calculating PSNR is given by:

$$\begin{aligned} \text{PSNR} &= 10 \cdot \log_{10} \left(\frac{\text{MAX}^2}{\text{MSE}} \right) \\ &= 20 \cdot \log_{10} \left(\frac{\text{MAX}}{\sqrt{\text{MSE}}} \right) \end{aligned} \tag{5}$$

where MAX is the maximum possible pixel value of the image. For an 8-bit image, it is 255 (however, since the data is normalized, the maximum value is 1), and MSE is the mean squared error between the original and the compressed/reconstructed image.

During the overfitting, the PSNR value of one loop filter model is compared to the PSNR of the unfiltered data that comes from the VTM's reconstruction that serves as input to the CNN, thus trying to obtain a PSNR-gain over the VTM. This comparison is done under the metric of delta-PSNR (dPSNR), where the higher dPSNR value is preferred, showing that the overfitted model has gained a higher PSNR.

During overfitting, a weighted loss is used, where MSE and PSNR are calculated over the luma (Y), and two chroma channels (Cb, Cr). In the loss function, these three measures are added together with their weights (w), like the following:

$$\text{MSE}_{\text{YCbCr}} = \text{MSE}_{\text{Y}} \cdot w_{\text{Y}} + \text{MSE}_{\text{Cb}} \cdot w_{\text{Cb}} + \text{MSE}_{\text{Cr}} \cdot w_{\text{Cr}} \tag{6}$$

$$\text{PSNR}_{\text{YCbCr}} = \text{PSNR}_{\text{Y}} \cdot w_{\text{Y}} + \text{PSNR}_{\text{Cb}} \cdot w_{\text{Cb}} + \text{PSNR}_{\text{Cr}} \cdot w_{\text{Cr}} \tag{7}$$

Higher weights are given to the luma component due to humans' higher sensitivity to luma than to chroma. In the experiments the following weights were tested for Y:Cb:Cr: 4:1:1, 4:2:2, and 12:1:1.

3.6.2 Structural similarity index

The Structural Similarity Index (SSIM) was developed to better assess the visual quality of images and videos as perceived by humans, as the human visual system is good at looking for and extracting structural information from scenes. In this metric, changes in structural information of images and videos are considered, such as luminance and contrast. Its value ranges from -1 and 1, where the higher values indicate similarity and lower values indicate major differences in terms of structural

information, luminance, and contrast [98, 101]. The formula of SSIM is given as in the following:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{\left(2\mu_{\mathbf{x}}\mu_{\mathbf{y}} + C_1\right)\left(2\sigma_{\mathbf{xy}} + C_2\right)}{\left(\mu_{\mathbf{x}}^2 + \mu_{\mathbf{y}}^2 + C1\right)\left(\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2 + C_2\right)} \tag{8}$$

where $\mu_{\mathbf{x}}$, $\sigma_{\mathbf{x}}^2$ and $\sigma_{\mathbf{xy}}$ is the mean of $\mathbf{x}$, the variance of $\mathbf{x}$, and the covariance of $\mathbf{x}$ and $\mathbf{y}$, respectively.

Multiscale SSIM (MSSIM) was developed to incorporate the variations of viewing conditions, compared to traditional single-scale structural similarity methods.

3.6.3 Bjøntegaard-delta rate

The Bjøntegaard Delta rate (BD-rate) is a method for calculating the average difference between two rate-distortion (RD) curves. It was introduced in 2001 by Gisle Bjøntegaard. The BD-rate measures the percentage of bitrate reduction provided by a codec provides while preserving the same quality, as determined by objective metrics (usually measured in terms of PSNR or SSIM). It has become a standard metric in the video coding community to compare the efficiency of different video compression algorithms. A negative BD-rate indicates that the first codec is more efficient (i.e., requires fewer bits to achieve the same quality), while a positive BD-rate indicates that the second codec is more efficient. This also means that the order of codecs is important in the calculations. To calculate the BD-rate the curves have to be fitted to the rate-distortion points and the area between these curves is integrated [12, 24].

4 Results

The results are divided into two sections. At first, optimal hyperparameters were searched, in terms of the learning rate, loss function, colour weighting for loss calculation (Y:U:V; in what portion does each channel's loss metric contribute to the weighted loss), epoch number (number of times to iterate over all the batches, i.e., over the dataset of video patches). In the second section, experiments were done on two-stage overfitting. The first-stage model in the two-stage overfitting comes from the best model attained in the hyperparameter tuning.

4.1 Hyperparameter tuning

The results in Table 4 show experiments on Learning Rate (LR), colour weighting for the loss function, and epoch numbers. The metric value is the Bjøntegaard-Delta rate (BD-rate), which is calculated with regard to the Peak Signal-to-Noise Ratio (PSNR) for the luma (Y), and chroma (U, V) components separately. When regarding the BD-rates, the lower values are preferred, showing a gain over the anchor model: NNVC 5.0.

LR	Colour weight	Epochs	Y-PSNR	U-PSNR	V-PSNR
0.01	4:1:1	100	-0.63%	-8.28%	-7.69%
0.001	4:1:1	100	-0.88%	-7.71%	-6.95%
0.0001	4:1:1	100	-0.94%	-5.86%	-5.52%
0.0001	12:1:1	100	-0.96%	-3.50%	-3.21%
0.0001	12:1:1	200	-1.01%	-4.28%	-3.61%

Table 4: BD-rate results for hyperparameter tuning, with MSE loss function

The PSNR-Y-gain training curve of one model is shown in Figure 12. This model was overfitted on *BasketballPass* video sequence with Quantization Parameter (QP) of 22.

One experiment was done with another loss function other than the Mean Squared Error (MSE), which was the Structural Similarity Index (SSIM). This model was overfitted for 100 epochs, with 4:1:1 colour weighting, and with a learning rate of 0.001. The BD-rates were: 4.01% for Y-PSNR, 3.04% for U-PSNR, and 2.65% for V-PSNR. Since these values were very high, experimentation on the SSIM loss function was discontinued.

When experimenting with the learning rate, BD-rate was calculated by aggregating 5 QP levels, but also by aggregating only 4 QP levels. Two sets of results were obtained, for the lower QPs (22, 27, 32, 37), and the higher ones (27, 32, 37, 42). These results are displayed in Tables 5, and 6.

There have been experiments on hyperparameters, that weren't simulated, thus no BD-rates were obtained. However, through the overfitting losses, we can still conclude insights about the hyperparameters. These results are shown in Table 7. For the losses, Mean Squared Error (MSE), and Mean Absolute Error (MAE) were tried. In the

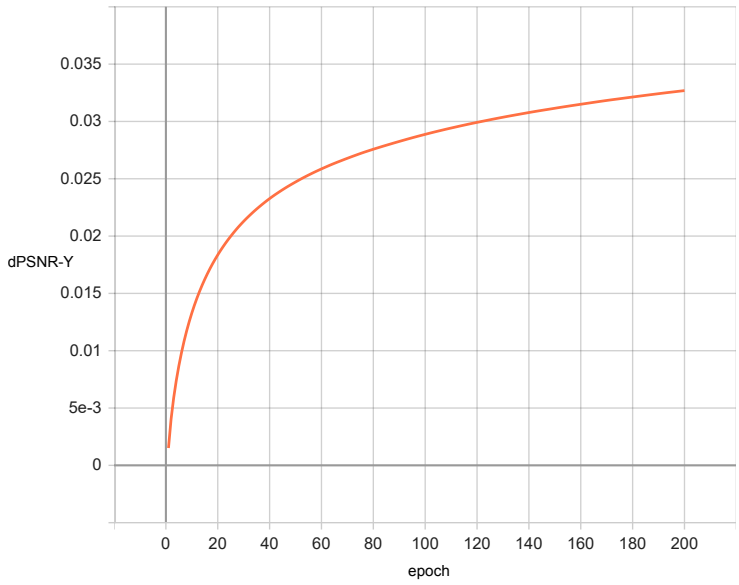


Figure 12: Delta-PSNR-Y(dPSNR-Y) curve during overfitting for one video sequence. LR=0.0001, colour weighting=12:1:1, epoch count=200, loss function=MSE

LR	Colour weight	Epochs	Y-PSNR	U-PSNR	V-PSNR	Y-MSSIM
0.01	4:1:1	100	-0.95%	-7.37%	-7.26%	0.30%
0.001	4:1:1	100	-1.24%	-6.84%	-6.64%	-0.04%
0.0001	4:1:1	100	-1.18%	-4.98%	-5.22%	-0.30%

Table 5: BD-rate results for hyperparameter tuning, for the lower 4 QP values (22, 27, 32, 37), with MSE loss function

YCbCr-Loss column, the separate losses were combined with the *colour weights* for Y:Cb:Cr. The results were sorted by the *Y-dPSNR* component.

4.2 Two-stage overfitting

From the results of the previous section displayed in Table 4, the best performing model was selected as the first-stage model, which is the last entry in the table, i.e., LR = 0.0001, loss function = MSE, colour weighting = 12:1:1, epoch count = 200. Results are in Table 8. The horizontal line is put below the second entry for ease of comparison. The 2nd stage model that started from the base model has to be compared with the 1st stage model that also started from the base model because both have been trained for 200 epochs. The same is true for the 400 epoch models.

Delta-PSNR-Y has been calculated on these 4 overfitted models, averaged on all the sequences and QP values, on both the whole dataset of all the patches from the video sequence, and also on the subset of the dataset which only contains the patches from the video frames that had a positive dPSNR-Y for the first-stage model. These results are in Table 9. Next to the PSNR-gain, the average number of patches were

LR	Colour weight	Epochs	Y-PSNR	U-PSNR	V-PSNR	Y-MSSIM
0.01	4:1:1	100	-0.33%	-9.23%	-8.35%	0.95%
0.001	4:1:1	100	-0.60%	-8.69%	-7.45%	0.70%
0.0001	4:1:1	100	-0.75%	-6.74%	-5.91%	0.17%

Table 6: BD-rate results for hyperparameter tuning, for the higher 4 QP values (27, 32, 37, 42), with MSE loss function

LR	Loss	Colour weight	Y-dPSNR	Y-Loss	YCbCr-Loss
0.001	MSE	12:1:1	0.11046	0.0004955	0.0004405
0.001	MSE	4:1:1	0.10374	0.0004963	0.0003666
0.01	MSE	4:1:1	0.10127	0.0004964	0.0003666
0.001	MSE	4:2:2	0.09555	0.0004972	0.0003015
0.01	MSE	4:2:2	0.09153	0.0004976	0.0003014
0.001	MAE	4:1:1	0.08923	0.0122756	0.0100800

Table 7: Loss results for hyperparameter tuning, without simulation. Epochs=100

counted, that had positive dPSNR-Y out of the batches of size 64. On the top row of this table, the base model's performance is displayed. This is the model from which the overfitting starts in all cases.

The weight update generated by overfitting the CNN model is also transmitted to the decoder, next to the encoded video bitstream. This weight update is encoded using the Neural Network Compression and Representation (NNR) standard, and it also accounts for the compression efficiency of the overall coding. The bitrates of these encoded NNR updates are calculated and they resulted in a 0.82% increase for the second-stage model which started from the base model compared to the first-stage model's NNR update, and a 98.2% increase for the second-stage model which started from the first-stage model.

Stage	Start	Epochs	Y-PSNR	U-PSNR	V-PSNR
1st	base	200	-1.01%	-4.28%	-3.61%
2nd	base	200	-1.00%	-4.13%	-3.60%
1st	1st stage	400	-1.01%	-4.74%	-3.68%
2nd	1st stage	400	-0.62%	-4.43%	-3.33%

Table 8: BD-rate results for two-stage overfitting. LR=0.0001, loss function=MSE, colour weighting=12:1:1

Stage	Start model	dPSNR-Y Full	dPSNR-Y Subset	Nr. pos. Full	Nr. pos. Subset
base	-	0.197	0.208	56.1	58.2
1st	base	0.272	0.28435	61.3	64
2nd	base	0.271	0.28428	61.1	63.7
1st	1st stage	0.2767	0.28897	61.5	63.9
2nd	1st stage	0.2764	0.28898	61.3	63.8

Table 9: PSNR-Y-gain (*dPSNR-Y*), and the average number of positive dPSNR-Y patches in a batch out of 64 (*Nr. pos.*) for the models shown in Table 8. The top row shows the base model's performance. Calculated on the full dataset of patches from the video frames, and on the subset of the dataset (dPSNR-Y > 0)

5 Discussion

In this section, analysis and explanation will be given for the above-listed results, while also recommendations for future research are given.

5.1 Interpretation of results

When comparing model performances, mostly the luma (Y) component is taken into account due to the human eye's sensitivity to luminance over chrominance (U, V). For the Bjøntegaard-Delta rates (BD-rates), lower values are preferred, such that if the value is negative, it shows a gain over the VVC Test Model's (VTM) performance. Also for the losses: MSE, MAE, and SSIM, the lower values are more advantageous, while in the Peak Signal-to-Noise Ratio (PSNR), and delta-PSNR metrics, the greater values indicate better performance, due to it inversely correlating to the Mean Squared Error (MSE).

5.1.1 Hyperparameter search

From the fine-tuning experiments, we were able to draw conclusions about which hyperparameters were suited for the overfitting task on the architecture presented in Section 3.4.2:

- Learning rate

  Out of the tested Learning Rates (LR) of 1e-2, 1e-3, and 1e-4, based on the BD-rates obtained in Table 4 we concluded that the LR of 1e-4 gives the best results. The learning rate is dependent on many factors, such as the NN being trained, the data and its characteristics. Therefore, it's difficult to predict a priori. For this reason, the best-suiting LR is found mostly by trial and error. Preliminary tests were made with LR of 1e-5 that weren't displayed in the results section, however, their evaluation didn't perform that well, giving us the insight that the best-suited learning rate for this task lies between 1e-3 and 1e-5.

  A limitation of this approach is that the learning rates tried were all constant. Using adaptive learning rates, or learning rate schedulers would yield us more profound results, because these methods adjust the learning rate based on the characteristics of the data, potentially obtaining a good learning rate automatically [13].

- Loss function

  While developing neural video coding techniques, the standardized way to compare different models is by the Peak Signal-to-Noise Ratio (PSNR), which is directly correlated with Mean Squared Error (MSE). Thus, if the network is optimizing over the MSE, it will obtain the best PSNR values as well. This reasoning is proven with the overfitting dPSNR-Y values in Table 7, and from the BD-rates of the SSIM loss function. From these results, we saw that using the Mean Absolute Error (MAE) and the Structural Similarity Index (SSIM),

the metrics were performing worse compared to the models using MSE by a greater margin. These two losses aren't related directly to PSNR. Also, the MSE is more sensitive to outliers, which is an advantage since it minimizes large errors, not allowing bigger artefacts in the decoded video sequence [24, 98].

- Epoch count

  Experimenting with epoch counts brings with itself a trade-off between quality and training time, since with a higher epoch count better results can be obtained, however training for a very long time is not preferred in the context of video coding, when usually the videos have to be served without much delay [13]. Although, for the purpose of fine-tuning the models, long overfitting times are experimented with, to observe how well can the NN model perform.

  During the experiments, epoch counts of 100, 200, and 400 were tested, in Tables 4, and 8. Sequentially, the higher epoch counts brought better results. From Figure 12 can be observed that the model hasn't yet converged, i.e., with further epochs, higher dPSNR-Y could be obtained, although it slowed down after approximately epoch 50.

- Colour weighting

  Colour weighting influences the loss function calculation since losses are calculated for each colour (and luma) component separately. These separate losses are added together, each having a weight associated with it. Results from Tables 4, and 7 suggest that the weights of 12:1:1 are the most preferable and that 4:2:2 weighting yields worse results than 4:1:1 weighting.

  In the 12:1:1 weighting, $\frac{12}{14}$ of the loss is given by the loss of the luma component, i.e., the model is largely optimizing over the luma component. This is advantageous due to the higher priority of luminance over chrominance. Higher Y-weights obtain better Y-loss, Y-dPSNR, and Y-BD-rates. These are the metrics that are mostly taken into consideration when comparing models.

  From the table that shows results without simulation (Table 7), the aggregated YCbCr-loss is displayed which also takes into account the chroma components (Cb, Cr). Based on this metric, the weighting of 4:2:2 performs the best (lower loss is preferred).

5.1.2 Insight on quantization parameters

The Quantization Parameters (QP) indicate how much quantization and compression is done to the video frames, where lower QP values mean a higher quality. From Tables 5, and 6, where the BD-rate of 3 models are shown but not for all QPs, but for the lower- (22, 27, 32, 37), and higher (27, 32, 37, 42) QP values, it can be observed that for the lower QPs, the models perform better. This quality is also reassured by the Y-Multiscale-Structural Similarity Index (Y-MSSIM), which is more closely related to the visual quality of images and videos as perceived by humans. This finding suggests

that a higher gain is possible to obtain on the sequences with higher QPs, i.e., on videos with higher quality.

5.1.3 Two-stage overfitting

In the two-stage overfitting, the best-performing first-stage model was preserved and held as an anchor model, to which the second-stage model is compared. This first-stage model has as hyperparameters the following: LR=1e-4, loss function=MSE, colour weighting=12:1:1, and epoch counts=200. This first-stage model was the baseline on which the PSNR-Y of each patch in the video sequence was calculated. After that, the negative PSNR-Y patches were removed to form a subset of the dataset on which the second-stage model was overfitted. Since there were two second-stage models, one that started overfitting from the base NN model, and the other that started from the first-stage model, one of them had 200 epochs of training, and the other had 400 since the first-stage model already trained for 200 epochs (but on the whole dataset). For comparability, the first-stage model was further trained for 200 epochs to match the epoch count of the 400-epoch second-stage model.

The results in Table 8 show the BD-rates of the 4 models, and suggest that the second-stage models underperform, i.e., they have higher Y-PSNR BD-rates, compared to the first-stage models. The second-stage model with 200 epochs also underperformed compared to its first-stage model, although, only by a small margin. This is also shown in the table with the overfitting dPSNR values (Table 9), where we don't see improvement in second-stage dPSNR values. For comparison, in the same table, we can observe that the difference between the base model's and the first-stage model's dPSNR-Y is much greater than between the first- and second-stage models' dPSNR-Y values. These results are contradictory to the anticipated results, since by removing the "bad elements" from the dataset, i.e., the patches that brought negative PSNR-gain, the metrics should have improved. Also, general overfitting usually happens due to fewer amounts of samples in the dataset, since the model has fewer characteristics to learn on, so it can pick up on the particularities, which is our purpose during the content-adaptive training of loop filters [13]. Instead, they have gotten worse.

Since the VVC Test Model (VTM) decides at each Coding Tree Unit (CTU) if it will use the overfitted NN in-loop filter or not, if some patches have negative dPSNR for the first-stage model, the gain that the overfitting brings only shows up in the patches with positive dPSNR, thus by only focusing on the positive patches to make them obtain even higher gains, should bring better BD-rates as well.

The reason why this is not happening is that during inference (simulation), the second-stage model obtains higher bitrate overhead if it gets used, thus the VTM tries to minimize its usage since the decision to use the NN filter depends on the Rate-Distortion Optimization (RDO) (where the bitrate is also being taken into account). After observing the NN's weight update bitrates, it was shown that the drop in performance is most likely due to this reason. For the second-stage model which started overfitting from the first-stage model, its NNR bitrate nearly doubled compared to the first-stage model's weight update bitrate. It is true that for the second-stage

model that started from the base model, the weight update bitrate is almost identical to its first-stage correspondence, however, its BD-rate is also almost identical

Another possible explanation is that during overfitting, the model "sacrifices" some patches (on average 2.6) that don't "follow the trend" of the majority, in the favour of the others, so the others can perform better. Thus, by removing these patches, the model doesn't have other patches to "sacrifice". In Table 9, we can observe that the average number of positive dPSNR patches in the subset of data for the second-stage model drops from 64 to 63, i.e., on average one patch obtains negative dPSNR instead of maintaining its characteristic of having positive dPSNR-Y value. (Since the beginning of the training of the second-stage model, all 64 patches in a batch have positive dPSNR.)

From Table 9, we can observe that the second-stage models omit on average 2.6 patches from each batch of size 64. This is not a substantial amount from each batch, however, it means that if the whole dataset had 60 batches in total, in the subset of the dataset, the number of batches of size 64 would only amount to 55 batches. This means that over 200 epochs, less training is happening for the second-stage model, by 1000 iterations of batches (if we look at the processing of one batch as one iteration). This could be the reason why in the same table, we don't see improvement in dPSNR-Y values for the 2^{nd} stage models compared to the 1^{st} stage ones.

5.2 Future work

Possible future research can be performed both on hyperparameter tuning and two-stage overfitting.

In the **hyperparameter search**, adequate results and insights were obtained with the best-performing models, however, it is not believed that the search was extensive enough to find the best possible model. For this reason, the following opportunities could be exploited:

- The learning rates used in the experiments were all constant LRs, which has its limitations, due to different stages of the training might require different granularities, e.g., in higher epochs, a smaller learning rate is required when the model shows signs of convergence. Thus, the use of adaptive learning rates, or LR schedulers ought to be used to explore the model's highest possible training curves [13, 55].

- As mentioned, choosing the epoch count is a trade-off between training time and performance. For those applications where training time is not a substantial aspect, much larger epoch counts could be considered to understand the neural network's ability to overfit on one sequence, i.e., experiments could be done to find out at what point the model converges.

- Because out of all the metrics calculated the most regarded metrics are the ones of the luma component, the higher weighting the Y-losses get, the better they shall perform. Experiments could be done on different colour weighting

schemes, where higher weight would go on luminance, but not too large, to overthrow the chrominance components.

- Since, during the whole process, the multipliers of the CNN loop filter were overfitted, experiments could be done on overfitting on the bias of the NN model, instead of the multiplier. Research has already shown successful content-adaptive overfitting biases for post-filters [11, 76]. The same techniques could be used for in-loop filters too.

In the **two-stage training**, as we have concluded, the first-stage models outperform their corresponding second-stage models. However, it is concluded that it happens due to the bitrate overhead caused by the weight update of the CNN. Further testing and research could be done to see the reasons behind it. One could examine the flags during inference that show if the VTM has used the overfitted loop filter, to check the difference between the first and second-stage models. They could be indicative of the ratio of patches with the NN loop filter used and not used. This could show if the patches that have been overfitted on, are being decoded with the NN in-loop filter. Examination can also be done on the reasoning for why the bitrate overhead is higher than for the first-stage model, and to experiment with reducing this overhead.

6 Conclusions

The constantly growing need to develop and evolve new techniques in the realm of video coding is pushing the research in the direction of Machine Learning (ML). Neural Networks (NN) are exploited to reduce blocking, ringing, and blurring artefacts produced by the block-based modern video coding standards such as the High Efficiency Video Coding (HEVC), or Versatile Video Coding (VVC).

In-loop filters are used to enhance the visual quality of video frames, and NN-based in-loop filters are employed to be used next to the existing filters, such as the Deblocking Filter (DBLK), Sample Adaptive Offset (SAO), or the Adaptive Loop Filter (ALF). These filters differ from post-processing filters because the frames that have gone through the loop filter, are referenced for other frames to be coded.

Low-complexity Convolutional Neural Networks (CNN) have been utilized to learn from the video that is meant to be coded, in a content-adaptive nature, such that after a base loop filter model has been pretrained on a large general dataset, it is further overfitted on one video sequence, and the network's weight update is signalled to the decoder side while being encoded using the Neural Network compression and Representation (NNR) standard. To minimize this bitrate overhead caused by the overfitting, only one set of parameters, named multipliers is fine-tuned.

While using a particular CNN architecture that has shown favourable results, a hyperparameter search has been conducted during this study to shed light on the hyperparameters that can enhance the performance of these NN loop filters:

- Results have shown that the optimal Learning Rate (LR), out of the tested ones, is of value 1e-4, however, it is assumed that an adaptive learning rate or a learning rate scheduler would yield better results, due to their aspect of dynamically changing when learning slows down.

- Experimenting with epoch counts is a trade-off between training time and visual quality since the models haven't converged with 200 or 400 epochs, thus with more epochs, better performance can be achieved, however, training times would multiply.

- The Mean Squared Error (MSE) is visibly the leading loss function to use in this context, where the comparison between different video coding techniques is done with the Peak Signal-to-Noise Ratio (PSNR) metric, that is computed with the MSE.

- Since modern video coding algorithms exploit the human visual system's sensitivity to luminance (Y) over chrominance (U, V), more emphasis is put on the luma component of the metrics. With this reasoning, the loss function is calculated by giving higher priority to the luma component, with the hyperparameter of colour weighting. In the experiments, better results were obtained with 12:1:1 weighting for Y:U:V components, compared to the 4:1:1, and 4:2:2 settings.

With the best-performing hyperparameters, we were able to produce a gain over the unfiltered video data, i.e., the reconstruction of the VTM NNVC 5.0 standard. This gain accounted for, on average, 1.01% (Y), 4.28% (Cb), and 3.61% (Cr) Bjøntegaard Delta rate (BD-rate).

After finding a beneficial set of hyperparameters, the best-performing model was set as the first-stage model and baseline of comparison for the second-stage models. Based on this model, the patches that returned negative PSNR-Y-gain, compared to the pretrained base loop filter, were removed from the dataset, forming a subset of patches. The second-stage models overfitted on this subset, either by starting their training from the pretrained model, just like the first-stage model, or from the first-stage model that has already trained for 200 epochs on the full dataset. Although their performance was still outputting positive results, they underperformed by, on average, 0.20% (Y), 0.23% (Cb), and 0.18% (Cr) BD-rate compared to the first-stage models. This phenomenon happens due to the bitrate overhead caused by the second-stage model, which is so high that the Versatile Video Coding (VVC) Test Model (VTM) decides to use the overfitted in-loop filter less often than the first-stage model. However, prospective research could be conducted on understanding the reason, and to reduce the bitrate overhead.

While the two-stage overfitting technique has partly failed to outperform its first-stage model, research is needed to exploit every possibility to enhance the video coding efficiency demanded by the modern worldwide usage of the internet.

References

[1] Cisco. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. White paper, 2019.

[2] Cisco. Cisco Annual Internet Report (2018–2023) White Paper, 2020.

[3] Dong Liu, Yue Li, Jianping Lin, Houqiang Li, and Feng Wu. Deep Learning-Based Video Coding: A Review and a Case Study. *ACM Comput. Surv.*, 53(1), feb 2020.

[4] Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the High Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1649–1668, 2012.

[5] Benjamin Bross, Ye-Kui Wang, Yan Ye, Shan Liu, Jianle Chen, Gary J. Sullivan, and Jens-Rainer Ohm. Overview of the Versatile Video Coding (VVC) Standard and its Applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3736–3764, 2021.

[6] Yue Li, Li Zhang, and Kai Zhang. Convolutional Neural Network Based In-Loop Filter For VVC Intra Coding. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 2104–2108, 2021.

[7] Chuanmin Jia, Shiqi Wang, Xinfeng Zhang, Shanshe Wang, Jiaying Liu, Shiliang Pu, and Siwei Ma. Content-Aware Convolutional Neural Network for In-Loop Filtering in High Efficiency Video Coding. *IEEE Transactions on Image Processing*, 28(7):3343–3356, 2019.

[8] Zhijie Huang, Jun Sun, Xiaopeng Guo, and Mingyu Shang. Adaptive Deep Reinforcement Learning-Based In-Loop Filter for VVC. *IEEE Transactions on Image Processing*, 30:5439–5451, 2021.

[9] Fatemeh Nasiri, Wassim Hamidouche, Luce Morin, Nicolas Dhollande, and Gildas Cocherel. Model Selection CNN-based VVC Quality Enhancement. In *2021 Picture Coding Symposium (PCS)*, pages 1–5, 2021.

[10] Maria Santamaria, Ruiying Yang, Francesco Cricri, Honglei Zhang, Jani Lainema, Ramin G. Youvalari, Hamed R. Tavakoli, and Miska M. Hannuksela. Overfitting multiplier parameters for content-adaptive post-filtering in video coding. In *2022 10th European Workshop on Visual Information Processing (EUVIP)*, pages 1–6, 2022.

[11] Maria Santamaria, Francesco Cricri, Jani Lainema, Ramin G. Youvalari, Honglei Zhang, and Miska M. Hannuksela. Content-Adaptive Neural Network Post-Processing Filter with NNR-Coded Weight-Updates. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 2251–2255, 2022.

[12] Gisle Bjøntegaard. Calculation of average PSNR differences between RD-curves. 2001.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] Ticao Zhang and Shiwen Mao. An Overview of Emerging Video Coding Standards. *GetMobile: Mobile Comp. and Comm.*, 22(4):13–20, may 2019.

[15] Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[16] Joint Video Experts Team document management system. https://jvet-experts.org/. Accessed: 2023-09-05.

[17] Mário Saldanha, Gustavo Sanchez, César Marcon, and Luciano Agostini. *Versatile Video Coding (VVC)*, pages 7–22. Springer International Publishing, Cham, 2022.

[18] Mathias Wien. *Video Coding Fundamentals*, pages 23–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[19] Young-Ju Choi, Young-Woon Lee, and Byung-Gyu Kim. Design of perspective affine motion compensation for versatile video coding (VVC). In Jacques Blanc-Talon, Patrice Delmas, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, pages 384–395, Cham, 2020. Springer International Publishing.

[20] Yin Su, Lei Liu, Kongfen Zhu, and Xuefeng Zhang. Directional dct based depth image coding for 3-d video. In Peng You, Heng Li, and Zhenxiang Chen, editors, *Proceedings of International Conference on Image, Vision and Intelligent Systems 2022 (ICIVIS 2022)*, pages 249–256, Singapore, 2023. Springer Nature Singapore.

[21] N. Ahmed, T. Natarajan, and K.R. Rao. Discrete Cosine Transform. *IEEE Transactions on Computers*, C-23(1):90–93, 1974.

[22] Heiko Schwarz, Muhammed Coban, Marta Karczewicz, Tzu-Der Chuang, Frank Bossen, Alexander Alshin, Jani Lainema, Christian R. Helmrich, and Thomas Wiegand. Quantization and Entropy Coding in the Versatile Video Coding (VVC) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3891–3906, 2021.

[23] Guoqing Xiang, Huizhu Jia, Mingyuan Yang, Yuan Li, and Xiaodong Xie. A novel adaptive quantization method for video coding. *Multimedia Tools and Applications*, 77(12):14817–14840, Jun 2018.

[24] G.J. Sullivan and T. Wiegand. Rate-distortion optimization for video compression. *IEEE Signal Processing Magazine*, 15(6):74–90, 1998.

[25] Vivienne Sze, Detlev Marpe, Madhukar Budagavi, and Gary J. Sullivan. *Entropy Coding in HEVC*, pages 209–274. Springer International Publishing, Cham, 2014.

[26] David A. Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[27] Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic Coding for Data Compression. *Commun. ACM*, 30(6):520–540, jun 1987.

[28] Marta Karczewicz, Nan Hu, Jonathan Taquet, Ching-Yeh Chen, Kiran Misra, Kenneth Andersson, Peng Yin, Taoran Lu, Edouard François, and Jie Chen. VVC In-Loop Filters. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3907–3925, 2021.

[29] Mathias Wien and Benjamin Bross. Versatile Video Coding – Algorithms and Specification. In *2020 IEEE International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–3, 2020.

[30] Djordje Mitrovic. Video Compression. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0506/s0561282.pdf. Accessed: 2023-09-08.

[31] Wikipedia contributors. Chroma subsampling — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Chroma_subsampling&oldid=1171013401, 2023. Accessed: 2023-09-08.

[32] Yeongwoong Kim, Suyong Bahk, Seungeon Kim, Won Hee Lee, Dokwan Oh, and Hui Yong Kim. Neural Video Compression with Temporal Layer-Adaptive Hierarchical B-frame Coding, 2023.

[33] Krishna Rao Vijayanagar. I, P, and B-frames – Differences and Use Cases Made Easy. https://ottverse.com/i-p-b-frames-idr-keyframes-differences-usecases/, 2020. Accessed: 2023-09-05.

[34] Sudhakar Radhakrishnan. *Effective Video Coding for Multimedia Applications*. IntechOpen, Rijeka, Apr 2011.

[35] Jonathan Pfaff, Alexey Filippov, Shan Liu, Xin Zhao, Jianle Chen, Santiago De-Luxán-Hernández, Thomas Wiegand, Vasily Rufitskiy, Adarsh Krishnan Ramasubramonian, and Geert Van der Auwera. Intra Prediction and Mode Coding in VVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(10):3834–3847, 2021.

[36] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. Wiley Publishing, 2nd edition, 2010.

[37] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 13(7):560–576, 2003.

[38] Md. Zahirul Islam and Boshir Ahmed. Visualization of CTU Partitioning And Time Complexity Analysis in HEVC and VVC. In *2021 3rd International Conference on Electrical & Electronic Engineering (ICEEE)*, pages 169–172, 2021.

[39] HEVC: An introduction to high efficiency coding. https://www.vcodex.com/hevc-an-introduction-to-high-efficiency-coding/. Accessed: 2023-09-05.

[40] The Moving Picture Experts Group. Supplemental enhancement information messages for coded video bitstreams. https://mpeg.chiariglione.org/standards/mpeg-c/supplemental-enhancement-information-messages-coded-video-bitstreams. Accessed: 2023-09-09.

[41] Mihir Mody. Understanding in-loop filtering in the HEVC video standard. https://www.edn.com/understanding-in-loop-filtering-in-the-hevc-video-standard/, 2013. Accessed: 2023-09-06.

[42] Taoran Lu, Fangjun Pu, Peng Yin, Sean McCarthy, Walt Husak, Tao Chen, Edouard Francois, Christophe Chevance, Franck Hiron, Jie Chen, Ru-Ling Liao, Yan Ye, and Jiancong Luo. Luma Mapping with Chroma Scaling in Versatile Video Coding. In *2020 Data Compression Conference (DCC)*, pages 193–202, 2020.

[43] Andrey Norkin, Gisle Bjontegaard, Arild Fuldseth, Matthias Narroschke, Masaru Ikeda, Kenneth Andersson, Minhua Zhou, and Geert Van der Auwera. HEVC Deblocking Filter. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1746–1754, 2012.

[44] Soulef Bouaafia, Seifeddine Messaoud, Randa Khemiri, and Fatma Elzahra Sayadi. VVC In-Loop Filtering Based on Deep Convolutional Neural Network. *Computational Intelligence and Neuroscience*, 2021:9912839, Jul 2021.

[45] Chih-Ming Fu, Elena Alshina, Alexander Alshin, Yu-Wen Huang, Ching-Yeh Chen, Chia-Yang Tsai, Chih-Wei Hsu, Shaw-Min Lei, Jeong-Hoon Park, and Woo-Jin Han. Sample Adaptive Offset in the HEVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(12):1755–1764, 2012.

[46] Adireddy Ramakrishna, N. S. Prashanth, and G. B. Praveen. SAO filtering inside CTU loop for high efficiency video coding. In *2013 International Conference on Signal Processing and Multimedia Applications (SIGMAP)*, pages 19–22, 2013.

[47] Roman Guskov. Adaptive Loop Filter. https://vicuesoft.com/blog/titles/Adaptive_Loop_Filter/, 2021. Accessed: 2023-09-06.

[48] Kiran Misra, Frank Bossen, and Andrew Segall. On Cross Component Adaptive Loop Filter for Video Compression. In *2019 Picture Coding Symposium (PCS)*, pages 1–5, 2019.

[49] Dandan Ding, Zhan Ma, Di Chen, Qingshuang Chen, Zoe Liu, and Fengqing Zhu. Advances In Video Compression System Using Deep Neural Network: A Review And Case Studies, 2021.

[50] Luka Murn, Saverio Blasi, Alan F. Smeaton, and Marta Mrak. Improved CNN-Based Learning of Interpolation Filters for Low-Complexity Inter Prediction in Video Coding. *IEEE Open Journal of Signal Processing*, 2:453–465, 2021.

[51] Junru Li, Yue Li, Chaoyi Lin, Kai Zhang, and Li Zhang. A Neural-network Enhanced Video Coding Framework beyond VVC. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1780–1784, 2022.

[52] J. Pfaff, P. Helle, D. Maniry, S. Kaltenstadler, W. Samek, H. Schwarz, D. Marpe, and T. Wiegand. Neural network based intra prediction for video coding. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XLI*, volume 10752, page 1075213. International Society for Optics and Photonics, SPIE, 2018.

[53] Siwei Ma, Xinfeng Zhang, Chuanmin Jia, Zhenghui Zhao, Shiqi Wang, and Shanshe Wang. Image and Video Compression With Neural Networks: A Review. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(6):1683–1698, 2020.

[54] Shuai Huo, Dong Liu, Feng Wu, and Houqiang Li. Convolutional Neural Network-Based Motion Compensation Refinement for Video Coding. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–4, 2018.

[55] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[56] Nannan Zou, Honglei Zhang, Francesco Cricri, Hamed R. Tavakoli, Jani Lainema, Miska Hannuksela, Emre Aksu, and Esa Rahtu. L2C – Learning to Learn to Compress. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020.

[57] Yat Hong Lam, Alireza Zare, Caglar Aytekin, Francesco Cricri, Jani Lainema, Emre Aksu, and Miska Hannuksela. Compressing Weight-updates for Image Artifacts Removal Neural Networks, 2019.

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

[59] Ties van Rozendaal, Iris A. M. Huijben, and Taco S. Cohen. Overfitting for Fun and Profit: Instance-Adaptive Data Compression, 2021.

[60] Honglei Zhang, Francesco Cricri, Hamed Rezazadegan Tavakoli, Maria Santamaria, Yat-Hong Lam, and Miska M. Hannuksela. Learn to overfit better: finding the important parameters for learned image compression. In *2021 International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–5, 2021.

[61] Woon-Sung Park and Munchurl Kim. CNN-based in-loop filtering for coding efficiency improvement. In *2016 IEEE 12th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, pages 1–5, 2016.

[62] Xiandong Meng, Chen Chen, Shuyuan Zhu, and Bing Zeng. A New HEVC In-Loop Filter Based on Multi-channel Long-Short-Term Dependency Residual Networks. In *2018 Data Compression Conference*, pages 187–196, 2018.

[63] Yongbing Zhang, Tao Shen, Xiangyang Ji, Yun Zhang, Ruiqin Xiong, and Qionghai Dai. Residual Highway Convolutional Neural Networks for in-loop Filtering in HEVC. *IEEE Transactions on Image Processing*, 27(8):3827–3841, 2018.

[64] Ming-Ze Wang, Shuai Wan, Hao Gong, and Ming-Yang Ma. Attention-Based Dual-Scale CNN In-Loop Filter for Versatile Video Coding. *IEEE Access*, 7:145214–145226, 2019.

[65] Dandan Ding, Lingyi Kong, Guangyao Chen, Zoe Liu, and Yong Fang. A Switchable Deep Learning Approach for In-Loop Filtering in Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(7):1871–1887, 2020.

[66] Shufang Zhang, Zenghui Fan, Nam Ling, and Minqiang Jiang. Recursive Residual Convolutional Neural Network- Based In-Loop Filtering for Intra Frames. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(7):1888–1900, 2020.

[67] H. Wang, J. Chen, K. Reuze, A. M. Kotra, and M. Karczewicz. EE1-related: neural network-based in-loop filter with constrained computational complexity. *JVET 23rd Meeting, document JVET-W0131*, Jul. 2021.

[68] Jian Yue, Yanbo Gao, Shuai Li, Hui Yuan, and Frédéric Dufaux. A Global Appearance and Local Coding Distortion Based Fusion Framework for CNN Based Filtering in Video Coding. *IEEE Transactions on Broadcasting*, 68(2):370–382, 2022.

[69] Zhijie Huang, Jun Sun, Xiaopeng Guo, and Mingyu Shang. One-for-All: An Efficient Variable Convolution Neural Network for In-Loop Filter of VVC. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(4):2342–2355, 2022.

[70] Woody Bayliss, Luka Murn, Ebroul Izquierdo, Qianni Zhang, and Marta Mrak. Complexity Reduction of Learned In-Loop Filtering in Video Coding, 2022.

[71] Mingze Wang, Shuai Wan, Hao Gong, Yuanfang Yu, and Yang Liu. An Integrated CNN-based Post Processing Filter For Intra Frame in Versatile Video Coding. In *2019 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1573–1577, 2019.

[72] Ren Yang, Mai Xu, Tie Liu, Zulin Wang, and Zhenyu Guan. Enhancing Quality for HEVC Compressed Videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(7):2039–2054, jul 2019.

[73] Fan Zhang, Chen Feng, and David R. Bull. Enhancing VVC Through CNN-Based Post-Processing. In *2020 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020.

[74] Kai Cui, Ahmet Burakhan Koyuncu, Atanas Boev, Elena Alshina, and Eckehard Steinbach. Convolutional neural network-based post-filtering for compressed YUV420 images and video. In *2021 Picture Coding Symposium (PCS)*, pages 1–5, 2021.

[75] Di Ma, Fan Zhang, and David R. Bull. MFRNet: A New CNN Architecture for Post-Processing and In-loop Filtering. *IEEE Journal of Selected Topics in Signal Processing*, 15(2):378–387, feb 2021.

[76] Maria Santamaria, Yat-Hong Lam, Francesco Cricri, Jani Lainema, Ramin G. Youvalari, Honglei Zhang, Miska M. Hannuksela, Esa Rahtu, and Moncef Gaubbuj. Content-adaptive convolutional neural network post-processing filter. In *2021 IEEE International Symposium on Multimedia (ISM)*, pages 99–106, 2021.

[77] Fan Zhang, Di Ma, Chen Feng, and David R. Bull. Video Compression With CNN-Based Postprocessing. *IEEE MultiMedia*, 28(4):74–83, oct 2021.

[78] Ruiying Yang, Maria Santamaria, Francesco Cricri, Honglei Zhang, Jani Lainema, Ramin G. Youvalari, and Miska M. Hannuksela. AHG11: Content-adaptive neural network loop-filter. *JVET 31st Meeting, document JVET-AE0093*, Jul. 2023.

[79] Jay N. Shingala, Ajay Shyam, Ajat Suneja, Siddarth P Badya, Tong Shao, Arjun Arora, Peng Yin, Fangjun Pu, Taoran Lu, and Sean McCarthy. EE1-1.1: Complexity Reduction on Neural-Network Loop Filter. *JVET 30th Meeting, document JVET-AD0156*, Apr. 2023.

[80] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks. *CoRR*, abs/1511.08458, 2015.

[81] Phil Kim. *Convolutional Neural Network*, pages 121–147. Apress, Berkeley, CA, 2017.

[82] Thomas Wood. Convolutional Neural Network. https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network. Accessed: 2023-09-08.

[83] Sai Balaji. Binary Image classifier CNN using TensorFlow. https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697, 2020. Accessed: 2023-09-09.

[84] Srikari Rallabandi. Activation functions: ReLU vs. Leaky ReLU. https://medium.com/mlearning-ai/activation-functions-relu-vs-leaky-relu-b8272dc0b1be, 2023. Accessed: 2023-09-12.

[85] Moving Picture Experts Group. MPEG. https://www.mpeg.org/. Accessed: 2023-09-09.

[86] Heiner Kirchhoffer, Paul Haase, Wojciech Samek, Karsten Müller, Hamed Rezazadegan-Tavakoli, Francesco Cricri, Emre B. Aksu, Miska M. Hannuksela, Wei Jiang, Wei Wang, Shan Liu, Swayambhoo Jain, Shahab Hamidi-Rad, Fabien Racapé, and Werner Bailer. Overview of the Neural Network Compression and Representation (NNR) Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(5):3203–3216, 2022.

[87] Ruiying Yang, Maria Santamaria, Francesco Cricri, Honglei Zhang, Jani Lainema, Ramin G. Youvalari, and Miska M. Hannuksela. Low-precision post-filtering in video coding. In *2022 IEEE International Symposium on Multimedia (ISM)*, pages 137–140, 2022.

[88] Jianle Chen, Yan Ye, and Seung Hwan Kim. Algorithm description for Versatile Video Coding and Test Model 11 (VTM 11). *JVET 20th Meeting, document JVET-T2002*, Oct. 2020.

[89] JVET AHG on Neural network-based video coding. VVC Software VTM NNVC-5.0. https://vcgit.hhi.fraunhofer.de/jvet-ahg-nnvc/VVCSoftware_VTM/-/tree/NNVC-5.0?ref_type=tags. Accessed: 2023-09-10.

[90] Elena Alshina, Ru-Ling Liao, Shan Liu, and Andrew Segall. JVET common test conditions and evaluation procedures for neural network-based video coding technology. *JVET 30th Meeting, document JVET-AD2016*, Apr. 2023.

[91] Di Ma, Fan Zhang, and David R. Bull. BVI-DVC: A Training Database for Deep Video Compression. *IEEE Transactions on Multimedia*, 24:3847–3858, 2022.

[92] Fan Zhang, Di Ma, and David Bull. BVI-DVC. *University of Bristol*, 2020.

[93] Tong Shao, Jay N. Shingala, Peng Yin, Arjun Arora, Ajay Shyam, and Sean McCarthy. A low complexity convolutional neural network with fused cp decomposition for in-loop filtering in video coding. In *2023 Data Compression Conference (DCC)*, pages 238–247, 2023.

[94] Tamara G. Kolda and Brett W. Bader. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, 2009.

[95] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017.

[96] TensorFlow. https://www.tensorflow.org/. Accessed: 2023-09-17.

[97] Franck Galpin, Thierry Dumas, Philippe Bordes, Pavel Nikitin, Fabrice Le Léannec, and Edouard François. AHG11: Small Ad-hoc Deep-Learning Library. *JVET 23rd Meeting, document JVET-W0181*, Jul. 2021.

[98] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[99] *Mean Squared Error*, pages 337–339. Springer New York, New York, NY, 2008.

[100] MathWorks. Peak Signal-to-Noise Ration. https://se.mathworks.com/help/vision/ref/psnr.html. Accessed: 2023-09-10.

[101] Z. Wang, E.P. Simoncelli, and A.C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003.