Deep Learning-based Video Coding

Helena Cristina Frutuoso Oliveira

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Fernando Manuel Bernardo Pereira

Prof. João Miguel Duarte Ascenso

Examination Committee

Chaiperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino

Supervisor: Prof. Fernando Manuel Bernardo Pereira

Member of the Committee: Prof. Nuno Miguel Morais Rodrigues

November 2021

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, I would like to thank my family. I would like to thank my mother, Irene, for all her love, support, and encouragement, especially during the final stretch of this journey, I don't think I could have finished this Thesis without you. I would like to thank my father, Arlindo, for all the love, advice, ideas, and availability to lend a helping hand, without you this Thesis would have been much harder. I would also like to thank my sister, Ana, for being there and hearing all my rambling about this work and giving me helpful advice along the years.

I would like to thank my boyfriend, Guilherme, for his unwavering faith in my ability to see this project through. Thank you for your support and for always being eager to help me, either by providing welcoming distractions or helpful words of motivation, not only in this latest endeavor but since the very start. Thank you for everything.

I am very grateful for my closest friends, Marta, Maria, and Filipa, and that they have celebrated with me all the small milestones, it was a pleasure having this journey alongside you guys. To all the members of QNFEF, Serra, Lucas, Catarina, Carmona, Zé, Mariana, Pedro, PC, I sincerely thank you for your friendship and all the good memories we have together, even in the most stressful of times.

I would also like to thank Teresa, Cabecinha, Miguel, Francisco, Ricardo, Sophie, and all the friends I made at IST, for without you, this journey would have been much harder and a lot less fun. Thank you for all the late nights and experiences we shared over these 5 years. I also want to thank João Silva for the constant availability to help and for the advice you gave me during this work.

Last but not least, I would like to thank my advisors, Prof. Fernando Pereira and Prof. João Ascenso, for all the hard work they invested into this Thesis. I am very grateful for all the advice along this journey, the keen eye for detail while reviewing this work, and the quick replies to any question I had. I also thank Instituto de Telecomunicações for providing me with the resources and working space I needed to pursue this project. I would also like to express my profound gratitude to Ren Yang for always promptly and kindly answering my emails and helping me in vital parts of this work.

Thank you all.

Resumo

A codificação de vídeo tem como objetivo processar a informação de vídeo digital originalmente adquirida e codificá-la de forma a alcançar uma compressão eficiente através da exploração das redundâncias espacial, temporal, estatística e percetiva. À medida que mais pessoas têm acesso a meios para capturar, transmitir, armazenar e visualizar vídeos, a codificação de vídeo torna-se uma tecnologia cada vez mais essencial. Atualmente existe uma grande procura de vídeos de alta qualidade e, mesmo com os codecs de vídeo tradicionais mais recentes a compressão oferecida nunca é suficiente, visto que a resolução espacial e a quantidade de vídeos criados continuam a aumentar. Várias gerações de normas de codificação de vídeo têm sido publicadas, todas elas com base em ferramentas de processamento desenhadas pelo engenho humano para tirar partido das redundâncias acima referidas, usando o tipo de arquitetura híbrida, aqui designada como 'tradicional'. Contudo, a procura de maior eficiência de compressão continua a aumentar e novas tecnologias têm vindo a surgir.

Recentemente, os métodos de aprendizagem profunda têm alcançado ótimos desempenhos para tarefas relacionadas com visão computacional e processamento de imagem; assim sendo, é natural que esta nova tecnologia seja também aplicada a outros desafios na área do processamento de sinal, tal como a codificação de imagem e vídeo. Atualmente, várias soluções de codificação de imagem já utilizam métodos de aprendizagem profunda, e nomeadamente também modelos de atenção, com grande sucesso. Por conseguinte, apesar da pesquisa em codificação de vídeo baseada em métodos de aprendizagem profunda estar apenas a começar, este parece um caminho promissor.

O principal objetivo desta dissertação é melhorar a eficiência de compressão de um codec de vídeo baseado em aprendizagem profunda já existente, expandindo o mesmo com um modelo de atenção. Para alcançar este objetivo, esta dissertação revê vários codecs de vídeo baseados em aprendizagem profunda e fornece uma avaliação de alguns destes codecs comparativamente à norma de codificação de vídeo mais recente (VVC). Adicionalmente, é replicado o treino do codec escolhido (RLVC), para garantir um ponto de partida robusto e fiável para as posteriores alterações. De seguida, a arquitetura do RLVC modificada com um modelo de atenção é detalhada, e as subsequentes configurações de treino utilizadas são explicadas e usadas para obter resultados preliminares. Os resultados obtidos com o RLVC modificado não oferecem ainda ganhos consistentes na eficiência de compressão, mas a integração de modelos de atenção em arquiteturas de codificação de vídeo baseados em aprendizagem profunda é um tópico recente e a necessitar ainda de muito investimento, mas que se acredita venha a atingir bons resultados no futuro.

Palavras-chave: codificação de vídeo, eficiência de compressão, aprendizagem profunda, redes neuronais, modelos de atenção

Abstract

Video coding is the process that takes raw, original, digital video data and codes it into a binary representation, targeting efficient compression by exploiting the spatial, temporal, statistical and perceptual redundancies.

As more and more people have means to produce, transmit, store, and watch video data, in multiple application domains, and with increased spatial resolution, video coding has become increasingly more important. There is currently a large demand for high quality videos and even with the most recent (and more complex and efficient) traditional video codecs, the offered compression is never enough, whether for storage or transmission, since the spatial resolution keeps increasing and the amount of video data transmitted keeps rising exponentially. Several generations of so-called ‘traditional’ video coding standards have been issued, all handcrafted designed to exploit the data redundancies. However, since the demand for higher compression efficiency is increasing, new technologies may emerge to play a major role in video coding.

Recently, deep learning-based methods have shown great performance for tasks related to computer vision and image processing, so it was only natural that this novel technology started to be applied to other signal processing challenges like image and video coding and more recently even to point cloud coding. While many deep learning-based image coding solutions have already emerged, notably also exploiting attention models, with very competitive performance, research is only just starting on deep learning-based video coding, which, given the results achieved in image coding, also seems to be a promising path.

The key objective of this Thesis is to improve an already established deep learning-based video codec by extending its architecture with an attention model. To achieve this goal, this Thesis reviews several state-of-the-art deep learning-based video coding solutions and provides a solid and meaningful benchmarking for some of the reviewed solutions with publicly available software against the most recent traditional video codec, the Versatile Video Coding standard, to provide awareness on the current compression performance of these emerging video codecs. Moreover, the training of the selected deep learning-based coding solution (RLVC) is replicated, to guarantee a solid starting point for the following extensions. Then, the RLVC architecture is extended with an attention model and the corresponding training configurations detailed to obtain some preliminary compression performance results to be analyzed. The results achieved with the extended RLVC codec do not yet show consistent compression gains since the integration of attention models in learning-based video coding is still a novel research topic that looks very promising and should return gains in the future.

Keywords: video coding, compression efficiency, deep learning, neural networks, attention models

Table of Contents

List of Figures

List of Tables

List of Acronyms

2D	Two Dimensional
3D	Three Dimensional
AAC	Adaptive Arithmetic Coding
AE	Autoencoder
AI	Artificial Intelligence
AVC	Advanced Video Coding
BD	Bjøntegaard Delta
Cb, Cr	Chrominances
CBAM	Convolutional Block Attention Module
CNN	Convolutional Neural Network
CVPR	Conference on Computer Vision and Pattern Recognition
DCT	Discrete Cosine Transform
DL	Deep Learning
DVC	Deep Video Compression
FCNN	Fully Connected Neural Network
FFNN	Feed-Forward Neural Network
GAN	Generative Adversarial Network
GDN	Generalized Divisive Normalization
GOP	Group Of Pictures
GRU	Gated Recurrent Units-conditioned prior
HD	High Definition
HEVC	High Efficiency Video Coding
HLVC	Hierarchical Learned Video Compression
HVS	Human Visual System
ICCV	International Conference on Computer Vision
iDCT	Integer Discrete Cosine Transform
JCT-VC	Joint Collaborative Team on Video Coding
JPEG	Joint Photographic Experts Group
LSTM	Long Short-Term Memory
MLP	Multi-Layer Perceptron
MPEG	Moving Picture Experts Group
MS-SSIM	Multi-Scale Structural Similarity Index Measure
MSE	Mean Square Error
NLP	Natural Language Processing
NN	Neural Network
PSNR	Peak Signal-to-Noise-Ratio
RA	Random Access
RAE	Recurrent Autoencoder

RD	Rate-Distortion
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RNN	Recurrent Neural Network
RPM	Recurrent Probability Model
RLVC	Recurrent Learned Video Compression
SD	Standard Definition
TV	Television
UHD	Ultra High Definition
UVG	Ultra Video Group
VCEG	Video Coding Experts Group
VMAF	Video Multimethod Assessment Fusion
VQEG	Video Quality Experts Group
VTL	Video Trace Library
VVC	Versatile Video Coding
X-RLVC	Extended Recurrent Learned Video Compression
Y	Luminance

1. Introduction

This first chapter introduces and motivates the topic and objectives of this MSc Thesis, entitled 'Deep Learning-based Video Coding'. It will start by presenting the context and motivation and move on to describe the objectives of this work and the structure of this document.

1.1 Context and Motivation

As technology advances, more and more people use multiple ways to create and consume image and video data, in an increasing variety of environments, applications and devices (see Figure 1). The Human need for visual-based communications has led video data to become nowadays the largest share of Internet traffic. It is estimated that by 2022, online videos will make up more than 82% of this traffic and, by 2023, two-thirds of the installed flat-panel TVs will have 4K definition, also known as Ultra-High-Definition (UHD) [1]. While videos with UHD resolution increase the quality of the user experiences, notably their realism and immersion, they also require more than double the bitrate of High-Definition (HD) video, and nine times more than Standard-Definition (SD) video [1] for appropriate target qualities. To keep up with these rate demands, notably fitting the available network and storage capabilities, after video and image acquisition, the visual data needs to be represented in a compressed form, this means asking for significantly less rate than the raw acquisition rate for the target quality, using image and video coding technology. Thus, image and video coding technology target the representation of visual information in an efficient way by exploiting the spatial and temporal redundancies, the coding symbols statistical redundancy and the irrelevance or perceptual redundancy; this mainly targets minimizing the coding rate for a target quality. For most applications, the basic idea is to preserve the perceptually relevant information to reconstruct a useful replica of the original video.

While image coding focuses on exploiting the spatial redundancy, video coding also has to exploit the larger temporal redundancy, since it would be extremely rate costly to code video ignoring the similarity (or correlation) between consecutive frames. If only data redundancy is exploited, then image and video are coded in a lossless manner, in the sense that the visual data is coded preserving all the information, thus allowing to decode a replica, which is mathematically the same as the original. However, lossless coding is rate expensive and most applications do not require mathematically perfect image and video replicas; this allows using so-called *lossy coding,* which takes advantage of both redundancy and perceptual irrelevance; in this case, the coding process does not target preserving all the image and video information but rather obtaining a decoded replica which is able to offer acceptable quality for the considered application. If desired, transparent quality may be obtained with lossy coding, where the decoded data is perceptually similar to the original data, although it is mathematically different; the advantage is that rate is significantly reduced regarding lossless coding. For most visual-based applications, notably entertainment services with hundreds of millions of users, lossy coding is the typical solution since mathematical equivalence is not really needed to offer a good quality of experience. This may be different for medical applications that usually require lossless coding due to the potential extreme consequences of using less than original visual data. The secret of lossy coding is the exploitation of the human perception characteristics by modeling the Human Visual System (HVS) and considering this model during the coding process.

Figure 1: Top: Example of applications that heavily rely on video content [2]. Bottom: Example of video reproducing devices [3].

The key objective of any lossy video codec is to achieve a good trade-off between the perceptual quality and the coding rate. Traditional image and video codecs, notably used in the very popular Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MPEG) standards, work with lossless coding tools exploiting the spatial, temporal, and statistical redundancy, and lossy coding tools (notably quantization) exploiting the perceptual irrelevancies to reduce the coding rate for the target quality. The most important of these processes is exploiting the temporal redundancy since consecutive video frames share a large amount of information; moreover, since videos include much activity associated to motion, motion estimation and compensation tools are paramount for video codecs to spare plenty of rate by creating frame predictions based on the already decoded frames; this allows coding only the frame difference, the so-called *residual*, between the corresponding original and prediction frames. The spatial redundancy, in the original frames or also in the residual frames, is typically exploited by applying a transform, which creates coefficients with a frequency interpretation as in the popular Discrete Cosine Transform (DCT) used in the JPEG and MPEG coding standards. These transform coefficients are very convenient to eliminate information to which the HVS is less sensitive to, such as higher spatial frequencies or small stimulus masked by larger stimuli, notably by performing appropriate quantization where rate may be reduced without a significant or any perceptual quality penalty. In traditional video codecs, each tool is carefully handcrafted and designed to achieve the best compression trade-offs for a wide range of rates. Along the last three decades, successively more efficient image and video coding standards have been designed by adopting more sophisticated and complex coding tools since both the encoding and decoding platforms offer increasing computational power and memory. For video, the most recent coding standard is the Versatile Video (VVC) Coding standard, also known as ITU-T H.266 finalized in 2020, by the Joint Video Experts Team (JVET), a joint video expert team of the Video Coding Experts Group (VCEG) working group of ITU-T Study Group 16 and the MPEG working group of ISO/IEC JTC 1. It is the successor of the High Efficiency Video Coding (HEVC/H.265) and Advanced Video Coding (AVC/H.264) standards, and offers a rate reduction of around 50% regarding HEVC for the same perceptual quality.

Recently, machine learning has emerged as the most efficient technology for many processing tasks (see Figure 2), notably for multimedia analysis tasks, with emphasis on detection, recognition, and classification tasks. Many studies have shown that machine learning tools, and more specifically deep learning (DL) tools, have powerful visual representation capabilities and can be applied with benefits to multiple application domains. Thus, it was without surprise that DL tools have recently emerged to build competitive solutions for compression efficient image coding, notably comparing to standards such as JPEG 2000 and High Efficiency Video Coding (HEVC) in Intra mode. These advances have been recognized by the JPEG standardization group, which launched a project on DL-based image coding known as JPEG AI (Artificial Intelligence) which is launching a Call for Proposals in January 2022.

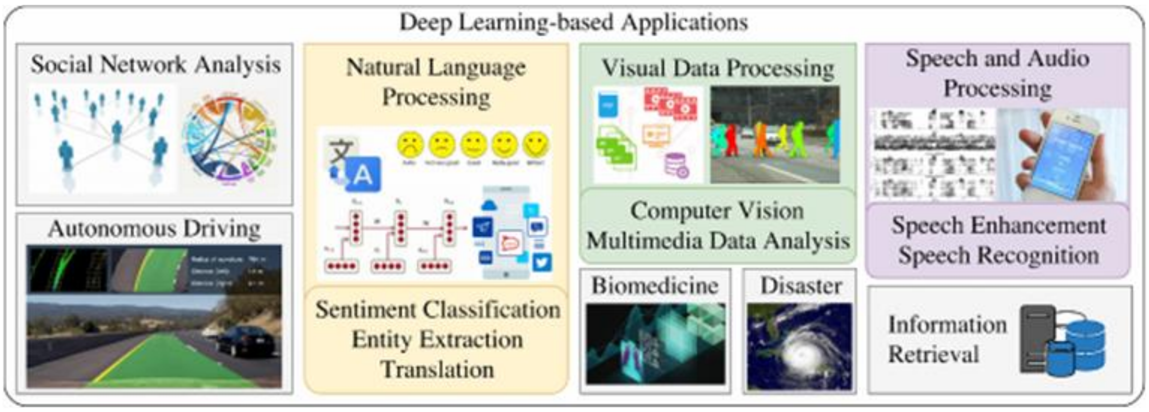
Deep Learning-based Applications
Social Network Analysis
Natural Language Processing
Visual Data Processing
Speech and Audio Processing
Autonomous Driving
Computer Vision Multimedia Data Analysis
Speech Enhancement Speech Recognition
Sentiment Classification Entity Extraction Translation
Biomedicine
Disaster
Information Retrieval

Figure 2: Examples of deep learning-based applications [2].

For video coding, this DL transition process is less advanced than for image coding for two main reasons: i) exploiting the temporal redundancy and understanding the motion in video requires additional, complex tools beyond those proposed for image coding; and ii) video coding implies dealing with much larger amounts of data thus requiring larger computational power and memory.

It is well known that attention plays a key role when humans interact with the world around them and thus also with visual information; in fact, in visual data, not all pixels carry the same amount of information for the purposes of the target applications. For this reason, mechanisms that model human attention have been developed along the years and integrated in several multimedia solutions, notably in conventional video coding.

In this context, it was rather expected that one of the most recent developments in DL is related to attention models, which are techniques for neural networks that enable them to process complex inputs and focus on specific aspects of these inputs [4], which have already been exploited to solve some problems with great success, e.g., Neural Machine Translation. The first DL-based attention models were designed to be integrated into DL architectures, with the goal to improve the performance of Natural Language Processing (NLP) tasks [5], which highly depend on temporal correlations, and have since proven to be a great addition; so much so that solutions based only on attention models have shown remarkable results [6]. Recently, attention models have been applied with large success to computer vision tasks [7] [8], notably video classification [9]. Thus, since attention models have been created to deal with temporal data, and have already been applied successfully to image and video classification tasks, this Thesis speculates that they might come to also have positive impact on DL-based video coding solutions.

Since video coding is nowadays an omnipresent technology in our society, with exponentially increasing usage, and DL a very promising and powerful technology, this work embraces the challenge to study and, ideally, advance the state-of-the-art on DL-based video coding trying to follow the steps already made for DL-based image coding, thus targeting the same competitive compression efficiency performance regarding traditional video codecs, notably by integrating an attention model in a DL-based video coding solution. It is worthwhile to highlight that DL-based video coding may not only be very efficient to represent the video data for visualization purposes but may also have the additional benefit of allowing multiple analysis tasks by machines with the same compressed representation, thus offering a dual-purpose coding approach, for humans and machines, not embraced by current video coding standards.

1.2 Objectives

The final objective of this MSc Thesis is to advance the state-of-the-art on DL-based video coding, notably by:

- Reviewing the literature on DL-based video coding;
- Benchmarking several DL-based video coding solutions in the literature regarding the most advanced conventional video coding standard, i.e., VVC;
- Selecting one of the DL-based video coding solutions in the literature with available software to perform further experiments and developments;
- Replicating and studying the training process and performance for the selected DL-based video coding solution;
- Proposing novel architectures based on the selected DL-based video integrating an attention model;
- Preliminarily studying the compression performance of the DL-based video coding solutions integrating the attention model.

This goal has many challenges, notably because video coding needs to consider the temporal redundancies and most research on DL-based visual coding up to now has only targeted image coding. A specific challenge of this goal is the very large complexity associated to the training process, which requires large training times even for the most powerful computational platforms available.

It is worthwhile to note that the benchmarking of several DL-based video coding solutions in the literature regarding the most advanced conventional video coding standard has led to a conference paper entitled “Conventional versus Learning-based Video Coding Benchmarking: Where Are We?”, already accepted and selected as a best paper award candidate at the International Workshop on Advanced Image Technology (IWAIT) to be held in Hong Kong, January 2022.

1.3 Structure of the MSc Thesis Report

This MSc Thesis report will be structured as follows: in Chapter 2, conventional video coding solutions and standards will be briefly overviewed; in Chapter 3, first the basic concepts and tools of DL and neural networks will be reviewed, then a novel classification of DL-based video coding solutions is proposed and, finally, some key solutions in the literature for each defined class will be reviewed; Chapter 4 will present the methodology and results achieved for the benchmarking of several DL-based video coding solutions reviewed in Chapter 3 regarding the most recent conventional video coding standard; Chapter 5 will describe the training process and replication of performance results for one of the most promising DL-based video coding solution in the literature, which has been selected as the basis for further work in this Thesis; Chapter 6 will report on the preliminary study of the integration of an attention model into the selected DL-based video coding solution; finally, Chapter 7 will conclude this MSc Thesis by discussing the conclusions and highlighting some relevant future work.

2. Conventional Video Coding: A Brief Review

As mentioned in Chapter 1, several goals need to be considered in efficient video coding, notably the exploitation of the temporal, spatial and statistical redundancies as well as the perceptual irrelevance. Before entering the new learning-based video coding world, this chapter offers a brief review of the current conventional video coding world.

First, the HVS characteristics are exploited by acknowledging the higher sensitivity to brightness, or luminance, than to color in images and video. This leads to adopting, for efficient coding, a color space based on three components, which are not typically the Red, Green, and Blue, the components used for acquisition, but rather the luminance (Y, that represents an image in grayscale) and two chrominance components (Cb and Cr); this allows adopting chroma subsampling where the chrominance components are coded with a lower spatial resolution than the luminance as suggested by the HVS characteristics. The most common chroma subsampling format is 4:2:0, which means that for every chrominance sample there are four luminance samples, i.e. the double for each direction.

To exploit the spatial and temporal redundancies, video codecs usually code groups of samples in frames together, also called coding units, in one of two key ways: the so-called *intra coding mode*, corresponding to coding not exploiting any temporal redundancy, very much like coding an image, where only the spatial and statistical redundancies are exploited, eventually performing intra predictions to compute a spatial residue; and the so-called *inter coding mode*, that exploits also the temporal redundancy, where each coding unit is coded by a *temporal residue* computed as the difference from the original to an appropriate temporal prediction, eventually after motion compensation.

Video frames are composed by many coding units, and it is the task of the *encoder control* to decide which is the best coding mode for each of these units: intra or inter coding. However, because some applications demand functionalities like *random access* and *fast forward playing*, it is necessary to have some frames where none of their coding units depends (gets predictions) on any other previously decoded frames; for example, random access in video storage or TV broadcasting requires fully Intra-coded frames so that a video can start playing from almost any point in time without having to decode all (or many) the previous frames, which would happen if they were all Inter-coded. The need for these fully Intra-coded frames impacts the compression performance of a codec since temporal redundancy cannot be fully exploited and, as such, they should be used in a paced manner, i.e., just to satisfy the random access requirements, typically one frame every half second. In summary, depending on the way the key temporal redundancy is exploited, there are three main types of coding for video frames in traditional codecs:

- **Intra frames (I-frames)**: All the coding units in these frames are coded with the intra coding mode. It is possible to exploit the frame spatial redundancy to create Intra-frame predictions at a coding unit level and thus code an Intra-prediction residue. Still, because temporal redundancy is not exploited, these frames are the most rate expensive to achieve a target decoded quality.

- **Predicted frames (P-frames)**: Coding units in these frames may use the intra coding mode, or the inter coding mode but only with forward temporal prediction, this means depending only on previously decoded past frames. Typically, P-frames can only be predicted from previous I-frames or P-frames and are more efficient than I-frames as they partly exploit the temporal redundancy.
- **Bidirectionally predicted frames (B-frames)**: Coding units in these frames may use the intra coding mode, or the inter coding mode with both forward and/or backward temporal prediction simultaneously, based on either I-frames, P-frames or even other B-frames. Because of backward prediction, which uses predictions from future previously decoded frames, B-frames may introduce algorithmic delay and also require more memory to store more reference frames. Since B-frames exploit temporal redundancy more, they are the most rate efficient to achieve a target quality.

When using the Inter coding mode, motion estimation and compensation are very important. In the most recent traditional video codecs, coding units may be partitioned into prediction units (with associated motion vectors) that may vary adaptively in size, to better suit the type of motion happening within each area of a coding unit. Motion estimation and compensation are performed between a previously decoded frame, and an original frame that are close in time. After estimating the motion for each potential prediction unit, the motion vector corresponding to the most efficient solution is applied to create a prediction unit using motion compensation; the more similar the prediction to the original it intends to replicate, the smaller the residue and the lower the rate to achieve a target quality. However, motion estimation is a computationally heavy process, whose sophistication has been increasing along time, very much depending on the affordable complexity in real deployments. When using an Intra coding mode, a prediction may also be created, in this case using already decoded data from the same frame, i.e., typically neighboring pixels are used to create predictions for some prediction unit.

After computing the temporal or intra residue, a transform is applied to exploit the spatial redundancy in this residue, often using blocks of 8x8 samples. This transform, usually the popular Discrete Cosine Transform (DCT) in integer or floating-point precision, is applied to represent the original signal in another domain, commonly a frequential domain, where there is less spatial redundancy and the signal's energy is more compact. To exploit the perceptual irrelevance, the transform coefficients are appropriately quantized, eventually using a precision where the quantization error is below the HVS sensitivity; this allows increasing the compression factor by adopting a lossy coding process, eventually still reaching perceptually transparent quality. Finally, to further enhance the compression performance, all the coding symbols, i.e., motion vectors and quantized transform coefficients are *entropy coded* to exploit the statistical redundancy.

Figure 3 shows the typical encoding architecture of current video coding standards.

Naturally, the decoder side of the codec (diagram inside the dotted line in Figure 3) is also included in the encoder side because the encoder needs to create (temporal and spatial) predictions based on the same decoded information available at the decoder, thus guaranteeing a perfect encoder-decoder prediction synchronization. Therefore, after encoding each frame, the encoder also decodes it, to have available as starting point to create predictions the same decoded frame that is present at the decoder.

Naturally, channel errors may disrupt the encoder-decoder prediction synchronization (even if error correction is performed), and this may require periodic Intra coding for prediction resynchronization, thus limiting the propagation of decoding errors.

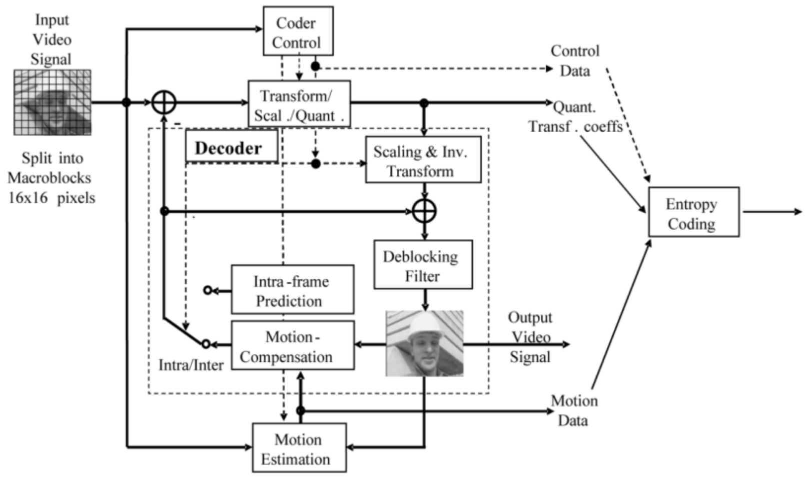

Figure 3: Typical standard video encoding architecture [10].

There are several conventional video codecs used nowadays. While the H.264/AVC standard [10] is clearly the most popular, the H.265/HEVC standard [11] is more advanced, complex, and recent than the H.264/AVC standard, and naturally achieves better compression performance; however, it has not been widely adopted, especially due to patent licensing issues. Very recently, the Versatile Video Coding (VVC) standard has been issued [12], which is even more advanced and complex but also more efficient than H.265/HEVC; typically, each new coding standard reduces to half the rate necessary to achieve the same perceptual quality. Later in this Thesis, the most recent VVC standard will be used to benchmark the learning-based video coding solutions.

To assess the rate-distortion (RD) performance of video codecs, the most common quality metrics used are the Peak Signal-to-Noise-Ratio (PSNR), basically a metric depending on the mathematical error between the original and corresponding decoded frames, which does not consider the HVS characteristics, and the Multi-Scale Structural Similarity Index Measure (MS-SSIM), that also compares the original and the corresponding decoded video frames, but uses a perception-based mathematical model to achieve better correlation with human perception (measured with subjective experiments) compared to PSNR. While MS-SSIM is typically applied only to the luminance, PSNR may be applied only to the luminance (PSNR-Y) or also to the three components (PSNR-RGB, PSNR-YUV). Another video quality metric which became recently very popular is the Video Multimethod Assessment Fusion (VMAF) [13], which was developed by Netflix, and estimates a quality score for the luminance channel by fusing the output of several quality assessment algorithms with a machine learning process (in this case, a support vector machine). The quality metric has a rather high correlation with the quality perceived by humans, mainly due to the use of machine learning and a large dataset of videos and corresponding subjective quality scores. Later in this Thesis, PSNR-Y and VMAF will be used as the video quality metrics.

3. Deep Learning-based Video Coding: Basics, Classification and Review

After starting by presenting the main concepts and tools on Deep Learning-based (DL-based) technology and proposing a classification ontology for DL-based video coding solutions, this chapter will review the most relevant video coding solutions in the literature adopting this technological paradigm.

3.1. Basic Concepts and Tools

Deep learning is a subset area of machine learning using tools and algorithms based on artificial neural networks, or simply, neural networks (NN). These NNs are loosely inspired by biological brains, and the neurons that constitute them. Thus, each NN is made up of neurons, or nodes, which transmit signals with information to each other, just like biological neurons transmit information through synapses. Neural networks are useful because they are able to perform complex tasks very efficiently, notably by learning from a set of examples. Typically, a dataset is needed (usually with labeled samples) for the learning of a neural network model; when this labeled dataset is a good representative of the objectives of the task being addressed, the neural network generalizes for unseen cases. While machine learning considers several types of learning methods, the most relevant ones in the context of video coding are supervised methods, where a model learns from labeled examples, meaning that it knows the ground truth labels for specific input data and can understand the relationship between the input (data) and output (labels); on the contrary, unsupervised methods, learn a model from unlabeled examples, where the inferences are achieved by extracting features and patterns.

Similar to a biological neuron, the nodes in a NN receive input signals which are processed, and its single output is transmitted to other neurons. Figure 4 shows the theoretical model of a neuron (*k*). Each neuron receives the input signals (x_{ki}), which are multiplied by the corresponding weights (w_{ki}) in a linear fashion. These weights may be negative or positive and are learned to achieve the best combination of impacts given to each specific input. After the inputs are multiplied by their respective weights, they are added together with a bias (b_k), thus allowing the neuron to represent any linear function. The so-called *activation function* ($\phi(v_K)$) receives the previous linear sum and produces the final output (y_k) of the neuron. This activation function is usually non-linear and plays a very important role in the approximation of complex functions as well as the convergence of the learning process. Some of the most commonly used activation functions are the *Rectified Linear Unit* (ReLU), that gives as output the maximum between 0 and the input value, the *Logistic function*, that gives as output a value between 0 and 1 based on the input value, and the *Softmax function*, which is a generalization of the Logistic function to multiple dimensions. In this context, a DL model consists of the hyperparameters defining the structure of the NN, that is, how many layers, how many nodes in each layer and which type of layers, as well as the weights and biases associated to each NN node; these parameters are obtained by training using an appropriate loss function to achieve a meaningful outcome during the learning process.

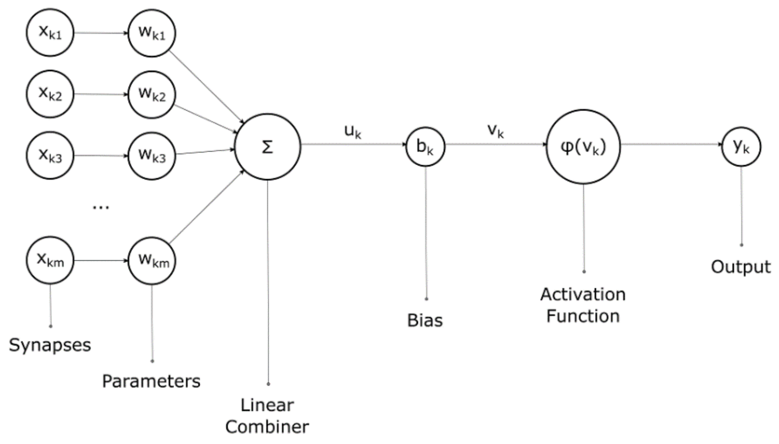

Figure 4: Model of a neuron [14].

Neural networks are made up of many of these neurons, all linked forming a connected graph that is organized in layers, which are groups of neurons usually at the same depth in the network and operating in parallel. Layers in NNs may be of different types; for example, when each neuron of one layer connects to all the neurons of the next layer, a fully connected layer is formed.

Depending on the type of graph formed by the neurons, a NN may be classified as a Feed-Forward Neural Network (FFNN) or a Recurrent Neural Network (RNN). In FFNNs, the inputs flow from the input layer to the output layer, without any cycles, which means that an output from one layer only affects the next layers and never the layers before. On the contrary, in RNNs, there are cycles in the network, so an output from one layer can not only influence the next layers but also influence the same layer itself, or even the layers before, thus giving this type of NN a kind of memory capability, as it can combine past information with current inputs to define the neuron output.

By combining many neurons together, layers are formed and, by combining many layers, neural networks are formed. In the following, the most relevant types of neural networks for video coding purposes will be reviewed.

Fully Connected Neural Network

A fully connected neural network (FCNN) is a type of feedforward neural network, where all the layers are fully connected meaning that each neuron is connected to all neurons from the previous layer. This type of NN has three distinct types of fully-connected layers: input, hidden and output. The input layer receives the data, e.g., image or video data, that may have been pre-processed and forwards it to the rest of the network. The hidden layers, which may be more than one and as many as needed, are all the layers between the input and output layers, which receive data and process it, to pass to the next layer. The output layer is the last layer in a NN and gives the final result in a form suitable for the task being solved, e.g., a label for an object detected in an image. Figure 5 represents an FCNN network with a single hidden layer.

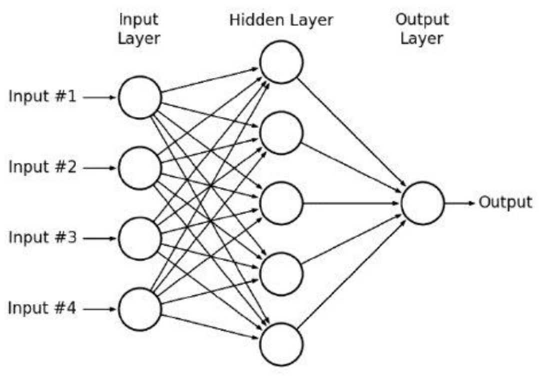

Figure 5: Architecture of a Fully Connected Neural Network with a single hidden layer [15].

Nowadays, FCNNs are the simplest type of neural networks and typically complement other more complex network types, e.g., acting like classifiers.

__Convolutional Neural Networks__

Convolutional Neural Networks (CNN) are useful to solve many image processing and computer vision tasks, such as image recognition, detection, segmentation, and classification. A CNN is a group of neurons, usually arranged in five different types of layers: input, convolution, pooling, fully connected and output. An example of this type of structure is represented in Figure 6.

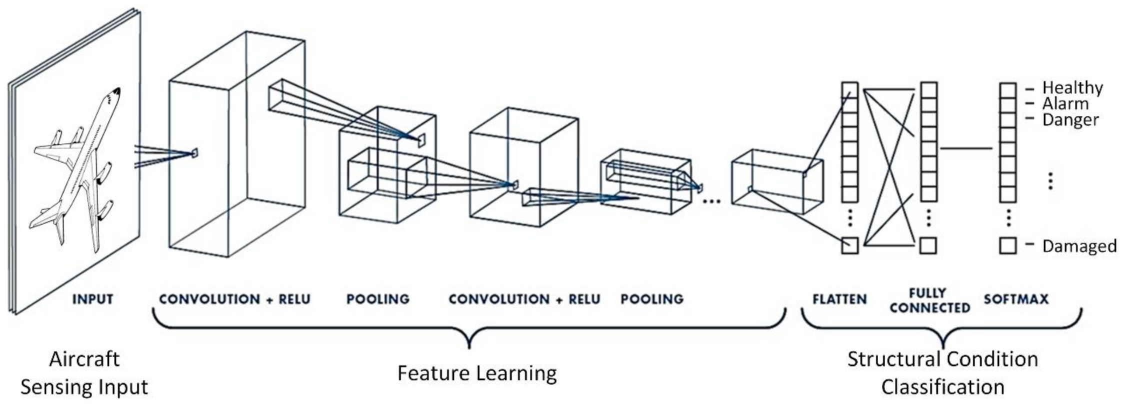

Figure 6: CNN architecture example with an image as input, two convolutional layers, two pooling layers and one fully connected layer [16].

Due to its importance, the convolutional and pooling layers are presented next.

Convolutional Layer

This type of layer is the most important layer in a CNN and is used to extract a set of low-level and/or high-level features, such as edges or objects in images. The convolutional layer is composed by a set of filters which are defined by *convolutional kernels*. A convolution is the application of a kernel to some input data, thus generating a filtered output. When a kernel is repeatedly applied to different parts of the input data (that may overlap or not), multiple outputs are generated, which together form the so-called *activation map* to be input to the activation function. The application of the convolution kernel consists of multiplying the input data values by the filter kernel values, and linearly summing all the values to get the output. The filter values, also called *parameters*, are learned weights. In a convolutional layer, the filter weights are the same for all filtered positions, notably to avoid an explosion on the number of these weights and thus on the size of the model. For images and video, the input data is usually a volume, corresponding to a 3D array with the three color components (usually RGB) for a 2D map (image or frame) or the output of a previous layer; however, if only the luminance is considered the input may be two-dimensional.

The convolutional kernel has a smaller size than the size of the full input data and is defined through a width and height, setting the so-called *filter support*. The kernel slides across the input according to a parameter called *stride*, which is the number of positions the filter skips before processing the next input position. When the kernel stride increases, the size of the activation map decreases. To apply the filter to the borders of the convolutional layer input data, it is necessary to perform padding, this means to extend the available input data beyond its dimensions; padding may simply correspond to the addition of zeros to create enough support for the filters being applied. Figure 7 illustrates two convolution operations, where the input data is a 4x4 matrix, the kernel has a height and width of 3 and moves with a stride of 1.

Each convolutional layer can use more than one kernel in parallel, thus generating one activation map for each kernel and producing an output volume with a depth corresponding to the number of kernels used.

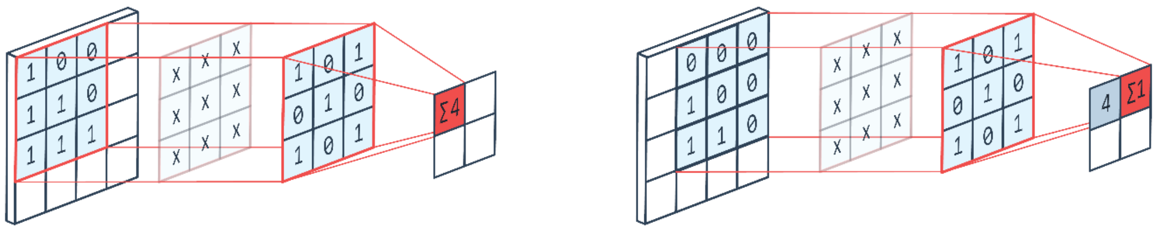

Figure 7: Representation of two convolution operations with stride 1 [17].

Pooling Layer

Pooling layers are typically used after convolutional layers to reduce the data volume and thus the number of weights in the next layers. This allows to reduce the computational power (and memory) needed to process the data and can also be used to extract dominant, high-level features. The most common types of pooling operations are the *max pooling* and the *average pooling*, both represented in Figure 8, although max pooling is the most popular pooling method. The max (or average) pooling returns the maximum (or average) value of a spatial window that is slid over the input layer. Naturally, the size and stride of this window defines how much reduction the output data has with respect to the input data. Figure 8 represents a 2x2 pooling filter that moves through the input with a stride of 2, i.e., two positions are skipped between each pooling. After pooling, the output data has smaller dimensions than the input data but remains with the same depth.

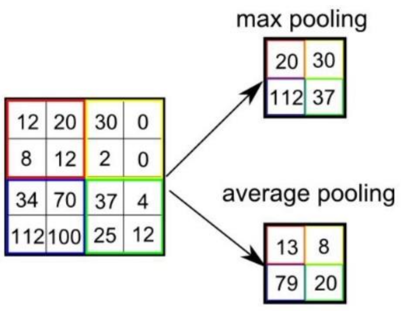

Figure 8: Pooling layer example with two different types of pooling, max and average [18].

Autoencoder

An autoencoder is a feedforward NN with two parts, an encoder and a decoder, separated by a bottleneck layer. The bottleneck forces the encoder part of the autoencoder (which performs the data analysis) to represent the input data with a lower dimensionality, the so-called *latent code, latent representation* or simply the *latents*. In this encoding process, the relationships and dependencies in the input data are exploited to produce compact latents, a compressed (usually lossy) representation.

The decoder part of the autoencoder (which performs the data synthesis) reconstructs the data from the latents. The encoder and decoder layers can be fully connected or convolutional layers. These networks can be trained for different tasks, such as denoising, segmentation and compression; in any case, an autoencoder produces a compact representation, which has a lower dimensionality compared to the original data representation. Figure 9 represents the architecture of an autoencoder, with the bottleneck layer represented in orange.

The so-called *loss function* used for training is an important element of an autoencoder since it guides the autoencoder to produce latents with specific properties. Usually, the loss function measures the distortion between the input and the output, and the training process objective is to minimize this loss. In a compression context, the loss function typically needs to also consider the rate as only minimizing the distortion would let the rate without control.

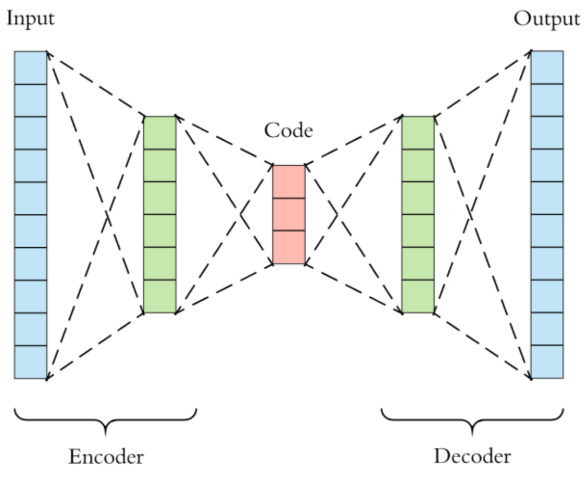

Figure 9: Architecture of an autoencoder neural network [19].

Generative Adversarial Network

A generative adversarial network (GAN) is a deep generative model, which is a deep model that can generate new data instances through unsupervised learning. Figure 10 presents the typical architecture of a GAN. A GAN is made up of two different networks, the so-called *generator* and the *discriminator*; they are trained jointly and are competing with each other in a special type of game, one always trying to deceive the other.

The generator, usually a CNN, is trained to create new data that should be similar to some input data examples, e.g., to produce new images that are considered realistic, starting from input data such as random noise or an original image, depending on the specific task at hand. On the other hand, the discriminator, which is a CNN including a fully connected layer, has to decide if the input data really looks realistic. Note that the discriminator does not know if its input data, e.g., images, come from the generator or are real examples. After the discriminator classifies its input data, the computed probability of the input being realistic is used to guide the learning process of both the generator (to create more realistic images) and discriminator (to distinguish better between real and 'fake' images). This sets some type of competitiveness between the generator and discriminator, leading to the capability to generate high quality data, like realistic images, at the end of the training process. This type of network has many applications such as high-quality image and video generation, music and speech generation, image enhancement and compression, and inpainting. In some image coding solutions [20], the generator can take the structure of an autoencoder (thus with a latents/rate bottleneck) that is trained with an appropriate loss function to replicate the original input image, with special attention to perceptual/realistic quality (trained with the discriminator); in practice, the discriminator only plays a role at training time to help the generator learning to create realistic 'decoded images'. This method capitalizes on the generator's capability to 'synthesize' data that looks authentic/realistic, while saving rate through the autoencoder bottleneck.

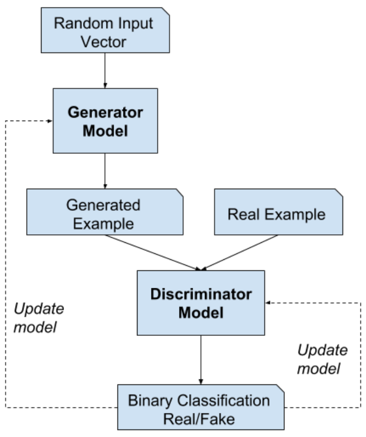

Figure 10: GAN architecture [21].

Recurrent Neural Networks

A recurrent neural network (RNN) is typically better at handling data evolving in time, like sequential or time series data, than a feedforward NN. In an RNN, past information can influence current outcomes by taking outputs derived from prior inputs and inserting them back into a network layer where current inputs are being processed. Usually, RNNs can be identified by their “memory” since data from previous outputs (iterations) may impact the current input and output (at some iteration). RNNs are able to model functions which have a dependency in time between the input and the output, thus creating cycles which give the network the capability to model dynamic temporal behaviors. Figure 11 shows a RNN architecture example, where the output of one layer is used as input to the previous layer. This means that the RNNs are iterative by nature and excel at tasks for which it is important to perform some sort of prediction using past observed events.

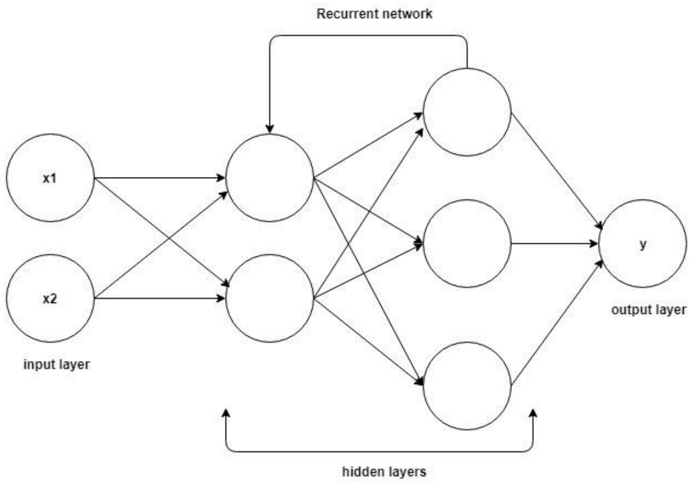

Figure 11: RNN architecture example [22].

The so-called *Long Short-term Memory* (LSTM) network is a specific type of RNN that is mostly used for natural language processing, speech recognition and many time-based data analysis problems, e.g., action recognition. This type of RNN was specifically created to improve the long-term memory of RNN networks because, given data with many time steps, e.g., frames, it becomes difficult for RNNs to recall earlier inputs. Therefore, LSTM cells are good at “remembering” inputs for long periods of time, if they are deemed important during training.

The building blocks of an LSTM network are its LSTM cells; each cell contains four layers interacting in a specific way, see Figure 12. Each yellow module in Figure 12 represents a node with its activation function, and each of them plays an important function in the cell, notably (from left to right): the *forget gate*, which decides which parts of the current input should be forgotten based on the previous hidden state; the *input gate*, that decides which values should be inserted into the new cell state and how much of each value should be added; the *candidate values,* which are to be inserted into the cell state; and, finally, the *output gate*, which decides what information from the cell state should be passed along into the hidden state. The black arrow running along the top of the cell is the cell state and carries the most important information along the cells; this information may change depending on the information output through each gate. The black arrow leaving upwards through the top-right part (and sideways through the bottom-right part) of the cell is the hidden cell state, which will be used in the next time step to make more informed decisions about the next input.

In the LSTM cell, the long-term memory is referencing the network weights that are learned in the training process while the short-term memory refers to the cell state of each cell, which changes at each iteration. This type of network was developed to deal with the vanishing gradient problem that happens in deep networks when the training gradients over several layers get too small to effectively update the weights being trained in the network, thus preventing the learning of long-term dependencies. The advantage of an LSTM cell over a neuron with a cycle in an RNN network is its memory behavior, that is, the capacity to retain information across many time steps, but also to forget useless information and add new relevant information, thus counteracting the vanishing gradient problem.

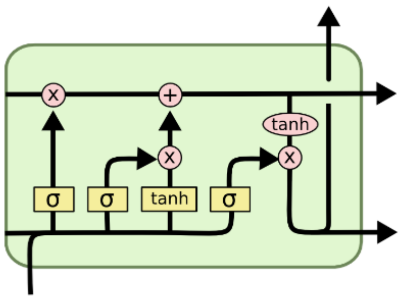

Figure 12: LSTM cell architecture example [23].

This type of LSTM cell cannot be applied directly to video coding, since the inputs it receives are one dimensional. To deal with three dimensional inputs, like video or images (divided among channels, e.g., RGB or YUV), the so-called *ConvLSTM cell* has been created [24], which replaces all the multiplication operations in an LSTM cell with convolutions, as it may be observed in Figure 13. Therefore, this cell can receive three dimensional tensors as inputs and maintain their dimensions, while extracting meaningful information with the convolutions.

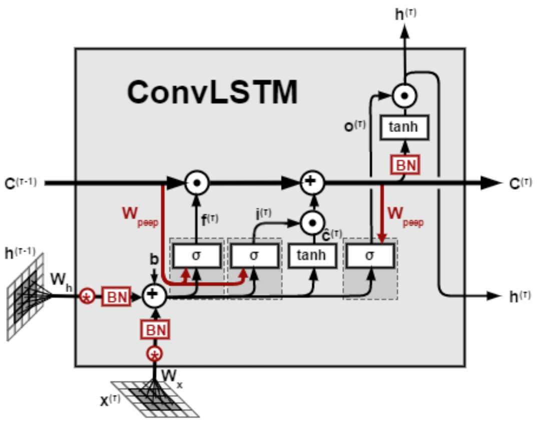

Figure 13: ConvLSTM cell architecture example [25].

3.2 Deep Learning-based Video Coding Solutions Classification

Several DL-based video coding solutions have been recently proposed and are available in the literature. To guide the reviewing of the most relevant DL-based video coding solutions, it is essential to identify their main commonalities, differences, and relationships, ideally by clustering them into meaningful classes. Thus, after studying and analyzing the DL-based video coding solutions in the literature, a classification ontology has been designed based on four classes which is proposed in the following:

- **Classical hybrid coding:** This class adopts an architecture similar to the conventional video coding architecture in existing video coding standard where each coding module is independently designed and performed using DL-based tools.
- **Temporal block joint coding:** This class jointly codes a fixed or adaptive length set of successive frames using and end-to-end trained DL-based model; the input data is a 3D block of frames instead of a succession of individual frames.
- **Motionless prediction/estimation-based coding:** This class individually codes a fixed or adaptive length set of frames bounded by two anchor/reference frames using DL-based models and without explicitly exploiting any type of motion data. In this class, the anchor/references frames are Intra (independently) coded and intermediate frames are interpolated or predicted without explicitly exploiting any type of motion data.
- **Motion-based prediction/estimation-based coding:** This class individually codes a fixed or adaptive length set of frames bounded by two anchor/reference frames using DL-based tools and explicitly exploiting some form of motion data. In this class, the anchor/reference frames are Intra (independently) coded frames and intermediate frames are interpolated or predicted while exploiting some form of motion data.

In the next section, representative DL-based video coding solutions are reviewed for each of the four classes defined above.

3.3 Reviewing Some Relevant Deep Learning-based Video Coding Solutions

In this chapter, five relevant DL-based video coding solutions from the literature will be reviewed, one for each (and by the same order) of the classes defined in the previous section, except for the last class, for which two DL-based video coding solutions will be reviewed. These specific coding solutions have been selected using as criteria their conceptual richness, compression performance, quality and completeness of the corresponding paper and software availability.

3.3.1 Video Coding with a Classical Hybrid Coding Architecture and Deep Learning-based Tools

A) Objective and Approach

At CVPR'2019, Lu *et al.* published a paper entitled "DVC: An End-To-End Deep Video Compression Framework" [26] proposing a video coding solution with the key objective to achieve the best possible rate-distortion (RD) performance adopting a classical hybrid coding architecture where most of the key modules are implemented using deep learning-based tools. A classical hybrid coding architecture is the type of architecture used in available video coding standards where temporal prediction tools are combined with (frequency) transforms and quantization to exploit the video temporal, spatial and perceptual redundancies.

The technical approach proposed in the DVC solution [26] incorporates neural networks into the classical video coding architecture by replacing the conventional tools in some key modules by deep learning-based tools. The motivation for this coding approach is to exploit neural network tools and associated characteristics to offer flexibility, while still taking advantage of the usual (temporal) predictive coding architecture. In the following, the term 'DVC model' will be used to designate the full set of used neural network architectures and associated weights, thus corresponding to the complete video codec (encoder and decoder) [1]. The DVC coding solution is labelled as end-to-end since all the neural networks in the DVC model are trained simultaneously, using a single loss function, as if they were a single neural network, i.e. end-to-end training.

B) Architecture and Walk Through

As previously stated, the DVC codec adopts an overall architecture similar to a classical video codec as shown in Figure 14 (left); however, the key coding modules are implemented using deep learning-based tools as shown in Figure 14 (right). In fact, the main modules are present on both the classical architecture (left) and the DVC architecture (right), just implemented in a different way.

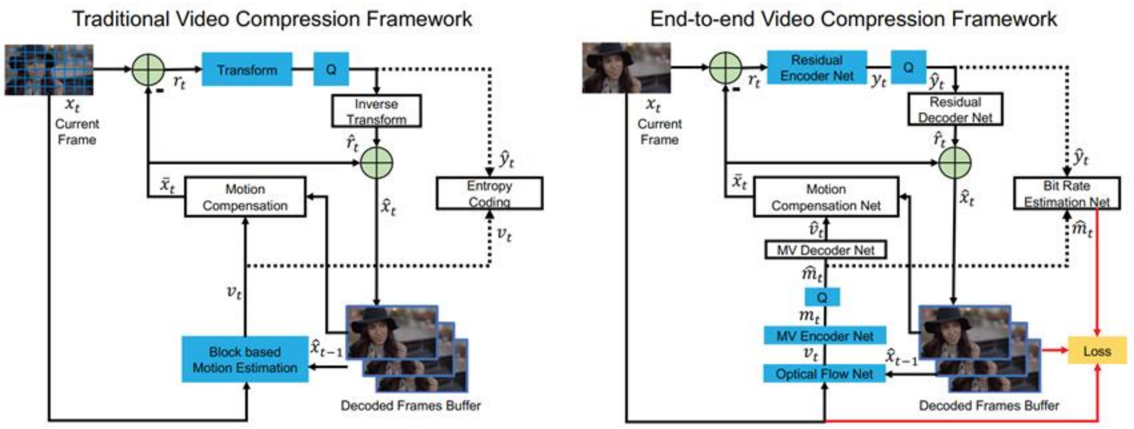

Figure 14: Traditional video codec framework (left); DVC architecture (right) [26].

In the encoding phase, the proposed DVC solution processes the original video frames according to the following walk through:

- **Optical Flow Net:** The DVC framework starts by performing motion estimation by estimating the optical flow between the previous decoded frame, and the current original frame. For this purpose, the CNN described in [27] has been adopted; this CNN extracts an optical flow, which provides very accurate motion information pixel-wise, unlike traditional motion estimation methods that typically work with motion vectors at (different size) block level.
- **MV Encoder-Decoder Net and (MV) Quantization**: Since the motion information created in the previous module needs to be efficiently coded, an MV Encoder-Decoder pair of modules is used to encode and decode the optical flow motion vectors in order the encoder uses for the prediction the same lossy decoded motion information that is made available to the decoder. To save bitrate, between the encoder and decoder sides, quantization is applied, thus reducing the precision of

the motion vectors. The decoded motion vectors coming out of the MV Decoder module are used by the DVC encoder for motion compensation to create the temporal prediction.

- **Motion Compensation Net:** With the decoded motion vectors and the previous decoded frame, the Motion Compensation Net module creates a frame prediction for the current frame using a CNN; further details will be provided in the next sub-section.
- **Residual Encoder-Decoder Net and Quantization:** After obtaining the predicted frame, the residual between the predicted and original frames is computed and encoded. The residual is processed by a non-linear transform [28], composed by several convolutional layers. The output of the residual encoder net is a latent representation, which is naturally quantized to reduce the rate [29]. The quantized residuals are sent to the DVC decoder, which is also included in the DVC encoder to obtain the final decoded frame. Thus, both the DVC encoder and decoder have the final decoded frame which can be used by both for future temporal predictions (when the next frame is coded).
- **Bit Rate Estimation Net:** Lastly, the Bit Rate Estimation Network [28] estimates the probability distribution for the latent representations to obtain the matching entropy and thus an ideal rate estimation, so that the RD performance results may be obtained.

Naturally, before the coding phase, the DVC model must be trained under appropriate conditions. In the training process, the most critical element is the so-called Loss Function which is based on a rate-distortion (RD) criterion. The end-to-end distortion is measured with the mean square error RGB (MSE-RGB, that computes the squared difference between the corresponding Red, Green and Blue samples and computes the overall mean) and the rate used is estimated by the Bit Rate Estimation Net module, using the entropy. The loss function includes a Lagrange multiplier (λ) which serves to adjust the trade-off between the rate and distortion, thus corresponding to different RD points. For the DVC model training, the loss function is computed using as inputs the original frame, the previous decoded frame (which are MSE-RGB compared) and the estimated rate for the coded bitstream.

To guarantee that the DVC model is end-to-end trained with a single loss function, every operation performed in the overall codec needs to be differentiable to allow backpropagation. For this reason, since quantization is a non-differentiable operation, it is replaced during the training step, by adding uniform noise to the latent representations [29]; naturally, in the coding phase, regular quantization is performed since no backpropagation is performed anymore.

C) Main Novel Tools

In the DVC framework, a couple of tools deserve special attention due to their originality or contribution for the codec performance, notably:

<u>MV Encoder-Decoder Net</u>

The module used to encode and decode the motion vectors is presented in Figure 15. CNN-based coding is adopted to encode the optical flow motion vectors since adopting a traditional motion vector coding method would take up too much rate, especially because one motion vector per pixel (and not per block as usual in classical codecs) is produced by the optical flow net. This CNN uses convolutional layers interspersed with non-linear transform functions, called Generalized Divisive Normalization

(GDN) [29], to be more efficient. After computing the latent representation, the latent variables are quantized since they must be rate efficiently sent to the DVC decoder. The quantized latent representation is decoded by performing the appropriate inverse convolutional operations and the inverse GDN (IGDN).

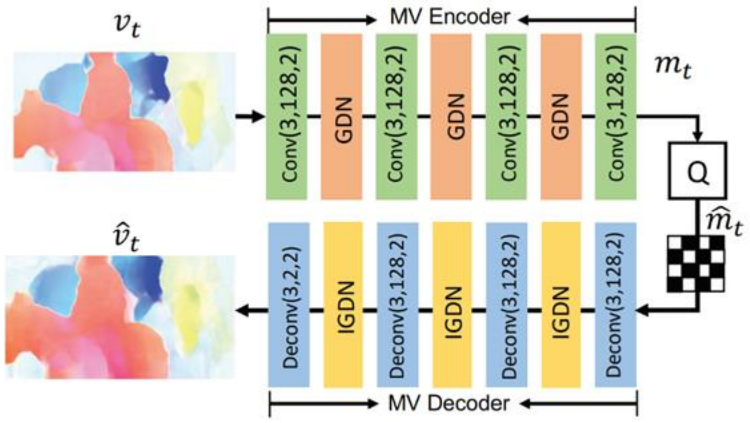

Figure 15: MV Encoder-Decoder Net module architecture [26].

<u>**Motion Compensation Net**</u>

The Motion Compensation Net module starts by warping the previous decoded frame using the (decoded) optical flow motion vectors to offer a temporal, pixel-wise motion compensation prediction to the current frame, which naturally is not perfect and typically includes some artifacts. To remove these artifacts, the prediction frame is given to a CNN, jointly with the previous decoded frame and the (decoded) motion vectors, to refine the warped frame to produce a better frame prediction after motion compensation, see Figure 16. This CNN plays the role of the in-loop denoising filters in traditional video codecs. The predicted frame produced by the Motion Compensation Net is used to compute the residual that will be sent to the decoder to be added to the decoder created frame prediction.

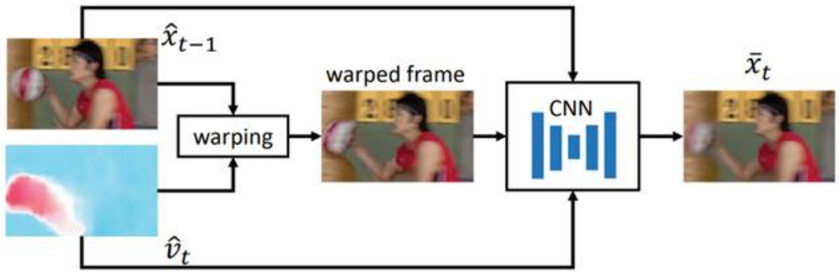

Figure 16: DVC's Motion Compensation Net module architecture [26].

D) Performance Assessment

Finally, the DVC performance is assessed, notably including:

- **Dataset**: The DVC model was trained on the Vimeo-90k dataset [30], which includes 89800 clips covering a large variety of scenes and actions, with a spatial resolution of 448x256 pixels; the Vimeo-90k dataset was designed for video processing tasks like temporal frame interpolation and video denoising. On the coding phase, the Ultra Video Group (UVG) dataset [31] and the HEVC test sequences were used [11]. The UVG dataset includes 16 videos with spatial resolution of 3840×2160 pixels and a framerate of 50 or 120 frames per second.

- **Objective quality metrics**: The quality was objectively assessed both with PSNR and MS-SSIM; note that the training was performed with MSE-RGB, which matches the PSNR metric but not the MS-SSIM metric.
- **Benchmarks**: The DVC codec was compared with a solution labelled as Wu_ECCV2018 [32] and the two most recent video coding standards, i.e. H.264/AVC [10] and H.265/HEVC [11]. Wu_ECCV2018 is a deep video codec developed by Wu *et al.* [32], which codes video based on successive frame interpolations within a group of pictures, where the first and last frames are intra coded using deep image coding techniques; the in-between frames are predictively coded through a residue computed after frame interpolation based on block motion estimation.

Figure 17 shows an excerpt of a decoded frame obtained with the various video codecs under comparison. The DVC solution designers claim that it achieves better subjective quality since the decoded image appears smoother than those obtained with H.264/AVC and HEVC. Moreover, the DVC codec attains slightly better MS-SSIM scores than both the coding benchmarks, for lower bit per pixel rate.

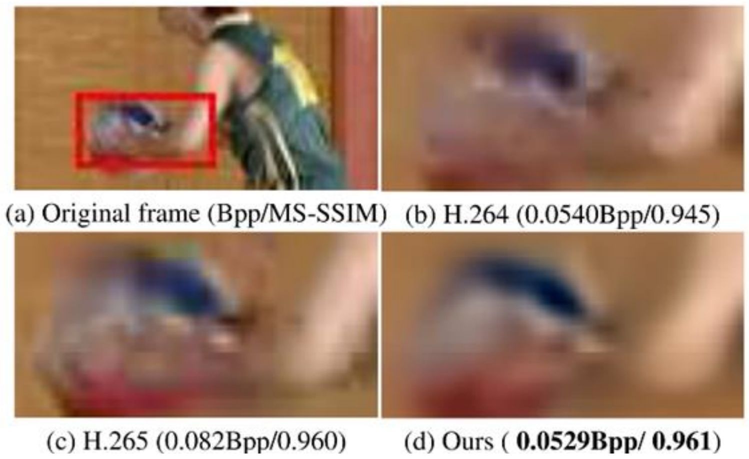
(a) Original frame (Bpp/MS-SSIM)
(b) H.264 (0.0540Bpp/0.945)
(c) H.265 (0.082Bpp/0.960)
(d) Ours (**0.0529Bpp/ 0.961**)

Figure 17: Visual quality for an excerpt of a decoded frame using several coding solutions (in brackets, the bits per pixel and the MS-SSIM score) [26]. (a) Original frame. (b) Excerpt of frame coded with H.264/AVC. (c) Excerpt of frame coded with HEVC. (d) Excerpt of frame coded with DVC.

To obtain enough RD points, four DVC models were trained using different lambdas in the loss function. Figure 18 shows the obtained RD curves for three different test videos, using both PSNR and MS-SSIM. The DVC codec outperforms H.264/AVC for almost every case, both in terms of PSNR and MS-SSIM; moreover, it performs better than HEVC for the higher rates and worse for the lower bitrates. In comparison to the Wu_ECCV2018 codec, DVC is better for all rates, both for PSNR and MS-SSIM, although results are only reported for a single test video.

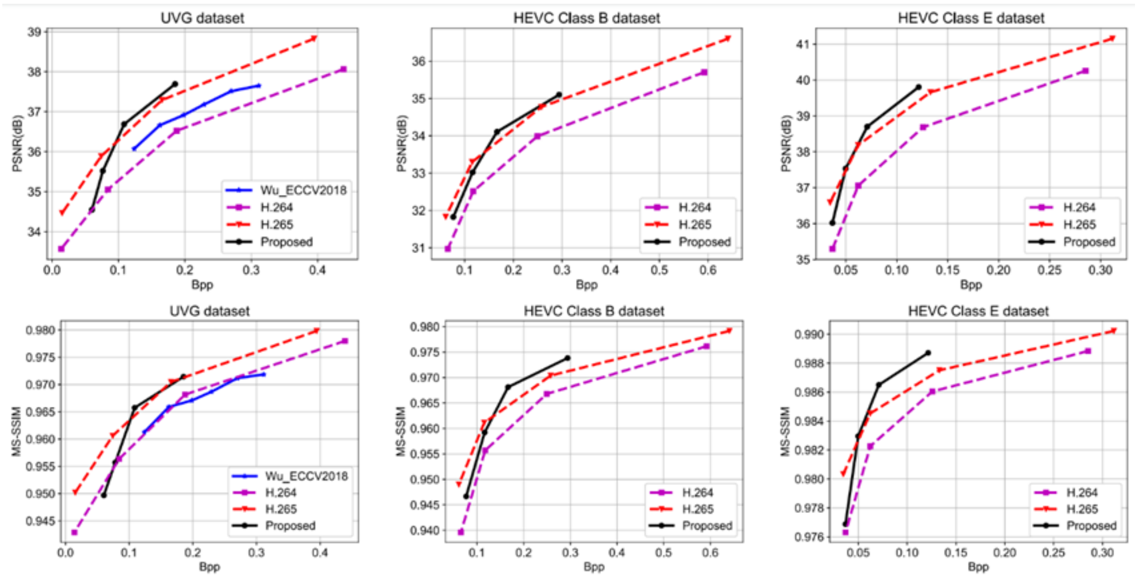

Figure 18: RD performance for the various video codecs under comparison [26].

Finally, it should be stated that the DVC framework may still be improved if the current tools for the various modules are replaced by more efficient solutions/models.

3.3.2 Video Compression with Rate-Distortion Autoencoder

A) Objective and Approach

In 2019, at ICCV, Habibian *et al.* proposed a novel video coding solution based on a deep generative model, a powerful way of learning any kind of data distribution using unsupervised learning, in the paper entitled "Video compression with rate-distortion autoencoders" [33]. The key objective of this video codec is to achieve the best possible rate-distortion (RD) performance by employing a 3D autoencoder, one of the most commonly used types of generative models, and an autoregressive prior, which are jointly trained . This coding approach takes several frames together and codes them jointly as if they were a 3D block, with two spatial dimensions while the third dimension corresponds to time. This type of coding solution has the drawback of increased delay (difficult to accommodate in some applications, such as videoconferencing), since the decoded video is not available frame by frame as usual, but 3D block by 3D block.

The autoencoder used in this method is called a Rate-Distortion Autoencoder since it learns to produce latents guided by a loss function which combines a rate and a distortion term. For entropy coding, an autoregressive model attempts to predict an outcome based on past data; this is especially efficient in a video coding context given the very high temporal correlation between frames, and therefore within a 3D block of successive frames; in this context, it is used as an autoregressive prior, for entropy coding.

In the following, the term 'RD Autoencoder model' will be referring to the whole 3D autoencoder (encoder and decoder) and the autoregressive prior, together with their corresponding weights.

B) Architecture and Walk Through

Figure 19 shows the architectural design of the proposed video coding model [33]. Both parts of the autoencoder are represented in green. The encoder side creates the 3D block of frames (represented as *x*) and encodes this data as latent variables (*z*), which are then converted to a bitstream (*b*) using adaptive arithmetic coding (AAC) and an autoregressive prior (entropy code model). The decoder side receives the bitstream and reconstructs the latent variables, which are after decoded by the decoder part of the autoencoder, thus transforming the entropy decoded latents into the reconstructed 3D block of frames ($\hat{x}$).

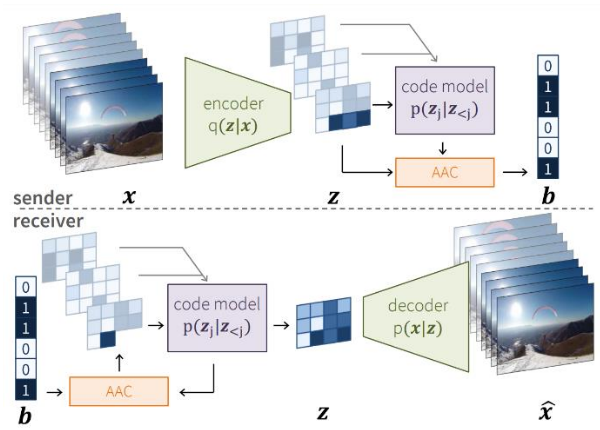

Figure 19: Rate-distortion autoencoder architecture [33].

The coding process proceeds as follows:

- **Encoder**: The encoder takes each 3D block of frames and transforms the sample data into latent variables ($\tilde{z}$), which have lower dimensionality.
- **Latents Quantization**: To reduce the rate, the latents ($\tilde{z}$) created at the encoder are quantized and discretized using a learned codebook. So, the obtained coded latents (*z*) are discrete variables and represent the important information to be transmitted. The objective is here to represent in a lossy way the most important visual information.
- **Entropy Coding**: The quantized latents are entropy coded using an autoregressive prior, which serves as the probability model (also referred as code model) for the arithmetic coding engine. While there are several strategies to obtain the prior, a conditional PixelCNN [34] was used here to model each latent element, *z*, based on past latent elements (from the same frame or past frames). Adaptive arithmetic coding is used to obtain the final bitstream as a function of the prediction and latents.
- **Decoder**: The decoder side receives the bitstream and, using the same prior and the bitstream information, recreates the latents. The entropy decoded latents are then input to the decoder part of the autoencoder, to obtain the reconstructed 3D block of frames.

To further understand the extent to which learning-based video coding can be beneficial, three different extensions were also proposed. The first extension was *semantic compression*, where the RD Autoencoder model is trained to allocate more rate (and thus quality) to designated objects, such as people, for example. To achieve this, the RD Autoencoder model is trained to prioritize rate for the regions of interest and thus, the objects in the frames were divided into two categories: the foreground, with more important objects, and the background, with less important objects. The frame segmentation ground truth was created using the Mask R-CNN [35]. The second extension was *adaptive compression*, where the RD Autoencoder model is trained for the specific target domain in which it will be used, thus achieving higher compression performance than non-adapted models. For example, for the visual surveillance domain, a model can be obtained for cameras placed at some fixed locations where the visual scene acquired does not change significantly; the videoconferencing domain can benefit from such adaptive compression approach. The third and last extension was *multimodal compression*, where the RD Autoencoder model also compresses visual data obtained from other modalities, like multi-view arrays of cameras and depth range sensors, where the information to code has different characteristics compared to a regular 2D monocular sensor. The authors state that the same codec can be used for many other modalities, without any change, by just using a model that is trained using a well-suited dataset. In the experiments performed, the proposed solution codes data from multiple cameras where each camera view is treated as one additional channel, so all the (multiple) views are coded together in the same 3D block.

The RD loss function used to train the RD Autoencoder model employs an MS-SSIM based distortion metric while the rate is measured using the log-likelihood of the latent probabilities. To control the trade-off between rate and quality in the loss function, a parameter β is used.

C) Main Novel Tools

This section describes the main innovative tools used in this video coding solution:

<u>3D Block of Frames based Coding</u>

Unlike most video codecs, this codec jointly codes multiple frames organized in a temporal block of frames, named a 3D block. This implies that this deep learning -based coding approach does not have to explicitly deal with motion and thus does not include any processing layers or methods to deal with motion, while still taking advantage of the high temporal correlation, given that all the information in the block is being compressed. While this approach may work well for more temporally stable content, video with high motion may only be efficiently coded using shorter 3D blocks.

<u>Autoregressive Prior</u>

In this coding solution [33], three different priors are considered for entropy coding purposes: an *unconditional prior*, where the code model only depends on the latents already analyzed in the current frame, implemented with the so-called PixelCNN [34]; a *frame-conditioned prior*, where the code model is based on the latents from the frame before, and the latents from the current frame, implemented with the so-called Conditional PixelCNN [34]; and a *Gated Recurrent Units-conditioned prior* (GRU), where the code model is obtained using relevant information from all previous frames summarized using a recurrent neural network and the latents from the current frame. A good prior helps to save rate when arithmetic encoding the latents and, in this case, a conditional prior can take advantage of the high temporal correlation between the frames in each 3D block.

D) Performance Assessment

The RD Autoencoder model performance assessment was done in the following training and test conditions:

- **Dataset**: To train the RD Autoencoder model, the Kinetics dataset [36] was used. The videos used have more than 720x720 spatial resolution, and the model was only trained on the first 16 frames. For comparison with the benchmark codecs defined in the following, the Ultra Video Group (UVG) dataset [31] was used, which includes videos with 1920x1080 spatial resolution and 50 or 120 frames per second. To train and test the model for the semantic compression extension presented above, the Human Activity dataset containing 1257 videos of real-world people doing mundane activities was used; from these, 1087 videos were used for training and 170 for testing. For the adaptive compression extension, an internal dataset, named Dynamics, was used for training and testing, consisting of 5 clips with a view from a car driving in different highways at different times of the day; four of these videos last 20 minutes and were used for training, while 1 lasts 14 minutes and was used for testing. Lastly, for multimodal compression, the Berkeley MHAD dataset [37] was used; it contains videos of 11 different human actions, filmed with four multi-view cameras; each action is repeated 5 times per participant, so this compression mode was trained on four repetitions and evaluated on the 5th.
- **Objective quality metrics**: The quality metric used for objective performance assessment was MS-SSIM.
- **Benchmarks**: This model was benchmarked against several state-of-the-art video coding solutions, including the Lu *et al*. DVC codec [26], an end-to-end trained deep learning-based video codec which replaces all the classical video coding modules with deep-learning modules, reviewed in Section 3.3.3; the Wu *et al*. video coding solution [32], which compresses the video data by performing repeated image interpolation; and, finally, the H.264/AVC [10] and H.265/HEVC [11] standard codecs.

Figure 20 shows the RD performance comparison between the RD Autoencoder model and several state-of-the-art coding solutions. The RD Autoencoder model outperforms both the learning-based video codecs (Lu *et al.* and Wu *et al*.) using the MS-SSIM metric, for every tested bitrate. Compared to the standard codecs, the RD Autoencoder model provides worse quality than both for lower bitrates but performs on par with them for the higher tested bitrates.

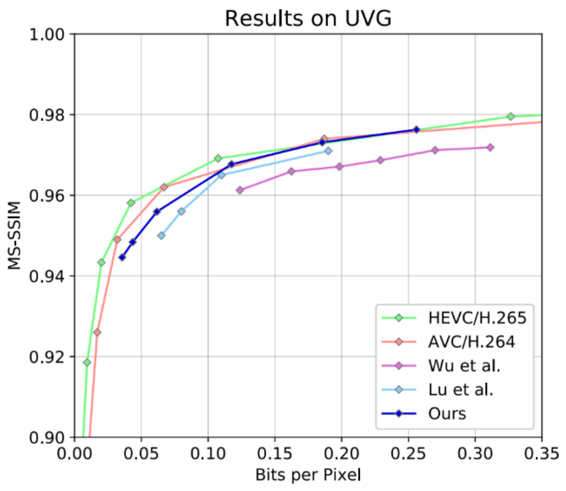

Figure 20: RD performance for the video codecs under comparison [33].

The RD Autoencoder model was also trained and tested for the three coding extensions described above. The results in Figure 21 show that, in general, the RD Autoencoder codec performs better with the extensions than without. Note that, for codecs with extensions, namely for adaptive compression, a generic model was fine-tuned with a dataset of a similar domain and, for multimodal compression, a multi-view dataset was used for training.

The semantic compression extension behaves as expected, i.e., allocates more rate to the foreground objects, which are generally more noticeable, and less rate to the background objects. However, while it does not outperform either video coding standards, it performs on par with them for some bitrate points.

The best improvement is observed for the multimodal compression extension, evaluated on the Berkeley MHAD dataset. Here the RD Autoencoder model outperforms both the H.264/AVC and H.265/HEVC video coding standards by far, as well as its own unimodal coding mode, as expected. The adaptive compression extension also outperforms both video coding standards.

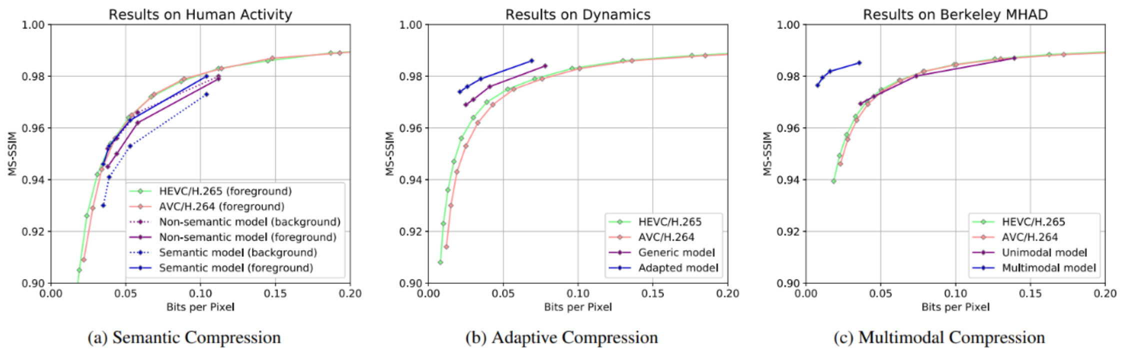

Figure 21: RD performance for the three coding extensions, evaluated with relevant datasets. Semantic compression (a); adaptive compression (b); and multimodal compression (c) [33].

3.3.3 Learning Image and Video Coding through Temporal Energy Compaction

A) Objective and Approach

In 2019, at CVPR, Cheng *et al.* proposed, in the paper entitled “Learning Image and Video Coding through Temporal Energy Compaction” [38], an image coding solution based on an autoencoder, which was extended to video coding by adding an interpolation loop. The proposed coding approach is based on a convolutional autoencoder architecture which encodes Groups of Pictures (GOPs) of variable sizes bounded by I-frames independently coded; the GOP middle frames are coded as residues after temporal interpolation although without explicitly using motion information, which is a key difference from other solutions using motion-based temporal interpolation/prediction.

In the following, the ‘image compression model’ will refer to the whole architecture of the image compression solution proposed in [38] with its corresponding weights, and the ‘video compression model’ will refer to the whole architecture of the video coding solution proposed in [38], along with all the corresponding weights.

B) Architecture and Walk Through

In Figure 22, both the image (left) and video (right) coding architectures are shown. The video compression model architecture builds on the image compression model by adding some blocks to deal with time, notably a motionless interpolation-based residue loop to encode the video. One of the key ideas exploited is the introduction of a spatial energy compaction-based criteria into the loss function to achieve higher coding efficiency. This criterion leads to an analysis transform which outputs latents where their energy is concentrated into a few values.

The networks and quantizer used by the image compression model to code the I-frames are also used by the video compression model to code the middle GOP residual-based frames. The video compression model divides a video into GOPs of variable size, depending on their temporal redundancy; when less temporal activity exists, the GOPs may be longer and vice-versa. The first and last frames of each GOP are taken as anchor frames, or I-frames, like in traditional architectures. After the frames in the GOP are successively interpolated based on the previously reconstructed neighboring frames, a residual is computed. This residual between each interpolated frame and the corresponding original frame is quantized and entropy coded. This process iterates until all the middle frames in the GOP are interpolated and their residues computed, quantized and entropy coded.

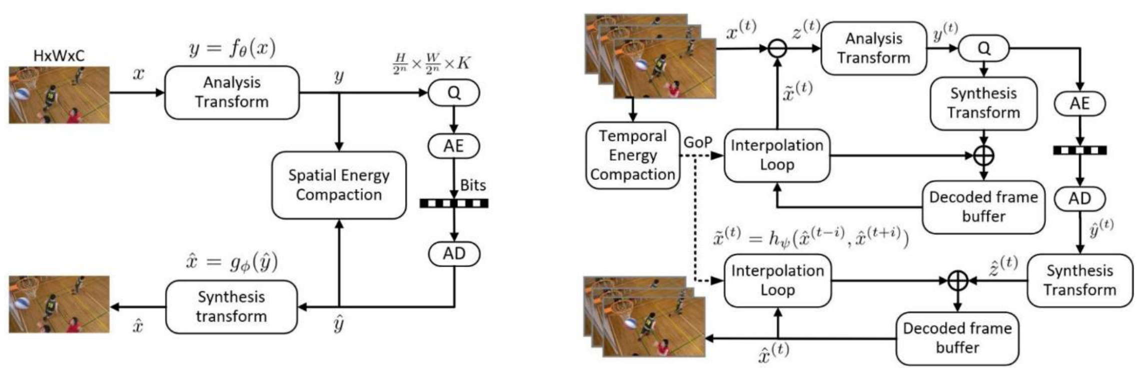

Figure 22: Learning-based image compression model (left); Learning-based video compression model (right) [38].

The coding process proceeds as follows:

- **Analysis Transform and Synthesis Transform**: The analysis and synthesis transform modules are two convolutional neural networks used to compress and decompress information and appear in both the image compression solution and video compression model. In the image compression model, the analysis module takes the original images and transforms them into a latent representation, while the synthesis module takes the quantized latent representation and obtains the reconstructed images. In the image compression model, the input of the analysis transform (*x*) and the output of the synthesis transform ($\tilde{x}$) correspond to the original and reconstructed images, respectively. In the video compression model, the input (z) and output ($\tilde{z}$) correspond to the original and the reconstructed residual between the current frame and the interpolated frame, respectively. An RD loss function is used to train these networks, with a parameter λ to control the tradeoff.
- **Quantization**: The quantization module (Q) is present both in the image compression model and the video compression model. During the training stage where differentiability has to be guaranteed for backpropagation, the quantization is performed by adding uniform noise [29]; however, in the coding phase, regular quantization is performed since no backpropagation is performed anymore.
- **Entropy Coding:** Entropy coding is performed with an AE (AE and AD in the architecture). For bitrate estimation, the entropy model of [28] is used to produce the estimated entropy.
- **Spatial Energy Compaction**: This module corresponds to the computation of a penalty term for the loss function used during the training of the analysis and synthesis networks. This term expresses the spatial energy compaction capability of the latent representation produced by the analysis network, i.e., if the produced latent code has most of the energy in a few elements. The introduction of this energy compaction property was inspired by conventional transforms (such as DCT) which also compact the energy into a few coefficients.
- **Interpolation Loop**: This module corresponds to an interpolation network that takes two anchor frames (I-frames or not) and interpolates the middle frame between them. This interpolation is formulated as a convolution over two reference frames [39] and no motion data is explicitly

involved. This network appears at both the encoder and decoder side of the video compression model to guarantee that the interpolated frames are the same on both sides and thus the encoder and decoder are synchronously creating interpolations and associated residues.

- **Temporal Energy Compaction**: As previously mentioned, each video is segmented into GOPs of different sizes depending on the temporal redundancy. This module does not explicitly use motion data and guarantees that a GOP does not span over too many frames when the temporal activity does not allow for an efficient interpolation. The temporal energy compaction module selects the most appropriate of three different GOP sizes (2, 8 and 16) along time.

The training loss function includes the common distortion and rate terms as well as a penalty related to the (spatial/temporal) energy compaction terms, where λ controls the RD tradeoff. When training the video compression model, the pretrained models from [39] were used for the interpolation loop.

C) Main Novel Tools

This section describes in more depth the key tools in the video compression model video coding solution.

Spatial Energy Compaction

For high compression efficiency performance, good energy compaction is necessary. After the analysis transform module, the latents data occupy *K* spatial channels, very much like in a subband coding system. Following some mathematical manipulation, the energy distribution of these channels (A_k) and the impact of the quantization on the reconstructed error for each channel (B_k) can be computed. After formulating the optimum rate allocation [40] for this problem, and considering a rate constraint, the minimized reconstruction error is proportional to

$$\prod_{i}^{K-1} A_k B_k \,. \tag{1}$$

If (1) can be minimized, then the spatial energy can be optimally compact. Thus, a penalty term is added to the RD loss function, depending on A_k and B_k, compacting most of the energy onto just a few latent channels, also considering the quantization error impact. The impact of this penalty on the loss function is weighted by β.

Temporal Energy Compaction

This module starts by computing the difference between two successive I-frames, initially situated 16 frames apart. After, the entropy (H_T) of this residue is computed and considered as a measure of temporal activity, which was used to obtain a decision in terms of the GOP size. If the entropy is high, then this GOP spans a set of frames with high temporal activity (likely high motion); if the entropy is low, then the temporal activity (also motion) is also low. The size of the GOP is given by

$$GOP\ size = \begin{cases} 2, & U \le H_T \\ 8, & L \le H_T < U \\ 16, & H_T < L \end{cases} . \tag{2}$$

Thus, (2) gives the size of the GOP based on the computed entropy, and whether it is above, below, or between the upper (*U*) or lower (*L*) bound. If the entropy is below *L*, then the GOP will include 16 frames, the selected default size; however, if the entropy is above *U*, the I-frames used for interpolation cannot achieve high quality interpolation, so GOPs of 2 frames are used, and the interpolation is performed for just one frame using the two neighboring I-frames; if the entropy is in between both of these thresholds, then the GOP will have half the size compared to the largest GOP of 16, this means GOPs with 8 frames.

D) Performance Assessment

To assess the video compression model performance, the following conditions were used:

- **Dataset**: For testing, the Video Trace Library (VTL) dataset [41] was used; the VTL dataset includes 20 videos with 352x288 spatial resolution and 8 test sequences with 832x40 and 416x240 spatial resolution.
- **Objective quality metric**: The popular MS-SSIM quality metric was used.
- **Benchmarks**: The video compression model performance was compared with the H.264/AVC, H.265/HEVC and the MPEG-4 Part 2 (Visual) standard codecs.

Figure 23 shows the RD performance obtained for the 8 test sequences in the VTL dataset. As shown, the video compression model always outperforms the MPEG-4 Part 2 standard. For most video sequences, video compression model performs on par with the H.264/AVC standard, usually showing better RD performance for lower bitrates; for the *Basketball Pass 416x240* and *BQ Square* video sequences, it clearly outperforms the H.264/AVC standard. Comparing with H.265/HEVC, the video compression model has lower performance for all but two sequences (*Basketball Pass 416x240* and *BQ Square*), although it may be competitive for videos with higher spatial resolution, notably for the lower bitrates.

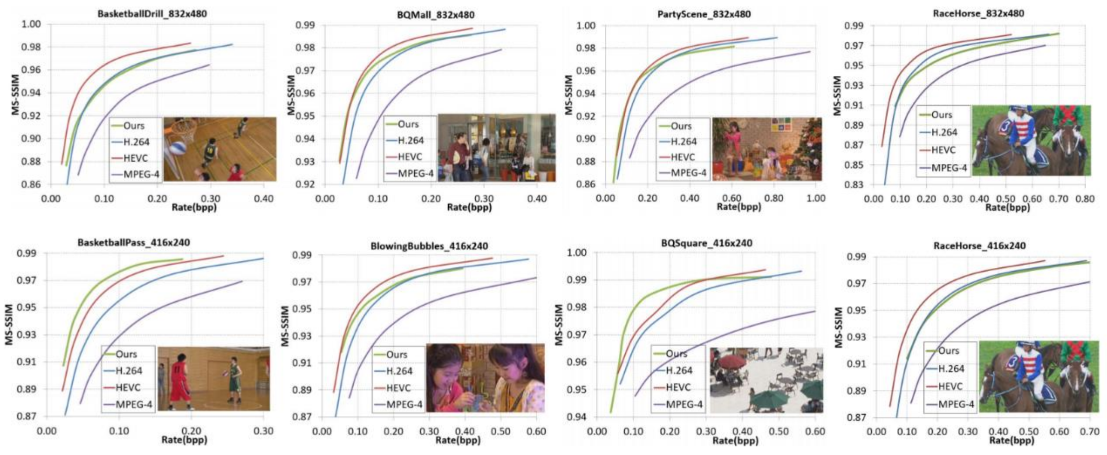

Figure 23: RD performance of the proposed video compression model and benchmarks [38].

3.3.4 Learning for Video Compression with Hierarchical Quality and Recurrent Enhancement

A) Objective and Approach

At CVPR'2020, Yang *et al.* published the paper entitled "Learning for Video Compression With Hierarchical Quality and Recurrent Enhancement" [42] proposing a video codec with three hierarchical quality layers and a recurrent enhancement network, which allows improving the RD performance. For this video codec, a new architecture, different from traditional methods, is proposed, as it performs motion-based frame prediction using four key tools, three of them neural network-based. The encoder codes different frames with three different quality levels (and subsequently different compression and rates), while the decoder uses a recurrent enhancement neural network to enhance the lower-quality frames by exploiting the temporal correlation and the better quality invested in some decoded frames; in this way some rate may be saved with the lower quality coded frames, while still achieving good quality for these frames by post-processing after decoding through the enhancement module. In the context of this codec, the 'Hierarchical Learned Video Compression' (HLVC) model corresponds to the full neural network architectures as well as the corresponding weights, present both at encoder and decoder.

B) Architecture and Walk Through

The HLVC model architecture (which is also the codec architecture) is shown in Figure 24. In this codec, the input video is divided into groups of pictures (GOP), each with 10 frames. In Figure 24, the white frames (x_i) represent the original video frames; the red, orange and yellow frames (x_i^C) the intermediate decoded frames for the three quality hierarchical layers; the green frames (x_i^D) the final decoded frames after enhancement (decoder side only); and the black and white strips the bitstream itself. In each GOP, the first and last frames are encoded with higher quality using an image coding state-of-the-art solution [43], [44]; this constitutes the 1st quality hierarchical layer. Then, the frame in the middle of the GOP is encoded using a bi-directional neural network, using the first and last GOP decoded frames as anchors, targeting a medium quality, thus leading to the 2nd quality hierarchical layer. The remaining frames belong to the 3rd quality hierarchical layer and are encoded with the lowest quality, using motion information and taking as anchor only one of the neighboring already decoded frames. For the 2nd and 3rd layer frames, motion maps and residuals are coded and sent to the decoder.

At the decoder side, the HLVC model takes the first layer decoded frames, the motion maps and the residuals associated to the remaining frames to produce all the intermediate decoded frames. Then a separate neural network is used to enhance the intermediate decoded frames' quality, thus obtaining the enhanced decoded frames.

During HLVC model training, the frame level distortion is measured with two objective quality metrics, PSNR-RGB and MS-SSIM; therefore, two different HLVC models are obtained. When training the HLVC model with PSNR-RGB, the distortion metric is the Mean Square Error RGB (MSE-RGB) while, when training with MS-SSIM, the distortion metric is 1 – MS-SSIM.

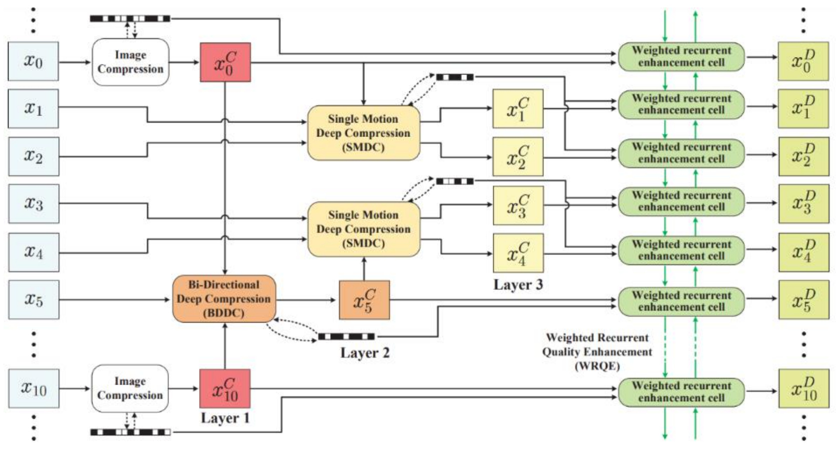

Figure 24: Hierarchical Learned Video Compression model architecture, exemplified for the first GOP [42].

The coding walk through with the HLVC codec/model evolves as follows:

- **Layer 1 - Image Compression**: The first codec layer regards the coding of the reference frames (first and last GOP frames) using a state-of-the-art image coding solution. These frames are traditionally referred as I-frames and provide error resilience and random access at the cost of higher bitrates, especially to reach higher quality. The intermediate decoded versions of these frames are represented in red in Figure 24. The image coding solution used for the 1st layer frames depends on the trained HLVC model, notably which distortion metric was used for training. For the PSNR-RGB trained HLVC codec, the Better Portable Graphics (BPG) image codec [43], which is based on a subset of the HEVC video coding standard tools, is used. For the MS-SSIM trained HLVC model, the image coding solution is the one proposed in Lee. *et al* [44], which includes a non-linear transform (based on [28]) and an advanced context-adaptive model.
- **Layer 2 - Bi-Directional Deep Compression (BDDC):** This Bi-Directional Deep Compression network corresponds to the second layer codec and encodes the frames in the middle of the GOPs by applying a bi-directional prediction neural network based on optical flow motion maps, using the first and last GOP decoded frames as anchors. The intermediate decoded frame is represented in orange in Figure 24.
- **Layer 3 - Single Motion Deep Compression (SMDC)**: The last hierarchical layer uses the Single Motion Deep Compression network for coding. This network encodes all the remaining frames, using as anchors the frames decoded in the previous two hierarchical layers. It encodes two frames at a time while using only the temporally closest decoded frame as anchor, i.e., if the network is encoding the frames number 1 and 2, it will only use the intermediate decoded frame 0 as anchor; if it is encoding frames 3 and 4, it will only use the intermediate decoded frame 5 as anchor and so on. Only one motion map is used to encode each pair of frames since these frame pairs are considered to be highly correlated. The intermediate decoded versions of these frames are represented in yellow in Figure 24; these are the frames encoded with lowest quality, thus also those consuming less bitrate.

- **Weighted Recurrent Quality Enhancement (WRQE)**: This network is present at the HLVC model decoder side and acts as a post-processing filter to further enhance the quality of the intermediate decoded frames for all three quality layers. The network receives the intermediate decoded frames and, by taking advantage of the higher quality decoded frames (when available) and the high temporal correlation, improves the quality of all the intermediate decoded frames; the enhancement is expected to be larger for the lower quality encoded frames.

The training loss function for all the encoder networks includes both rate and distortion terms, with a hyperparameter (λ) controlling the RD trade-off between these terms. The current HLVC model does not include an entropy coding module; instead, the final bitrate is estimated using an appropriate density model [29].

To guarantee that the different networks code the frames in the three layers with different quality, each network (BDDC and SMDC) is trained with a loss function using different values of λ, thus forcing higher or lower qualities (at higher or lower rates). The loss function to train the WRQE network at the decoder only includes a distortion parameter, since it only needs to be trained to improve the quality as the bitrate is irrelevant at this stage since this network only plays a post-processing role at the decoder.

C) Main Novel Tools

This section describes the main HLVC model tools, especially those that are novel and contribute more significantly for the overall HLVC codec RD performance.

Bi-Directional Deep Compression (BDDC)

The BDDC network architecture is presented in Figure 25 and is responsible for coding the middle frames in the GOP (2nd hierarchical layer). This network takes first the intermediate decoded anchor frames (first and last of the GOP) and inputs them to a *motion estimation* (ME) sub-network to estimate the optical flow between each anchor frame and the middle frame to encode. Since the motion spans a rather large temporal gap for the anchor frames, a pyramid network able to handle large motion is used [27]. After, the *motion compression* (MC) sub-network is applied, in this case an auto-encoder which codes the estimated optical flow motion data. The auto-encoder takes as input the bi-directional motion information (obtained by concatenating the motion data produced by both motion estimation sub-networks) and transforms this information into a latent representation (q_m). This motion latent representation is quantized and transmitted to the decoder, thus allowing the HLVC decoder to work with the same motion data as the HLVC encoder. Next, the anchor frames are warped to the middle GOP position, using the decoded motion information provided by the MC module. These warped frames are used as input to the *motion post-processing* (MP) sub-network, which merges both warped frames, to achieve the final motion compensated prediction frame for the middle GOP frame. Next, the residual between the original frame and the motion compensated frame is computed and coded with a *residual compression* (RC) auto-encoder sub-network, similar to the MC network. Finally, the residual is added to the motion compensated frame to obtain the intermediate decoded frame. A quality parameter (represented as Q_5 in Figure 25 and computed for PSNR-RGB or MS-SSIM) is sent to the decoder to express the intermediate frame quality, along with the coded motion information and the coded residual.

In Figure 25, the black arrows correspond to the operations performed both at the encoder and decoder, while the blue arrows correspond to the operations taking place solely at the encoder.

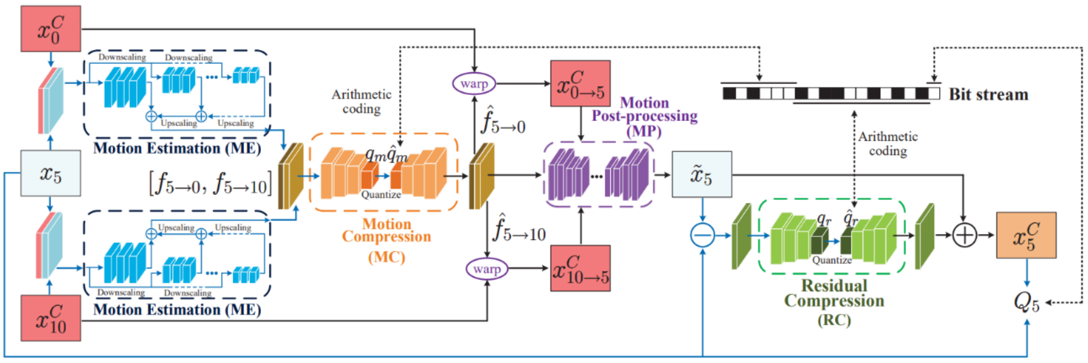

Figure 25: Bi-Directional Deep Compression network, exemplified for the first GOP [42].

<u>**Single Motion Deep Compression (SMDC)**</u>

The SMDC network architecture is presented in Figure 26; it shows, as an example, the coding process for frame numbers 1 and 2 in each GOP; this process is repeated for frames 3 and 4, 6 and 7, and 8 and 9 in each GOP. Like the BDDC network, this network sends to the decoder the coded motion information, the coded residual and the quality parameter value (not pictured in Figure 26). The SMDC network also uses sub-networks similar to those used in BDDC. This network codes two frames at a time, using as anchor only the nearest intermediate decoded frame in the GOP. The first operations are rather similar to the BDDC network (i.e., with *ME/MC*); thus, it starts by estimating the motion between the anchor frame (x_0^C) and the farthest of the two frames to code (x_2), obtaining a motion map. Then, with the motion map, the anchor frame (x_0^C) is warped to the target frame position (x_2), and the warped frame is fed into the motion post-processing sub-network *MP*. After, the residual is computed as the difference between the original and the motion compensated post-processed frames and coded using the *RC* sub-network. By adding this residue to the post-processed frame, the intermediate decoded frame (x_2^C) is obtained. To code the other frame (x_1), the motion map already obtained for frame x_2 is reused, thus allowing to save rate. In this case, using the motion from frame x_0^C to x_2, the motion field from frame x_1 to x_0^C and the motion field from frame x_1 to x_2 are obtained. Then, frames x_0^C and x_2 are warped to obtain two estimations for the target frame x_1. These warped frames go into the motion post-processing sub-network (*MP*) to obtain the final motion compensated frame, which is subtracted from the original frame to obtain the residual which is coded by the *RC* sub-network. Finally, the residual is used to obtain the intermediate decoded frame (x_1^C).

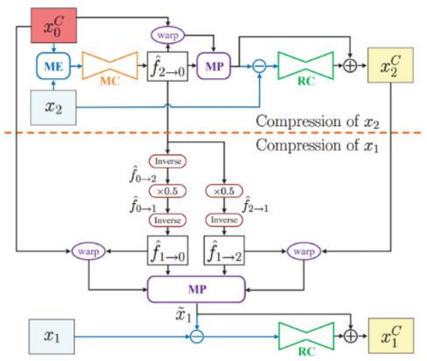

Figure 26: Single Motion Deep Compression network, exemplified for frames 1 and 2 in the first GOP [42].

<u>Weighted Recurrent Quality Enhancement (WRQE)</u>

The WRQE network is only used at the decoder side to enhance the quality of all intermediate decoded frames. It receives as input the intermediate decoded frames produced at the decoder as well as some bitstream data and produces as output the various layers enhanced decoded frames. This network was designed based on the QG-ConvLSTM method [45] shown in Figure 27. The blue layers are made of residual blocks and include skip connections to boost the enhanced frame quality. The weights generator uses some bitstream information (the quality parameter and rate) as the input to generate the weights to control the *Quality-gated cell*. This quality-gated cell controls which information from the intermediate decoded frames is more important, so it can be stored in memory. When the WRQE input is an intermediate decoded frame from the first layer, its high-quality information should be included in the memory (by increasing w_i^S) and the information retained before should be released (by decreasing w_i^M). For the lower quality frames, the weights are expected to change in the opposite way; thus, as expected, the contribution of each frame to the memory depends on its quality. Since these weights are generated as the output of a sigmoid function, they will never be larger than 1; therefore, the impact of the information in memory from previous frames will slowly decrease, expressing the fact that frames further apart have less information in common. In summary, the key idea is that the quality of the lower quality frames is improved by exploiting the higher quality of some temporally neighboring frames.

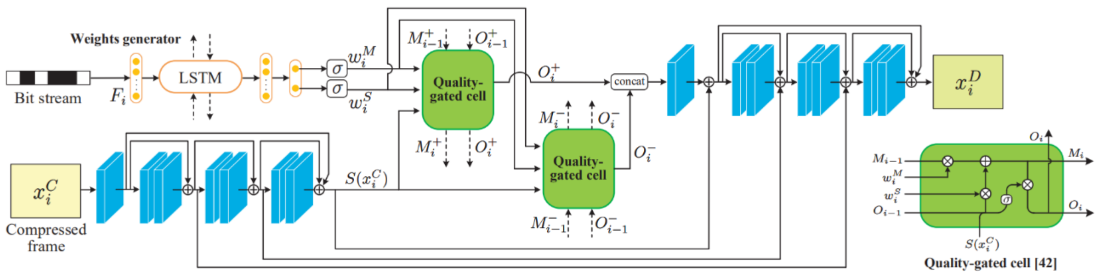

Figure 27: Weighted Recurrent Quality Enhancement network [42].

D) Performance Assessment

To perform the HLVC codec performance assessment, the following test conditions were used:

- **Dataset**: The BDDC and SMDC networks are trained separately on the same dataset, Vimeo-90k [30], which includes 89800 clips covering a large variety of scenes and actions, with a spatial resolution of 448x256 pixels. The WRQE network is trained on 142 clips from the Xiph [46] dataset, which includes videos with 720x486 resolution at 50 or 60 frames per second, and some other clips from the Video Quality Experts Group (VQEG) dataset [47]. To assess the HLVC codec, the Joint Collaborative Team on Video Coding (JCT-VC) [48] and the Ultra Video Group (UVG) [31] datasets were used. The JCT-VC dataset includes various classes of videos with different spatial resolutions; the selected videos have spatial resolutions of 1920x1080, 832x480 and 416x240 pixels. The UVG dataset includes 16 videos with spatial resolution of 3840×2160 pixels at a frame rate of 50 or 120 frames per second, and also some videos with spatial resolution of 1920x1080 pixels.
- **Objective quality metrics**: The HLVC model was trained and tested with the PSNR-RGB and MS-SSIM quality metrics.
- **Benchmarks**: The HLVC codec was compared with several learning-based video codecs, namely: 1) Habibian *et al*. deep video codec [33], reviewed in Section 3.3.3, which employs a 3D variational auto-encoder not relying on motion information and using an auto-regressive prior for entropy coding; 2) Wu *et al*. deep video codec [32], which codes the video sequences based on consecutive frame interpolations within a GOP, where the first and last GOP frames are coded using DL-based image coding techniques; 3) Lu *et al.* DVC codec [26], reviewed in Section 3.3.1, a deep video codec which replaces all the classical video codec modules with DL-based modules which are jointly trained (end-to-end). Finally, the two most recent video coding standards are also used for comparison, i.e. H.264/AVC [10] and HEVC [11], notably using the x264 and x265 "LDP very fast" mode software.

Figure 28 shows the RD performance obtained for both test datasets. To obtain enough RD points, the HLVC model was trained for four different values of λ, for each of the two selected quality metrics. The HLVC codec consistently outperforms the H.264/AVC standard and the DVC codec for every case, independently of the test set and quality metrics. When compared with the Wu *et al.* and Habibian *et al.* codecs, HLVC also performs better for the UVG dataset, for which results are available. The HLVC codec outperforms HEVC more significantly when the testing quality metric is MS-SSIM; for the PSNR-RGB, HEVC outperforms HLVC for the low bitrates.

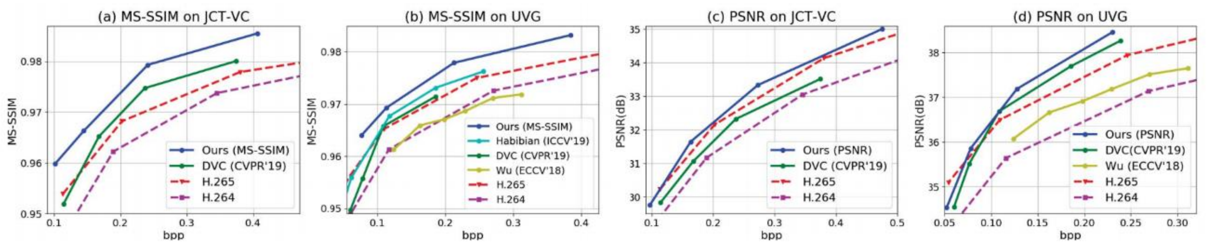

Figure 28: RD performance for PSNR-RGB and MS-SSIM using several benchmark codecs [42].

3.3.5 Learning for Video Compression with Recurrent Auto-Encoder and Recurrent Probability Model

A) Objective and Approach

In 2020, in the paper entitled “Learning for Video Compression with Recurrent Auto-Encoders and Recurrent Probability Model” [49], Yang *et al.* proposed a video codec based on Recurrent Neural Networks. This codec proposes a video coding framework based on two Recurrent Auto-Encoders (RAE) to compress motion and residual data, respectively, and two Recurrent Probability Models (RPM) to perform arithmetic coding of the motion and residue latent representations. Recurrent neural networks are used in this video coding solution to better exploit the temporal correlation along the video frames.

In the following, ‘Recurrent Learned Video Compression’ (RLVC) will be referring to the whole architecture as well as the associated model with corresponding weights [49].

B) Architecture and Walk Through

In Figure 29, the RLVC general architecture is shown. The model follows a prediction-based coding architecture with motion compensation and the calculation and further coding of a residual. The frames are structured in GOPs with Intra frames used as anchors to limit the GOP; the remaining frames in the GOP are coded based on motion-based prediction from the anchor frames and other GOP frames. The RLVC is originally trained for GOPs with 13 frames but can be used for GOPs of different sizes. Each GOP only has 2 types of frames: Intra frames (I-frames), which confine the GOP; and Predicted frames (P-frames), which exploit motion information from a single temporal direction, thus there are no bidirectionally predicted frames (B-frames). From the I-frame at the beginning of the GOP, the next 6 P-frames are predicted forward in time; moreover, from the I-frame at the end of the GOP, the previous 6 P-frames are predicted backward in time, thus achieving a 13-frame GOP size.

In each GOP, the 1st decoded (Intra) frame and the next original frame are used for motion estimation. Motion estimation is performed with a pre-trained pyramid optical flow network [27] and, then, the motion information is coded with the motion RAE model. The motion RAE works similarly to any autoencoder but includes an additional recurrent layer, which takes hidden states from past iterations and uses them to better code the current motion information and generate the associated latent representation. Next, the decoded motion information is used to create the motion compensated frame from the previous decoded frame; this prediction will be subtracted from the original frame to produce the frame residue. The residue is also coded with a (second) residue RAE model with the same architecture as the motion RAE model but now trained to code the residue. Finally, the decoded residue is added to the motion compensated frame to generate the final decoded frame.

The latent representations, for motion and residue, are transmitted to the RLVC decoder side after entropy coding with the RPM models for motion and residue, respectively. Similarly to the RAEs, the motion and residue RPMs have the same architecture, but are trained separately. Since adjacent frames are temporally correlated, i.e., the motion and residue latent representations are similar for neighboring frames, the RPM is trained to recurrently model the conditional Probability Mass Function (PMF), based on previous frames’ latent representations.

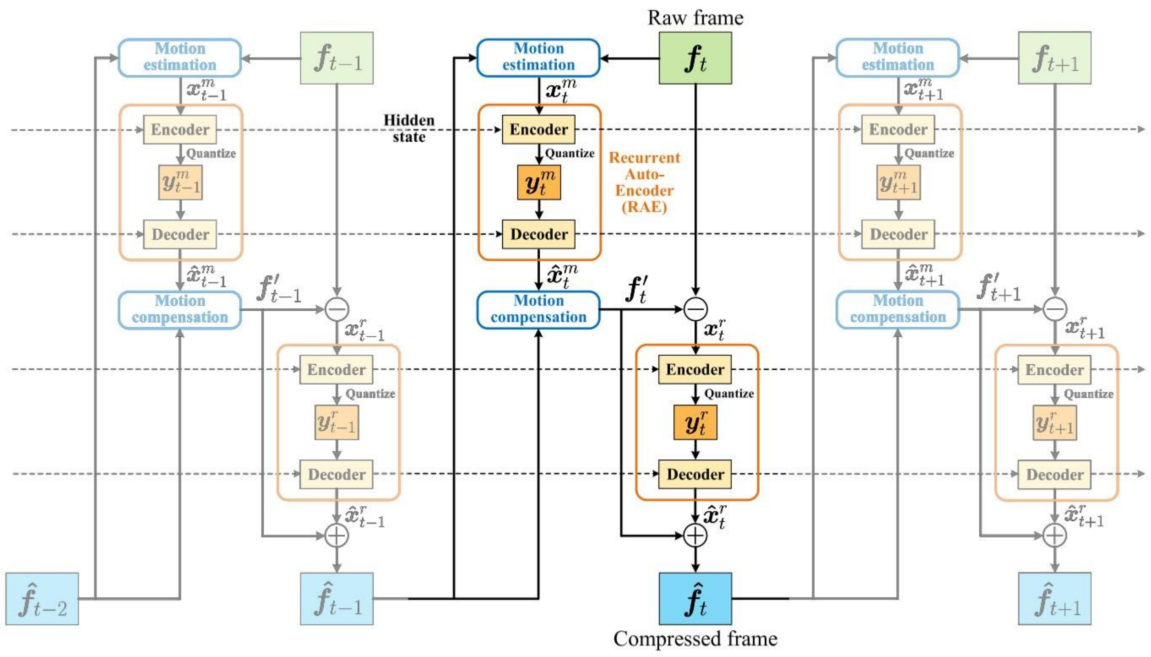

Figure 29: RLVC video coding engine architecture [49].

The RLVC coding process proceeds as follows:

- **Motion Estimation:** In the RLVC codec, the motion estimation is performed between the previous decoded frame and the next original frame to be coded; this is the first step in the video coding process. In this codec, motion estimation is performed with a so-called pyramid optical flow network [27], which can deal with large amplitude motions, as in the HLVC codec [42].
- **Motion Recurrent Auto-Encoder:** The motion RAE is responsible for coding the motion information obtained with the previous motion estimation network. The encoder part of the RAE receives the motion information and encodes this information with successive convolutional layers and a ConvLSTM layer (providing the recurrent behavior of RLVC), until a latent representation for the motion information is obtained. The consecutive convolutional layers used in the RAE result in a lossy coding process for the motion data. The decoder part of the RAE motion model takes the motion latent representation and decodes it back into useful motion information, to be utilized by the motion compensation block, using symmetrical layers to the encoder.
- **Motion Compensation:** Since lossy coding is applied to the motion information, the lossy decoded motion information must be recovered to be applied to the last decoded frame to obtain the corresponding motion compensated prediction frame. This process is accomplished with the same network used for motion compensation in the DVC codec [26], reviewed in Section 3.3.1, which firstly warps the previous decoded frame with the motion information to produce the warped frame. Then, to enhance the quality of the motion compensated frame, the warped frame, the previous decoded frame, and the motion information are processed by a CNN for artifact smoothing, and the final motion compensated frame is obtained.
- **Residue Recurrent Auto-Encoder:** The residue is obtained as the difference between the original frame and the corresponding motion compensated frame and then encoded and decoded

with the residue RAE. This RAE has the same architecture as the motion RAE but uses a different model, since they are trained for the same goal but with different types of information. The residue RAE generates the latent representation for the residue and decodes it back to obtain the (lossy) decoded residue. This information is added to the motion compensated frame, to finally obtain the current decoded frame.

- **Recurrent Probability Model:** To compress the motion and residue latent representations into a bitstream via arithmetic coding, the RPM model is used. Since video frames are temporally correlated, the motion and residue information produced when encoding each frame are also correlated; therefore, the RPM is used to estimate the conditional PMF of the latent representations of both the motion and residue. There are two RPM models, each one optimized to code motion or residue latent representations.

The RLVC training incorporates the different models/networks progressively, employing several loss functions. Initially, the motion estimation model is trained separately with a distortion-only loss function. Then, the motion estimation model is trained jointly with the motion RAE and the warping network (used to obtain the warped frame) with a RD loss function, where λ controls the RD tradeoff between the two terms. Next, the RLVC adds to the training the CNN for artifact smoothing, and finally, the last stage of training includes all the models/networks before the entropy coding process. The RPM is trained separately from the remaining network to reduce complexity and training time, with a rate-only loss function. The RLVC is also trained with two different types of distortion metrics: (1 – MS-SSIM), used to train the RLVC model optimized for MS-SSIM; and MSE, used to train the RLVC model optimized for PSNR-RGB. The training process will be described more thoroughly in Chapter 5.

C) Main Novel Tools

This section describes in more depth the key tools used in the RLVC DL-based codec, particularly those most innovative.

Recurrent Auto-Encoder (RAE)

The RAE model used in the RLVC codec is depicted in Figure 30. This RAE has as input the motion information extracted by the optical flow network, or the residue information obtained from the difference between the original frame and the motion compensated frame (x_t). These inputs are transformed into latent representations ($\tilde{y}_t$) that are quantized to their discrete value (y_t),thus creating the decoded motion or residue information ($\hat{x}_t$). The encoder and decoder sides of the RAE are symmetrical and include four convolutional layers, which down-sample or up-sample the information, using as activation function the GDN and IGDN [29], respectively. In the middle of the convolutional layers, there is a ConvLSTM layer [24], which conveys the recurrent nature of the RAE. The ConvLSTM layer receives hidden states from previous iterations of the RAE between previous frames, which carry relevant temporal information, and takes advantage of the temporal correlation between frames. Thus, the RAE creates new latent representations (y_t) considering the previous inputs as well as the information received from the current frame. In summary, the RAE exploits the temporal correlation along the video frames, targeting a more compact video representation.

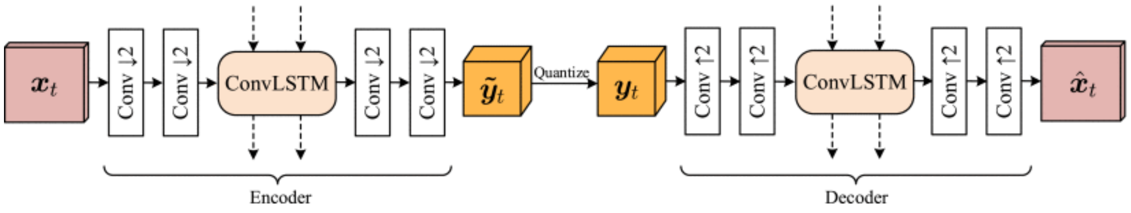

Figure 30: RLVC Recurrent Auto-Encoder architecture used for motion and residue encoding and decoding [49].

Recurrent Probability Model (RPM)

Figure 31 shows the RPM architecture. The RPM is used to estimate the conditional PMFs of the motion and residue latent representations (which take the form of a sigmoid distribution), which are used to improve the arithmetic coding process for these representations.

The arithmetic coding process usually performed uses the calculated PMFs to achieve a bitrate close to the value of the cross entropy, which is a function of the true independent PMF and the estimated independent PMF of the latent representations (the independent PMFs depend only on the current latent representations). If the calculated cross entropy is smaller, then the arithmetic coding will also achieve a smaller bitrate. If the estimated PMF has higher accuracy, i.e., the estimated PMF is more like the true PMF, then the cross entropy will be smaller, and therefore the bitrate achieved with arithmetic coding may also be smaller.

Due to the temporal correlation between video frames, the latent representations (y_t) produced at each time step tend to be correlated, thus by estimating the temporally conditional PMF, based on past (and current) latent representations, a more accurate entropy model is obtained. Therefore, the temporally conditional cross entropy may be lower than the independent cross entropy (which is just based on the current latent representation), and thus arithmetic coding can achieve a smaller bitrate.

In summary, in each time step the RPM recurrently estimates the parameters μ_t and s_t (midpoint and steepness of the PMF), using the ConvLSTM layer to account for information from previous frames and, therefore, compute the conditional PMF. Then, the conditional PMF is used by the arithmetic coding engine to obtain the bitstream for all latent representations.

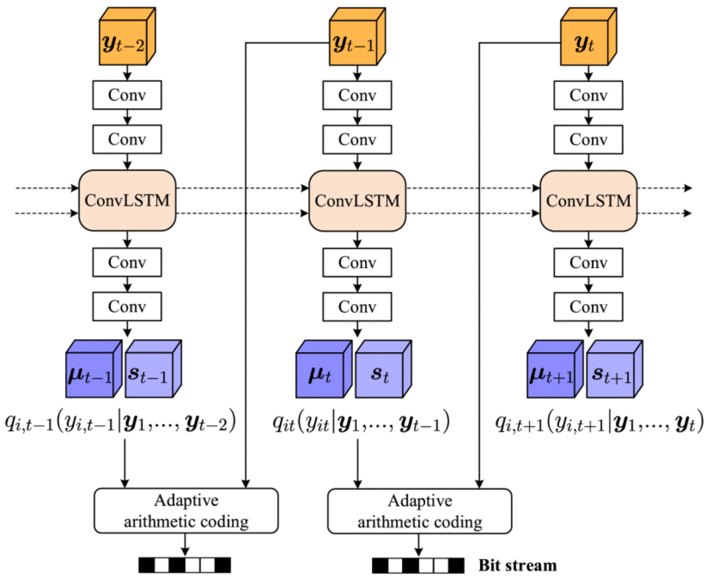
y_{t-2}
y_{t-1}
y_t
Conv
Conv
ConvLSTM
Conv
Conv
μ_{t-1}
s_{t-1}
μ_t
s_t
μ_{t+1}
s_{t+1}
$q_{i,t-1}(y_{i,t-1}|y_1,\dots,y_{t-2})$
$q_{it}(y_{it}|y_1,\dots,y_{t-1})$
$q_{i,t+1}(y_{i,t+1}|y_1,\dots,y_t)$
Adaptive arithmetic coding
Bit stream

Figure 31: RLVC Recurrent Probability Model architecture used to code the latent representations [49].

D) Performance Assessment

To assess the RLVC compression performance, the following conditions were used [49]:

- **Dataset:** For the training stage, the Vimeo-90k septuplet dataset [30] was used, which consists of 91701 video sequences of 7 frames, with spatial resolution of 448x256; these sequences are randomly cropped and used for training with spatial resolution of 256x256. There is a direct relation between the length of these sequences (7 frames) and the used GOP size, 13 frames, where 6 frames use forward motion and 6 frames use backward motion. To assess the RLVC codec RD performance, the classes B, C and D from the JCT-VC dataset [48] and the UVG dataset [31] were used. The JCT-VC class B and the UVG dataset have videos with spatial resolution of 1920x1080, and the JCT-VC classes C and D have videos with spatial resolutions of 832x480 and 416x240, respectively.
- **Objective quality metric:** For performance assessment, two common quality metrics were used, notably PSNR-RGB and MS-SSIM.
- **Benchmarks:** This RLVC codec was benchmarked against several DL-based video codecs, specifically: 1) Wu *et al.* [32], labelled as "Wu *et al.* (ECCV'18)" ; 2) Lu *et al.* DVC codec [26], reviewed in Section 3.3.1, labelled as "DVC (CVPR'19)"; 3) Habibian *et al.* [33], reviewed in Section 3.3.3, labelled as "Habibian *et al.* (ICCV'19)"; 4) Yang *et al.* HLVC codec [42], reviewed in Section 3.3.4, labelled as "HLVC (CVPR'20)"; 5) Liu *et al*. [50], which uses an accurate motion representation to exploit temporal correlation, and joint spatial-temporal priors based on recurrent neural networks, namely ConvLSTM, labelled as "Liu *et al.* (AAAI'20)"; and 6) Agustsson *et al.* [51], which proposes a more generalized method of warping that manages better common failure cases, denominated the scale-space flow, labelled as "Agustsson *et al.* (CVPR'20)". Besides the DL-based video codecs, the H.265/HEVC standard [11] was also used for benchmarking in

various configurations, namely: 1) x265 (LDP very fast); 2) x265 (LDP default); 3) x265 (default); 4) x265 (SSIM default); 5) x265 (slowest); and 6) x265 (SSIM slowest).

Figure 32 shows the RD performance comparison between the RLVC and other DL-based video codecs. Similarly, to other reviewed video codecs, to obtain enough RD points for a well-defined RD performance curve, RLVC coding was performed for four different λ values (and thus four models), one for each RD point for the various quality metrics. Comparing with the codecs trained with the PSNR-RGB distortion metric (which appear in sub-figures (c) and (d)), the RLVC shows better RD performance for both test datasets. For the MS-SSIM metric, the Agustsson *et al.* [51] codec seems to perform better than RLVC. The remaining DL-based codecs always show a worse performance than RLVC for both datasets and quality metrics.

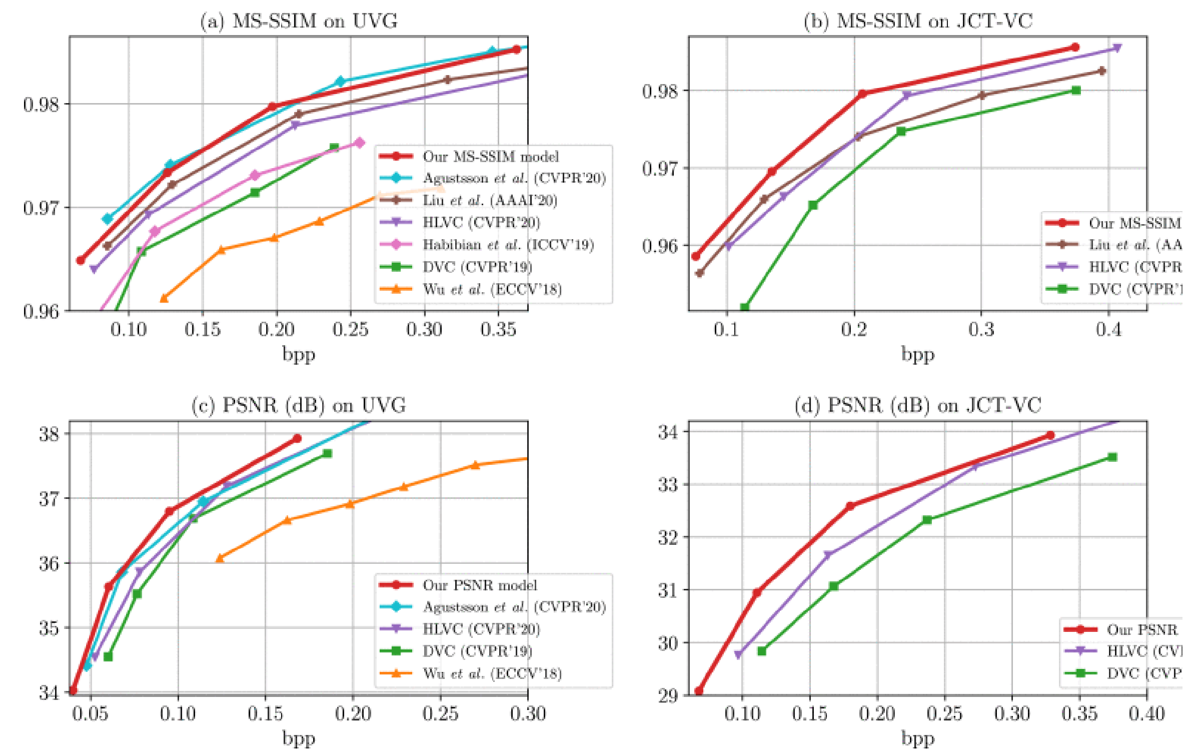

Figure 32: RLVC RD performance benchmarked against other DL-based video coding solutions: (left) UVG dataset; (right) JCT-VC dataset [49].

Figure 33 shows the RD performance comparison between RLVC and the H.265/HEVC standard [11] for various configurations. For MS-SSIM, RLVC is competitive with the x265 slowest methods (“SSIM slowest” and “slowest”), especially as the video quality increases. For the PSNR-RGB metric, the “slowest” and “default” methods have better overall RD performance, but RLVC performs on par with the “LDP default” and “LDP very fast” configurations.

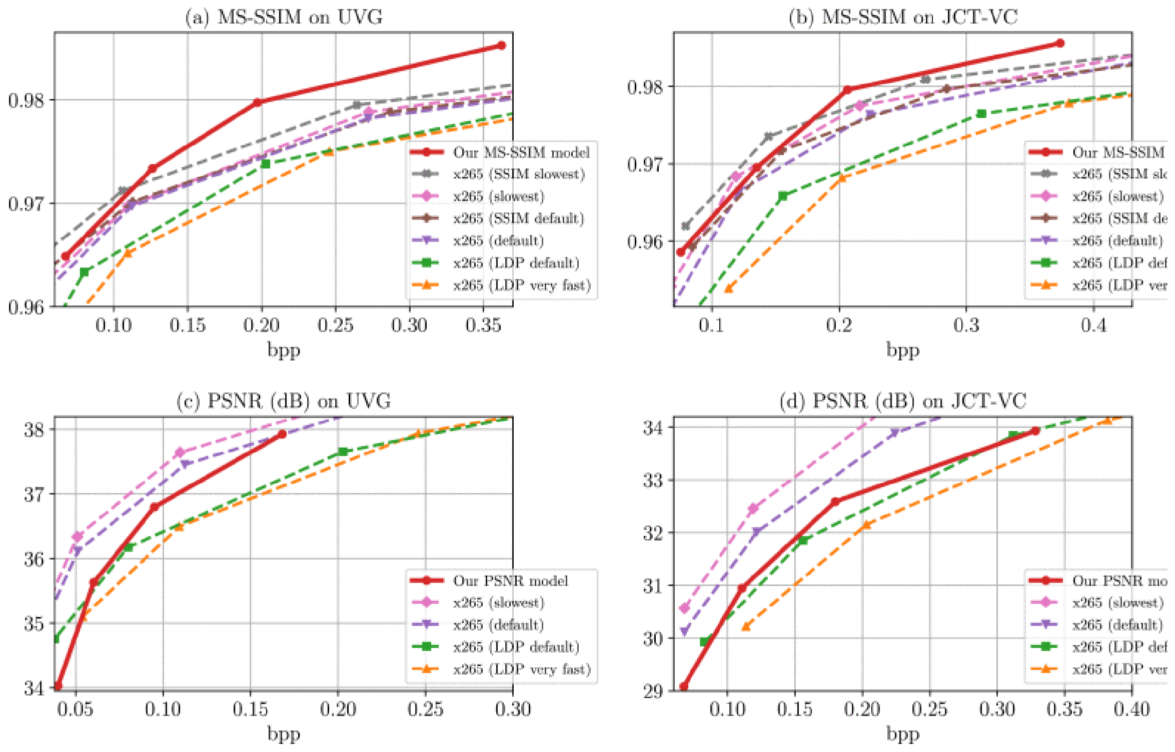

Figure 33: RLVC RD performance benchmarked against the H.265/HEVC standard in several configurations: (left) UVG dataset; (right) JCT-VC dataset [49].

4. Deep Learning-based Video Coding Benchmarking

In the context of this Thesis, it is very critical to start by knowing how the most recent DL-based video coding solutions perform in comparison to the most recent conventional video standard (the VVC standard [52]), which is the culmination of decades of development. Thus, the objective of this chapter is to perform a solid, meaningful, and extensive benchmarking of the compression performance of some recent DL-based video coding solutions regarding the most powerful and recent conventional video coding solution, the VVC standard. This comparison will be performed under the largely adopted JVET defined Common Test Conditions (CTC) and evaluation procedures for NN-based video coding technology [53]. This type of novel performance comparison is critical to know in a reliable way how far some recent DL-based video coding solutions are from the well-established conventional video coding solutions in terms of compression performance. This solid benchmarking is not yet available in the literature and implies gathering performance results under solid, meaningful, and comparable conditions, which has required software extensions, e.g., for 10-bit internal coding, and careful definition of the coding pipelines in order to perform a fair comparison.

It is worthwhile to note that the benchmarking described in this chapter has led to a conference paper entitled “Conventional versus Learning-based Video Coding Benchmarking: Where Are We?” already accepted and selected as a best paper award candidate at the International Workshop on Advanced Image Technology (IWAIT) to be held in Hong Kong, January 2022.

4.1 Video Codecs Under Study

This Section identifies the DL-based video coding solutions selected for this benchmarking study as well as the benchmarking reference, i.e. the VVC standard.

A) DL-Based Video Coding Solutions

The DL-based video codecs used for this benchmarking were selected based on the public availability of software, as using only the performance results available in published papers would not be enough to perform a solid comparison under common test conditions. Thus, the DL-based video codecs that are benchmarked in this chapter are:

- **DVC solution** [26], reviewed in Section 3.3.1, for which the used implementation was the OpenDVC [4] [5], provided by Yang *et al.*;
- **HLVC solution** [42], reviewed in Section 3.3.4, for which the used implementation was provided by the authors [56];
- **RLVC solution** [49], for which the used implementation was also provided by the authors [57].

B) Reference Codecs

The coding benchmark for this performance assessment is, naturally, the Versatile Video Coding (VVC) standard [52], the most recent and efficient representative of a succession of video coding standards based on the so-called hybrid coding architecture. This is naturally a very challenging benchmarking for the DL-based coding solutions, which are just emerging and did not have yet time to mature; however, this is also the most relevant benchmark to assess the performance gap between the technologies under comparison. In this context, three VVC coding configurations will be used:

- **VVC Random Access** (RA);
- **VVC Low Delay P** (only P frames);
- **VVC Intra**.

The VVC reference software version 11 [53] has been used for VVC coding using the JVET provided configurations [58].

C) Coding Pipelines

According to JVET NN CTC [58], VVC coding must be performed at 10-bit depth even if the content is available at 8-bit since this improves the VVC compression performance. This implies that the 8-bit videos had to be converted to 10-bit videos, in this case by adding two bits as ‘10’, i.e., in the central bin position.

To perform a fair benchmarking, the same had to happen with the DL-based codecs which were not prepared to code 10-bit videos; in this context, the software for the three selected DL-based codecs had to be upgraded to code videos at 10-bit instead of the previous 8-bit depth used in the original implementation; in this process, it was confirmed that the compression performance was improved for the selected quality metrics. Therefore, since the DL-based codecs need RGB input, the YUV, 4:2:0 8-bit depth video sequences were first converted to 10-bit depth as described above, and after, the RGB frames (16-bit PNG format) were obtained using ffmpeg following the BT.709 recommendation. After DL-based coding, the decoded 10-bit RGB frames were converted to 10-bit YUV frames as recommended by the JVET NN CTC for quality assessment.

4.2 Test Conditions and Evaluation Procedures

This section will describe the test conditions and evaluation procedures adopted for the performance benchmarking exercise in this section, which are largely those specified by JVET in the document “JVET common test conditions and evaluation procedures for neural network-based video coding technology”, called JVET NN CTC in the following [58].

A) Test Material and Coding Rates

The selected video test material corresponds to the key classes recommended in the JVET NN CTC [58], as shown in Table 1. All original videos are in the YUV color space with 4:2:0 subsampling; there are sequences at 8 and 10-bit per sample, which has required some software adaptations for the learning-based codecs. For the DL-based coding solutions, the various rates are obtained with four models trained for several bitrates using MSE-RGB in the loss function (since the JVET NN CTC uses PSNR-Y as quality metric), while for VVC the quantization steps specified in the JVET NN CTC [58] are used.

Table 1: Test video sequences and characteristics [58].

Class	Sequence Name	Number of Frames	Frame Rate	Bit-depth	Spatial Resolution
B	*MarketPlace*	600	60	10	1920×1080
	RitualDance	600	60	10	1920×1080
	Cactus	500	50	8	1920×1080
	BasketballDrive	500	50	8	1920×1080
	BQTerrace	600	60	8	1920×1080
C	*RaceHorses*	300	30	8	832×480
	BQMall	600	60	8	832×480
	PartyScene	500	50	8	832×480
	BasketballDrill	500	50	8	832×480
D	*RaceHorses*	300	30	8	416×240
	BQSquare	600	60	8	416×240
	BlowingBubbles	500	50	8	416×240
	BasketballPass	500	50	8	416×240

B) Performance Metrics

For test quality metrics, it was decided to adopt PSNR-Y as recommended by the JVET NN CTC [58] and VMAF [13] due to its recent popularity for video quality assessment. The distortion metrics used for training are those referred in Section 3.3 for each DL-based codec, which were selected by the respective authors since no retraining has been performed. As usual, the rate is measured in bits per second. For both metrics, BD-Rate values will be presented to express the rate savings/losses regarding the selected reference codec, in this case the VVC Low Delay P configuration; this choice was motivated by the need to have a good overlapping of rates and qualities among the assessed codecs to obtain reliable BD-Rate values.

As requested by the JVET NN CTC, all quality metrics computation happened at 10-bit and thus all video sequences originally at 8-bit depth were converted to 10-bit depth for quality metrics computation; remind that all coding is performed at 10-bit depth. Naturally, video sequences originally at 10-bit did not need the 8 to 10-bit conversion.

4.3 Performance Results and Analysis

This section presents the RD performance results and BD-Rates for the test conditions defined in the previous section.

A) RD Performance by Codec and Sequence

For a better visualization of the codecs' RD performance, this section includes the RD performance charts for the Class D test sequences, both for the PSNR-Y (Figure 34) and VMAF (Figure 35) quality metrics. These charts allow to make some first qualitative observations, notably: i) VVC Inter codecs (excluding VVC Intra) are clearly still the best in RD performance; ii) the VVC Inter to DL-based codecs quality gap seems to be shorter for the VMAF metric; and iii) the quality growth with rate is much faster for the VMAF metric. The RD performance gap for VMAF is much shorter than for PSNR-Y, which is a great sign considering the better VMAF correlation with perceptual quality.

B) Overall BD-Rate Performance

Table 2 and Table 3 include the PSNR-Y and VMAF BD-Rates respectively, using as anchor the VVC Low Delay P configuration. It is important to note that the RLVC BD-Rate results in [49] are rather different from those presented here for three main reasons: i) the quality metric in [49] is an uncommon PSNR-RGB based on the pixel-level average MSE for the R, G and B components and not the common PSNR-Y or PSNR-YUV metrics; ii) in [49], RLVC is compared with the HEVC standard with the coding performed with the x.265 software (and not with the VVC standard using the VVC reference software as in this chapter); and iii) the x.265 software is run at its 'LDP VERY FAST' configuration what may bring some performance penalty to reduce the complexity. These differences clearly demonstrate why a solid

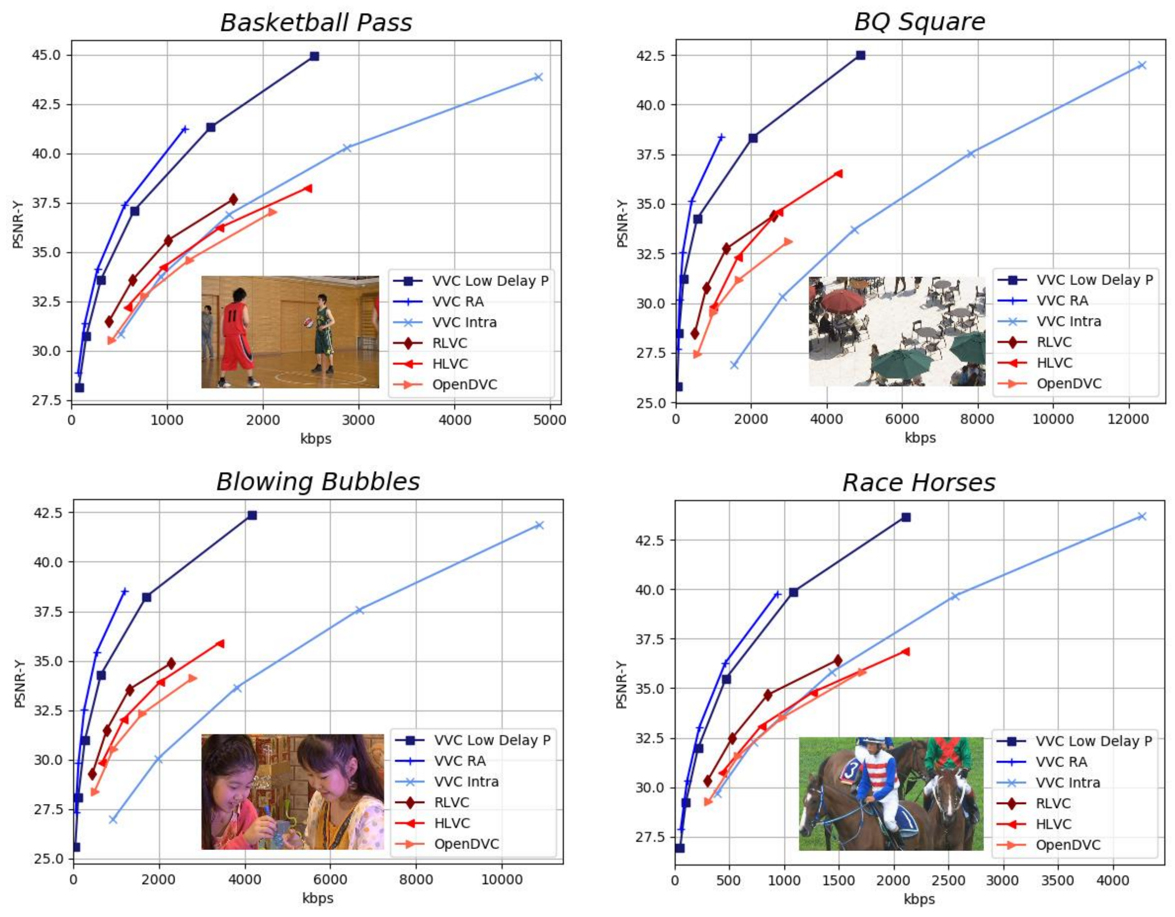

Figure 34: PSNR-Y RD performance for the Class D video sequences.

benchmarking needs clear, well-defined conditions, metrics, and pipelines as in this chapter. The RD performance results in the tables allow to derive the following conclusions:

- The best performing DL-based codec is RLVC followed by HLVC and OpenDVC. The RLVC modelling of temporal dependencies using recurrent networks (e.g., LSTM) brings performance benefits, due to its capability of making efficient predictions and tracking long-term dependencies, i.e., not only considering data of the past decoded frame.
- All DL-based codecs perform much better than VVC Intra but much worse than VVC RA. The best performing DL-based codec, RLVC, is outperformed by the VVC Low Delay P configuration performance (remind they both use Low Delay P settings). This means that DL-based tools exploiting the temporal correlation in a more efficient way are still needed.
- For the DL-based codecs, the VMAF BD-Rate is smaller than the corresponding PSNR-Y BD-Rate. Since VMAF is better correlated with human perception than the PSNR-Y mathematical fidelity measure, it means lower rate losses can be achieved if perception is considered. This is expected as DL-based codecs usually perform better for perceptual quality metrics where mathematical fidelity is not the target.

In summary, this chapter offers the first solid performance study on DL-based video coding, using the JVET NN CTC, and including the most recent conventional coding benchmark, VVC. The main conclusion is that the RD performance gap is currently still large (although lower for perceptual quality

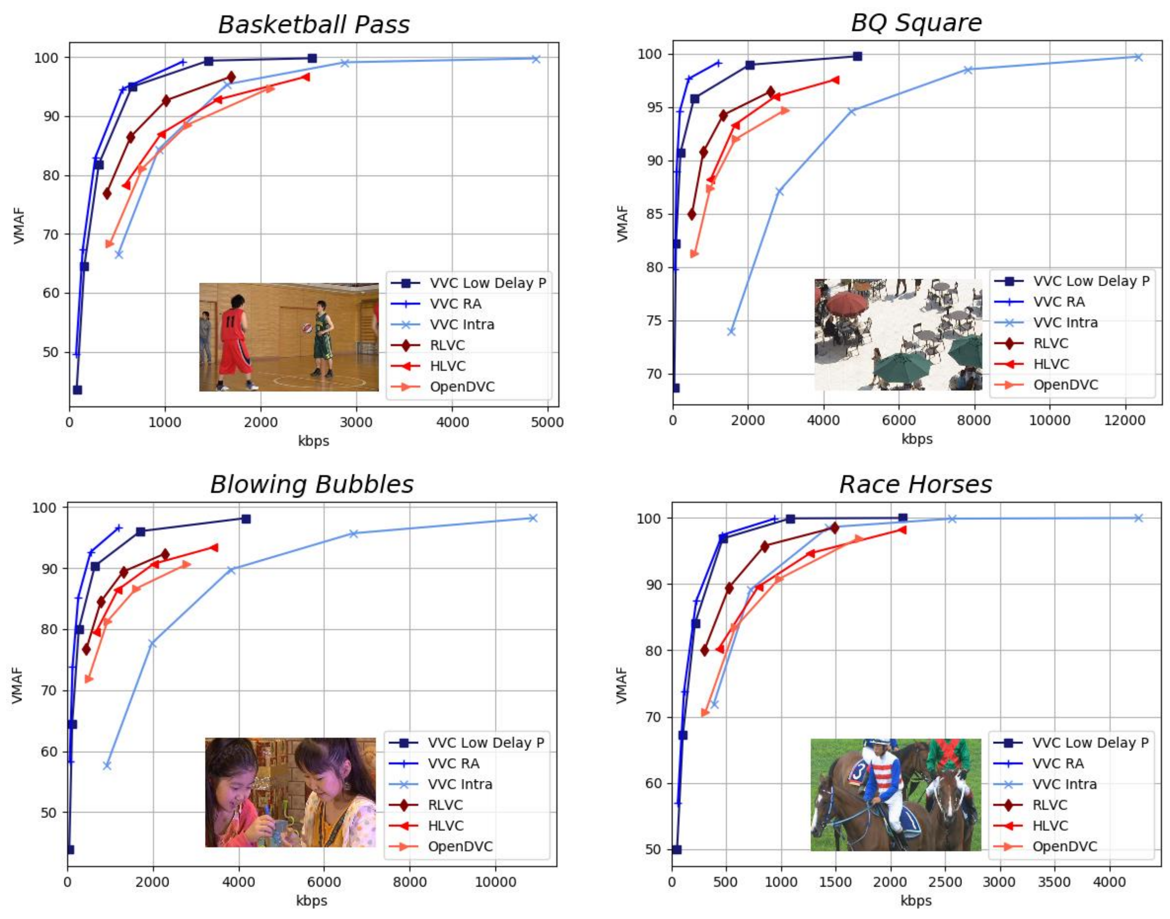

Figure 35: VMAF RD performance for the Class D video sequences.

metrics) and thus future improvements are still needed to reduce this gap as it occurred for DL-based image coding. While the DL-based codecs do not yet perform at the same level of VVC, their RD performance is very promising considering these codecs are just emerging, while VVC has had decades of research efforts behind it.

Table 2: PSNR-Y BD-Rate (%): VVC Low Delay P as reference.

		VVC Intra	**VVC RA**	**RLVC**	**HLVC**	**OpenDVC**
Class B	*MarketPlace*	370,81	-31,41	230,73	310,29	384,78
	RitualDance	245,68	-21,72	186,73	258,01	295,17
	Cactus	571,07	-22,57	333,68	467,64	546,12
	BasketballDrive	166,29	-24,83	258,84	320,54	425,50
	BQTerrace	493,17	-23,91	533,63	549,76	856,76
	Average	**369,40**	**-24,89**	**308,72**	**381,25**	**501,66**
Class C	*RaceHorses*	136,6	-18,2	224,4	297,8	348,3
	BQMall	447,2	-28,7	189,1	302,4	365,1
	PartyScene	509,2	-25,8	212,3	367,5	400,7
	BasketballDrill	413,4	-17,2	230,3	312,1	398,2
	Average	**376,6**	**-22,5**	**214,0**	**320,0**	**378,0**
Class D	*BasketballPass*	160,6	-21,1	110,5	175,4	206,1
	BQSquare	762,0	-37,6	345,4	401,9	725,2
	BlowingBubbles	515,8	-36,5	153,0	232,9	300,2
	RaceHorses	170,9	-36,7	120,9	198,2	212,6
	Average	**402,3**	**-33,0**	**182,5**	**252,1**	**361,0**

Table 3: VMAF BD-Rate (%): VVC Low Delay P as reference.

		VVC Intra	**VVC RA**	**RLVC**	**HLVC**	**OpenDVC**
Class B	*MarketPlace*		-28,42	197,33	259,56	383,42
	RitualDance		-19,63	163,85	259,56	285,45
	Cactus		-14,79	460,26	639,97	659,85
	BasketballDrive		-26,83	218,39	317,82	401,50
	BQTerrace		22,38	744,90	704,39	1 111,81
	Average		**-13,46**	**356,95**	**436,26**	**568,41**
Class C	*RaceHorses*	1 767,9	-21.12	170,1	301,4	318,5
	BQMall	1 114,4	-18,7	135,3	246,3	287,1
	PartyScene	794,3	-18,0	176,2	299,0	331,7
	BasketballDrill	458,8	-6,9	186,4	287,9	318,3
	Average	**1 033,9**	**-14,5**	**167,0**	**283,6**	**313,9**
Class D	*BasketballPass*	213,8	-19,5	78,2	156,9	184,6
	BQSquare	1 603,5	-35,3	340,1	452,3	674,1

	BlowingBubbles	699,2	-32,9	131,3	184,0	263,0
	RaceHorses	519,7	-14,0	93,0	185,8	210,9
	Average	**759,0**	**-25,4**	**160,7**	**244,8**	**333,2**

5 RLVC Training and Performance Validation

Considering the achieved results in Chapter 4, the DL-based video codec chosen to be improved was the RLVC, due to being the codec that achieved the best RD performance during the extensive benchmarking. Therefore, this chapter will present the methodology used to train the RLVC models [49] (reviewed in Section 3.3.5) with the key target to replicate, with the new trained RLVC models, the performance results obtained with the trained models made available by the RLVC authors. The newly trained models and the associated performance replication and validation are essential to be sure that the author of this Thesis is able to fully control the training and testing of the RLVC pipeline. This is critical to have a starting point that may serve as a reliable benchmark for future improvements and extensions made to the RLVC architecture, notably the integration of attention models.

The original RLVC models (i.e., made available by the RLVC authors) are optimized for two different distortion metrics, notably PSNR-RGB and MS-SSIM; for each metric, the RLVC model is trained for four different qualities using λ values of 256, 512, 1024 and 2048 for PSNR-RGB, and 8, 16, 32 and 64 for MS-SSIM. However, given the complexity and time constraints, the RLVC training and performance replication have only been performed for the PSNR-RGB metric; this choice is motivated by the fact that the VVC benchmark is optimized using the MSE distortion, thus a PSNR quality metric.

5.1 Training Conditions and Dataset

The RLVC architecture, show in Figure 36, includes six main modules that use neural networks and, therefore, must be trained:

1) Motion estimation network [27];
2) Motion RAE;
3) Motion compensation network, which has two main modules: the warping network, and the artifact smoothing network;
4) Residue RAE;
5) Motion RPM; and
6) Residue RPM.

The RLVC neural networks are defined and trained with Tensorflow.

The dataset used for training is the Vimeo-90k septuplet [30], which consists of 91701 video sequences, each with seven frames, with a 448x256 spatial resolution. Each time the frames are loaded for training, they are randomly cropped down to a 256x256 spatial resolution. RLVC performs RGB coding and not YUV coding as common for conventional video coding standards like HEVC and VVC. This is a common approach in DL-based video coding solutions to have three peer components, which may use the same coding model, what would not be the case with YUV video with chrominance downsampling.

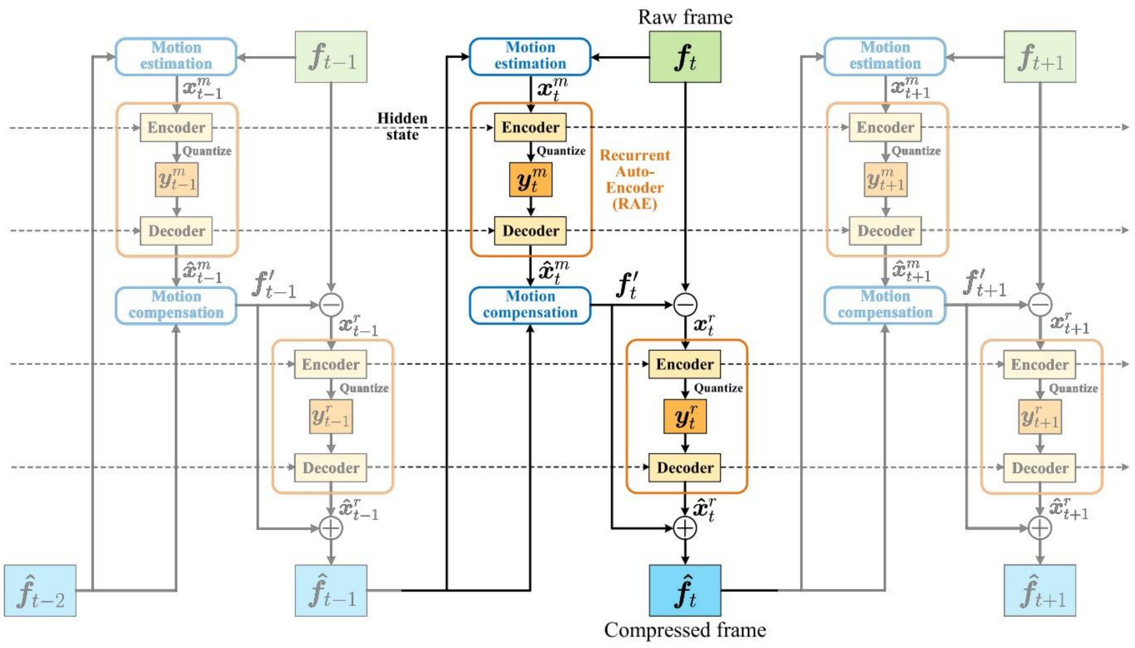

Figure 36: RLVC architecture [49].

Before performing RLVC training, some initial preparation steps need to be executed:

- **Training Dataset:** The training dataset based on the Vimeo-90k septuplet [30] needs to be stored in an easily accessible location.
- **I-frames Coding:** The first frames of all 7-frame video sequences are Intra encoded and decoded, to simulate the behavior that will occur during the testing phase where the first frame of each GOP is Intra coded. To match this Intra coding process with the Intra coding performed in the testing phase and the distortion metrics used in the training loss functions, the I-frames are encoded:
  - i) with the HEVC-based BPG codec, when the RLVC models are trained using the PSNR-RGB metric in the loss function;
  - ii) with the DL-based image coding solution presented in [44] which uses MS-SSIM in the loss function, when the RLVC models are trained using the MS-SSIM metric in the loss function.

  Since, as mentioned above, the newly trained RLVC models will be optimized for the PSNR-RGB metric, the first frame of each video sequence is coded with BPG with four different qualities, specified by the QP parameter. In this case, QP takes the values of 37, 32, 27 and 22 for the values of λ = 256, 512, 1024 and 2048, in the loss function, respectively.
- **Optical Flow Models:** Finally, the pre-trained models for the motion estimation network are obtained [27] and stored in an easily accessible location.

5.2 RLVC Training Phases

As reviewed in Section 3.3.5, the RLVC model is trained progressively with several, different loss functions and neural networks. The RLVC training strategy proposed by the RLVC authors includes three phases as follows:

- **Phase 1: Frame by Frame Training:** The first training phase employs three different loss functions and a progressive involvement of all the RLVC neural network modules with the exception of the RPMs (for entropy coding). The distinct characteristic of this first training phase is that its loss functions only use sequences of two frames (frame pairs) to train the networks, thus the naming "Frame by Frame".
- **Phase 2: Frame Sequence Training:** Since all the neural networks involved in the generation of the latent representations (both RAEs, motion estimation, and motion compensation networks) are initialized and pre-trained in Phase 1, Phase 2 performs the fine-tuning of these models using all seven frames of the training sequences. The loss function is similar to the final one used in Phase 1 while taking into account the information of the entire sequence of seven frames.
- **Phase 3: RPM Training:** Lastly, the RPM networks are trained, namely the motion and residue RPMs. The RPM training uses the already established models for the remaining networks (obtained at the end of Phase 2) and trains only the RPMs, maintaining all the other models fixed. The two RPMs are trained separately in this Phase 3 to allow for faster and less complex training for the rest of the neural network modules in Phases 1 and 2. Since the remaining network weights are not updated in this phase, the RPM training is rather quick.

5.2.1 Phase 1: Frame by Frame Training

Frame by frame training is the first phase of RLVC training. This phase uses three different loss functions and progressively includes networks into the training process until all neural networks, except the RPMs, are included. To play the role of the RPMs, an entropy bottleneck [28] (implemented by Tensorflow Compression) is used to model the arithmetic coding and quantization processes and estimate the rate when needed for the loss functions.

A) Loss Functions

The first loss function used in this phase only takes into account the motion estimation network, the motion RAE and the warping network, by computing the MSE-RGB between the original frame (f_t) and the warped frame (marked with (1) in Figure 37), which will be referred as "MSE-RGB (warped frame, original frame)". This loss function is an RD loss function, with a trade-off between the MSE-RGB and the bits per pixel (bpp) estimated with the entropy bottleneck used with the motion RAE, which will be referred as "bpp_{mv}". Since this loss function considers the distortion after the frame warping, but before its refinement, it will be referred as "WF Loss Function" (WF from Warped Frame) and the corresponding loss function is defined as:

$$WF\ Loss\ Function = \lambda \times MSE\_RGB(warped\ frame, original\ frame) + bpp_{mv}. \qquad (3)$$

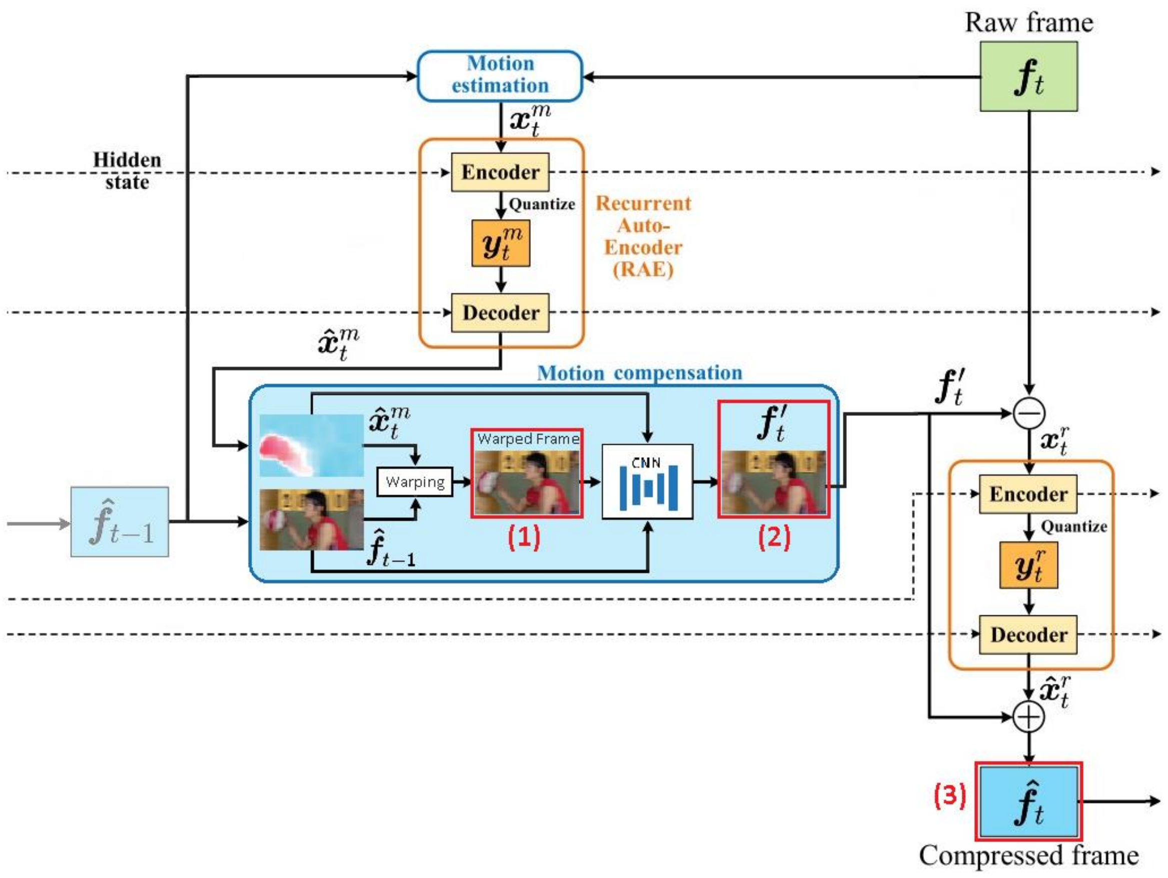

Figure 37: RLVC architecture for time step *t* with explicit Motion Compensation module.

The second loss function used in Phase 1 includes all the previously mentioned neural networks and adds the CNN used for artifact smoothing of the warped frame; therefore, this loss function computes the MSE-RGB between the original frame (f_t) and the motion compensated frame (marked with (2) in Figure 37), which will be referred as "MSE-RGB (motion compensated frame, original frame)". This loss function is also an RD loss function and continues to use as rate estimation the bpp_{mv} since the residue RAE model is not being trained yet. This loss function compares the frame obtained at the end of the Motion Compensation module with the original frame; therefore, this loss function will be referred as "MC Loss Function", and can be written as:

$$MC\ Loss\ Function = \lambda \times MSE\_RGB(motion\ compensated\ frame, original\ frame) + bpp_{mv}. \quad (4)$$

The third and last loss function used by this training stage includes all the neural networks, except the RPMs. Thus, it continues to include all the aforementioned networks used by the second loss function, and finally also includes the residue RAE. Thus, this loss function measures the distortion as the MSE-RGB between the original frame (f_t) and the final decoded frame (marked with (3) in Figure 37). For the rate part of the equation, both the rate of motion (bpp_{mv}) and residue (bpp_{res}) are considered, acknowledging that the residue RAE is also being trained with this loss function. This third and final Phase 1 loss function will be referred as "Total Loss Function" and is defined as:

$$Total\ Loss\ Function = \lambda \times MSE\_RGB(decompressed\ frame, original\ frame) + bpp_{mv} + bpp_{res}. \tag{5}$$

As can be observed from the training loss functions (3), (4) and (5), this training phase only takes into account the distortion and rate of one frame at a time (using another frame to create the predictions). This training process always takes frames in pairs, but not necessarily the first pair of the sequence as will be described later, and trains the networks with the distortion measured for the second (decoded) frame of the pair and the rate associated to obtain this decoded frame from the first (decoded) frame of the pair. Even though the loss function only considers one frame at each time for this training, the hidden states of the ConvLSTM layer are passed through, from one iteration to the next, when consecutive frames are used.

Although the RLVC coding process includes both forward P-frames and backward P-frames (depending on the motion compensation temporal direction), RLVC training is only performed in the forward time direction.

B) Hyperparameters and Batch Configuration

The main hyperparameters used in this training stage are defined in Table 4. This training stage is the one for which the RLVC authors propose most iterations. The loss function, the learning rate, and the batch configuration change accordingly to the iteration.

Table 4: Hyperparameters for RLVC Phase 1 training.

Hyperparameter	Value	
Iterations	700 000	
Batch Size	4	
Learning Rate	Initial	1x10^{-4}
	After 300 000 iterations	1x10^{-5}
	After 600 000 iterations	1x10^{-6}
Activation Function	Relu (default)	
	Tanh	
Optimizer	Adam	

Although the loss functions in this phase only use the distortion and rate of one frame (and not cumulative distortion or rate), the ConvLSTM layer still needs to be trained with a sequence of frames to offer improvements over the non-recurrent models. For the training, the frames are loaded in batches where a batch is a division of the training dataset into smaller portions so the network can be trained without needing too much memory, especially with larger datasets. In this case, the batch size is four, which means that at each iteration four different sequences will make a forward and backward pass through the networks and, at the end, the network parameters will be updated. This phase of training considers two different batch configurations, which change according to the number of iterations passed, as follows:

- **First Pair:** Initially, the networks are trained with the three loss functions described above but only considering for training the first pair of frames in each training sequence: Frame 1 (the pre-coded, GOP start, Intra frame); and Frame 2. With each new training iteration, a new batch of frames is loaded (the batch size is four, therefore in each batch there are four pairs of frames) and all ConvLSTM hidden states are forgotten. Consequently, this training configuration only uses the first pair of frames from the available sequences and does not fully exploit the sequence temporal dependency. Thus, this part of training does not make use of the recurrent nature of the ConvLSTM layer since the hidden states are always restarted after each iteration.
- **Pairs Within the Sequence:** The second batch configuration includes the whole 7-frame sequence, even though the loss function only processes the distortion and rate for one pair of frames at a time. In this batch configuration, each batch loads four full sequences (with seven frames) and passes them through the whole RLVC architecture (still without including the RPMs). The full sequence is loaded, and each pair processed (Frame 1 and Frame 2, then Frame 2 and Frame 3, etc.) corresponds to an iteration; a key difference in this batch configuration is that the ConvLSTM hidden states are not erased. So, after processing Frame 1 and Frame 2, the training goes on to processing Frame 2 and Frame 3 in the next iteration, making use of the ConvLSTM hidden states created in the previous iteration. In each iteration, only the distortion and rate corresponding to the current encoded frame are used by the loss function. Therefore, Phase 1 training is rightly designated as "Frame by Frame" training.

Each time a new batch is loaded, four sequences are chosen randomly, and, for each sequence, it is randomly chosen where the frames are going to be spatially cropped; this cropping is applied in the same way to all frames of each sequence. Depending on the batch configuration, each batch will have four pairs of frames, or four whole sequences of frames.

C) Training Process

Initially, all the model's weights are randomly initialized and the optical flow's pre-trained models are imported. An iteration corresponds to the processing of a pair of frames, independently from how many frames are in the batch. Therefore, a batch can last one iteration (in the "First Pair" configuration) or seven iterations (in the "Pairs within the Sequence" configuration); when the last pair in the batch is processed, a new batch is loaded. As stated above, the batch size is four; thus, in every iteration, the data tensor has four different pairs (or sequences) loaded.

With the loss functions and batch configurations defined, the training process suggested by the authors can be described considering successive iteration intervals as follows:

<u>Iterations 0 to 20 000</u>

In the beginning of training, all model weights are randomly initialized, except for the pre-trained motion estimation network. The learning rate starts with the initial value of $1x10^{-4.}$ The loss function in use is the "WF Loss Function", which will continue the refinement of the pre-trained optical flow models, and start training the motion RAE and the warping network. The batch configuration used in this iteration interval is "First Pair"; therefore, in this interval, each batch is comprised of four pairs of frames, all corresponding to the frames 1 and 2 of randomly chosen sequences, whose frames are cropped when they are loaded into the batch. In this stage, at the end of each iteration, a new batch is loaded, and the ConvLSTM hidden states are all reinitialized. The loss function works only with the distortion and rate associated to coding the second frame from the first (decoded) frame.

Iterations 20 000 to 40 000

The only change in this second iteration interval is the loss function. The batch configuration continues to be “First Pair” and the learning rate is maintained. The loss function in this stage is the “MC Loss Function”, which will continue to refine all the neural networks already mentioned in the previous iteration interval and will add the training of the CNN used for frame refinement after the warping.

Iterations 40 000 to 100 000

This stage of training considers for the first time the “Total Loss Function”. Since all the remaining training features remain the same, this stage only adds the training of the residue RAE model, by computing the loss function considering the final (decoded) frame and also adding the rate for residue coding.

Iterations 100 000 to 300 000

In this stage, the batch configuration is changed, but the loss function remains the same. Here, the “Pairs within the Sequence” configuration is used. Therefore, each batch after iteration 100 000 until iteration 700 000 (the limit) considers four different sequences (each with seven frames) and, in each iteration, one pair of those sequences is processed; the ConvLSTM hidden states are saved to be used with the next pair of frames within the sequence. The main characteristic of this stage is the use of the ConvLSTM layer to exploit the recurrent nature of the RLVC architecture.

Iterations 300 000 to 600 000

Every feature and parameter is maintained from the previous iteration interval, except the learning rate, which is divided by 10, and takes the value of $1x10^{-5}$. Since the loss function and batch configuration are already including the most important learning elements, the loss function takes into consideration all the networks except the RPM, and the batch configuration allows to make use of the recurrency present in the RAEs, the only step that needs to be taken to improve the training is to adapt the learning rate. In the beginning, the learning rate should be higher so the parameters can evolve faster towards the training optimal point. As that point approaches, the learning rate should be smaller so that the optimum point can be found with high accuracy.

Iterations 600 000 to 700 000

This iteration interval is the last stage of Phase 1 training. All parameters remain the same aside from the learning rate, which takes the value of $1x10^{-6}$. Thus, this training finishes using the “Total Loss Function” and the “Pairs within the Sequence” configuration for the batches. Phase 1 of training has as stopping criterion reaching iteration 700 001.

5.2.2 Phase 2: Sequence Training

At the start of this phase of training, all the neural networks (except the RPMs) have already been initialized and partially trained. As a consequence, only one loss function is used, which includes the training of all the previously mentioned models. This phase is called “Sequence Training” because the models are being optimized for the distortion and rate corresponding to the whole sequence of seven frames, and not the distortion and rate for only one frame.

A) Loss Function

Phase 2 of training only uses one loss function. As previously mentioned, this training phase trains all the networks except the RPMs; thus, this loss function considers the cumulative sum of the distortion and rate corresponding to the whole training sequence, instead of considering only the last encoded frame. Therefore, this loss function, which will be referred as "Whole Sequence Loss Function", can be written as:

$$\begin{aligned} &Whole\ Sequence\ Loss\ Function \\ &= \lambda \times \sum_{i=2}^{7} MSE\_RGB(decompressed\ frame_i, original\ frame_i) \\ &+ \sum_{i=2}^{7} (bpp_{mv_i} + bpp_{res_i}). \end{aligned} \tag{6}$$

B) Hyperparameters and Batch Configuration

Table 5 shows the main hyperparameters used in this training phase. While this phase includes less iterations, each iteration processes six pairs of frames; therefore, each iteration will take a longer time to process. The loss function and batch configuration remain the same through the whole training process in this phase, only the learning rate is adapted depending on the iteration number.

Table 5: Hyperparameters for RLVC Phase 2 training.

Hyperparameter	Value	
Iterations	150 001	
Batch Size	4	
Learning Rate	Initial	$1x10^{-4}$
	After 100 000 iterations	$1x10^{-5}$
	After 150 000 iterations	$1x10^{-6}$
Activation Function	Relu	
Optimizer	Adam	

In this phase of training, there is only one batch configuration for the loaded frames, the "Pairs within the Sequence". In each iteration, a batch is loaded with four different 7-frame sequences chosen and spatially cropped randomly. In this training stage, one iteration corresponds to the processing of the whole batch, since the loss function includes the MSE-RGB cumulative sum and bpp corresponding to all the frames in each training sequence.

C) Training Process

Phase 2 of training initiates by loading the models trained in Phase 1, thus all models are already pre-trained at this point and will only be fine-tunned (excluding RPMs). This training phase is referred as “Sequence Training” because the loss function aims to improve the distortion and rate over the whole sequences. Unlike the previous phase, where the loss function considers only one pair of frames in each iteration, in this phase, with the “Whole Sequence Loss Function”, it is possible to learn models which distribute the quality and rate between the most critical frame positions, instead of improving the quality solely for one frame at a time.

This training phase is more regular throughout training because most hyperparameters are the same during the whole process, and the only changes regard the learning rate; in fact, it is essentially a refinement phase. The learning rate begins with the same value as in Phase 1 ($1x10^{-4}$), and is successively divided by 10 after iteration 100 000, and again by 10 after iteration 150 000, so this training phase also ends with a learning rate of $1x10^{-6}$.

Phase 2 of training is the last phase to train the RAEs and remaining network models needed for encoding, except the RPM networks.

5.2.3 Phase 3: RPM Training

In this training phase, only the two RPM networks are trained. All the remaining networks have been trained and fine-tuned in Phase 1 and Phase 2 and the model weights obtained after Phase 2 are used in Phase 3 to train the RPMs: in practice, Phases 1 and 2 define the quality and Phase 3 defines the rate for each RD point. Therefore, this phase loads the previously trained models, but does not change them further, only trains the RPM models for motion and residue data entropy coding.

A) Loss Functions

Phase 3 training considers two distinct loss functions to train the motion RPM and the residue RPM, respectively. Since the other networks have already been trained and the RPMs are only used for arithmetic coding, the only relevant parameter to consider in this training phase is the rate, since the distortion does not change when performing arithmetic encoding to create the bitstream.

The first loss function, called “Motion Loss Function”, is used to train the motion RPM; therefore, its single objective is to reduce the rate associated to the motion latent representation along the whole sequence of frames. Thus, this loss function becomes:

$$Motion\ Loss\ Function = \sum_{i=2}^{7}(bpp_{mv_i}). \qquad (7)$$

The second loss function is used to optimize the residue RPM; therefore, only the rate associated to the residue latent representation along the training sequence is considered. This loss function will be referred as “Residue Loss Function”, and can be written as:

$$Residue\ Loss\ Function = \sum_{i=2}^{7}(bpp_{res_i}). \qquad (8)$$

B) Hyperparameters and Batch Configuration

The key hyperparameters used in this training phase are included in Table 6. Since only the RPM models are trained in this training phase while the remaining networks remain the same, this phase has much less iterations than the previous phases. All parameters, configurations and loss functions remain the same, except for the learning rate, which is updated once.

Table 6: Hyperparameters for RLVC Phase 3 training.

Hyperparameter	**Value**	
Iterations	20 010	
Batch Size	32	
Learning Rate	Initial	$1x10^{-4}$
	After 10 000 iterations	$1x10^{-5}$
Activation Function	Relu	
Optimizer	Adam	

As can be observed from the loss functions (7) and (8), they use the sum of the rate corresponding to the whole training sequence; therefore, the batch configuration is naturally the "Pairs within the Sequence". In each iteration of this training phase, a new batch is loaded, which includes 32 whole sequences, randomly selected and spatially cropped from the training dataset. In each iteration, the whole sequence is processed and the associated rate for the whole sequence of seven frames is used in the loss function to train both RPMs.

C) Training Process

Since there are two different models for the motion RPM and the residue RPM, although they have the exact same architecture and training process (except for the loss function), the same type of training is performed twice, first to obtain the motion RPM model, with the "Motion Loss Function", and second to obtain the residue RPM model, with the "Residue Loss Function".

Phase 3 training starts by loading the previously trained models, notably the optical flow network, the motion RAE, the motion compensation network, and the residue RAE. First, the RPM models are randomly initialized, similarly to Phase 1 for the other trained models (excluding the optical flow model). For the training of the RPMs, the training sequences are encoded with the RLVC trained models already obtained, and the latent representations achieved with the two RAEs are arithmetically encoded with the RPMs under training.

In each iteration, a new batch is loaded and fully processed. For the first pair of processed frames, the estimation of the conditional PMF is skipped because there are no previous PMFs from which to estimate. For the remaining pairs in the training sequence, the conditional PMF and the resulting bitrate are estimated based on the information from previous frames. The sum of the rate for all frames of the sequence is used to train the RPM networks, for motion and residue, separately.

Most hyperparameters in this training phase are constant, and the only change is the learning rate, which is divided by 10 at iteration 10 000, after starting at $1x10^{-4}$. The loss functions and the remaining parameters, namely the batch configuration, are maintained throughout the training. When Phase 3 concludes, the whole RLVC model is trained and ready to be tested.

5.3 Training Analysis

This section will analyze the training of the three training phases described in Section 5.2, namely the loss functions' evolution and time consumed in training. The GPU used for the training of the models was a NVIDIA Tesla V100, with 32GB of memory.

5.3.1 Phase 1: Frame by Frame Training

As previously described, this training phase considers three different loss functions, and performs 700 000 iterations. With the used GPU, this training phase takes approximately 150 hours (6 days and 6 hours) to train each model. The evolution of the "Total Loss Function" (5) can be observed in Figure 38.

In Figure 38, two curves can be observed, which correspond to the loss function values at each iteration (in light blue) and a smoothing of these values (in dark blue). The smoothed values provide an easier interpretation of training, as they ignore 'noisy' outlier values and better show the training trend.

As previously described in Section 5.2.1, since Phase 1 of training uses three different loss functions in succession, the "Total Loss Function" does not show much improvement in the earlier iterations. In fact, from iteration 0 to 20 000, the "WF Loss Function" is used, and from iteration 20 000 to 40 000, the "MC Loss Function" is used; therefore, until iteration 40 000 the training is not being optimized for the "Total Loss Function" displayed in Figure 38. From iteration 40 000, a sharp drop can be seen, as expected; however, after iteration 100 000, there is a slight deterioration of the loss function values, that may be explained by the change in the batch configuration, which after iteration 100 000 incorporates the remaining dataset; therefore, the RLVC starts leveraging the use of the ConvLSTM layer in this part of the training.

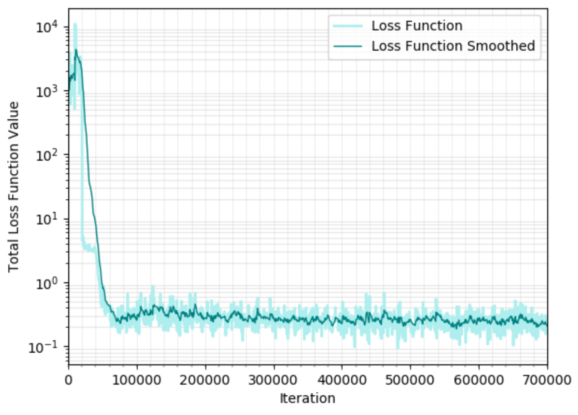

Figure 38: Evolution of the "Total Loss Function" in Phase 1 of training.

5.3.2 Phase 2: Sequence Training

With the used GPU, the second training phase takes roughly 112 hours (4 days and 16 hours) to train each model. Figure 39 shows the evolution of the "Whole Sequence Loss Function" used in Phase 2 of training.

This phase of training is rather different from Phase 1, since it performs fine-tuning; as such, the progress to be made is smaller and it is harder to optimize the loss function. In this case, the smoothed curve of the loss function provides a more straightforward insight into the training trend, and makes it easier to see the clear downward trend in the loss function, even though the evolution of this loss function is more erratic. While the values of this loss function have large variations, as observed in Figure 39, it is important to note that this loss function begins with a much smaller value than that of Phase 1, so every variation is more noticeable than in the previous training phase due to the smaller dynamic range of the variations. Moreover, since the “Whole Sequence Loss Function” takes into account the cumulative distortion and rate, any variation in the network weights that causes a negative progress will have a higher impact in the loss function of Phase 2 than it would in the loss function of Phase 1, which considers only the distortion and rate for one frame at a time.

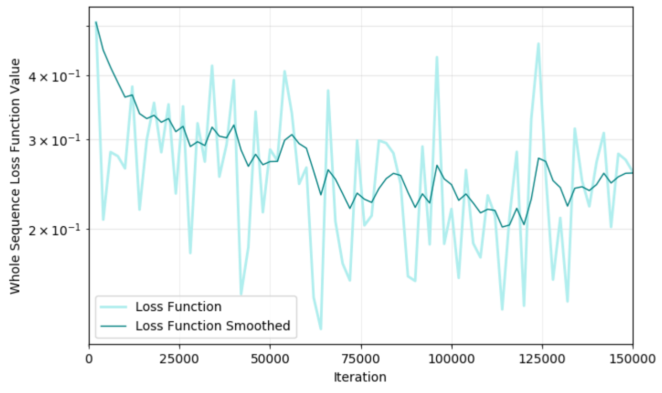

Figure 39: Evolution of the "Whole Sequence Loss Function" in Phase 2 of training.

Figure 40 shows the comparison of the PSNR-Y achieved for the Class D sequence, *Race Horses*, when using the models at the end of Phase 1 and Phase 2. It can be observed, that even though the Phase 2 training loss function seems to be hard to optimize, as can be observed in Figure 39, the models achieved at the end of Phase 2 clearly provide better PSNR-Y results.

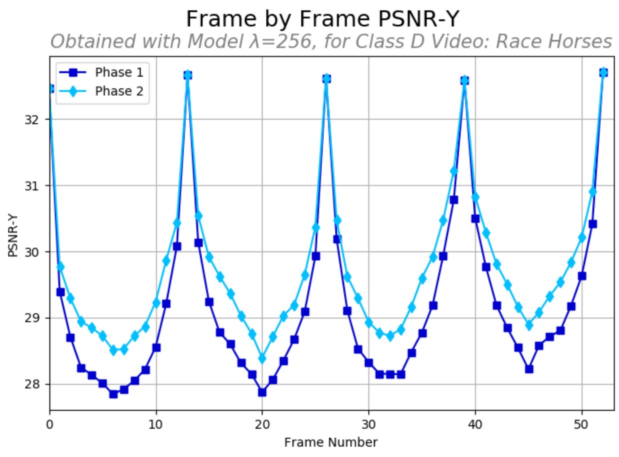

Figure 40: PSNR-Y temporal evolution per frame for the Class D sequence, *Race Horses*, corresponding to the models trained after Phase 1 (dark blue) and after Phase 2 (light blue).

5.3.3 Phase 3: RPM Training

The last training phase is rather fast, relatively to the previous training phases, since the models being trained in this phase, the RPMs, are smaller than the set of networks trained in the earlier phases. This phase of training grossly takes 24 hours, for each RPM of each model. Figure 41 shows the evolution of both loss functions used for the training of each of the RPMs (motion and residue, respectively).

This training phase progresses rather quickly and with a clear loss function trend, given that it only considers the bitrate, and the only weights to update are those for the RPMs, which means there are less weights to set, and therefore less elements to provoke instability.

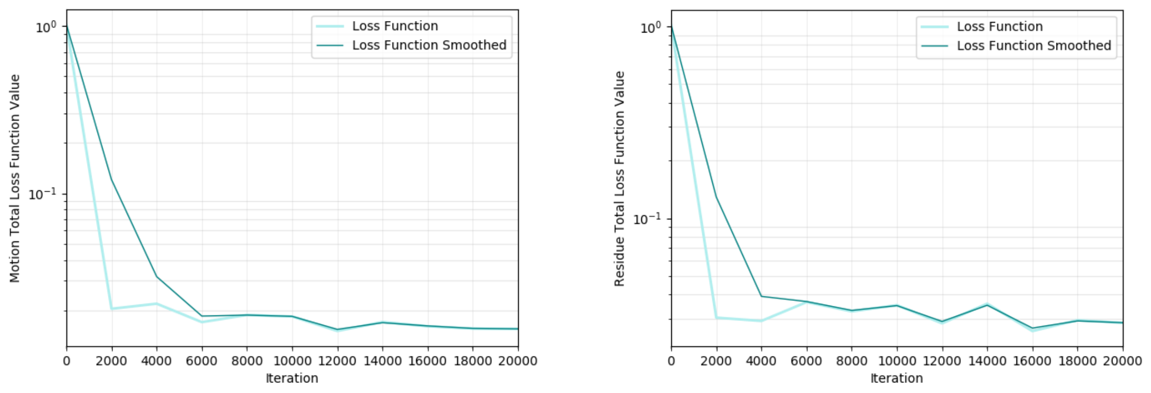

Figure 41: Evolution of loss functions in Phase 3 of training.
Motion Loss Function (left); Residue Loss Function (right).

5.4 Performance Validation

After completing all the training phases described in Section 5.2, the newly trained RLVC models were used to meet the key objective of this section: to replicate the RD performance obtained with the pre-trained RLVC models provided by the RLVC authors and, thus, to validate the newly trained RLVC models as a sound starting point for the improvement of the RLVC framework (and software). The newly trained RLVC models were used to obtain RD performance results for the RLVC codec, notably for the Class C and D JVET video sequences, in the same testing conditions as used for the RD performance results obtained with the original, pre-trained RLVC models, which have been reported in Chapter 4. Additionally, other RLVC configurations for the training phases were studied, with the purpose of determining if any time costly training phase could be skipped or shortened to less iterations with the goal to reduce the overall (large) training time.

Besides the 'Original RVLC Models', the RLVC training configurations studied were:

- **Newly Trained RLVC Models:** Trained by the author of this Thesis under the same precise conditions as suggested by the RLVC authors [49] to very closely replicate the original RD performance results.
- **RLVC Configuration 1 – Phase 1 RAEs:** The key characteristics of this configuration are:

  - Both RAEs, as well as the motion estimation, and motion compensation networks are only trained with Phase 1 of training (not Phase 2 as the “New RLVC Models”), notably with 700 000 iterations.
  - RPMs are trained from the RAEs, the motion estimation, and motion compensation network at the end of Phase 1 of training.
  - The training time of this configuration is cut short, as Phase 2 of training is totally skipped.
  - The key objective of studying this training configuration is to understand if Phase 2 brings a significant improvement to the quality of the video produced, and thus if it is essential, or if it might be skipped to shorten the training time to almost half the original time.
- **RLVC Configuration 2 – Phase 1 RAEs Shortened:** The key characteristics of this configuration are:
  - RAEs as well as the motion estimation and motion compensation networks are trained only with Phase 1 until iteration 300 000 (to save time).
  - RPMs are trained using the networks obtained at iteration 300 000 of Phase 1 of training.
  - The training time of this training configuration is shorter than for the ‘Phase 1 RAEs’ since Phase 1 is shortened to less than half the iterations.
- **RLVC Configuration 3 – Phase 1 RAEs Further Shortened:** This configuration is similar to “RLVC Configuration 2” with:
  - RAEs, motion estimation and motion compensation networks are trained until iteration 200 000 of Phase 1 (less than the 300 000 iterations used above), and the RPMs are trained from these models.

Figure 42 and Figure 43 show, for the Class C and D JVET videos, the PSNR-Y RD performance for the four target qualities using the PSNR-RGB metric as the training quality metric for the following configurations: i) Original RLVC Models (in dark blue); ii) the Newly Trained RLVC Models (in light blue); iii) RLVC Configuration 1 (in dark red); iv) RLVC Configuration 2 (in orange); and v) RLVC Configuration 3 (in yellow). Figure 44 and Figure 45 show a similar RD performance comparison for the VMAF quality metric.

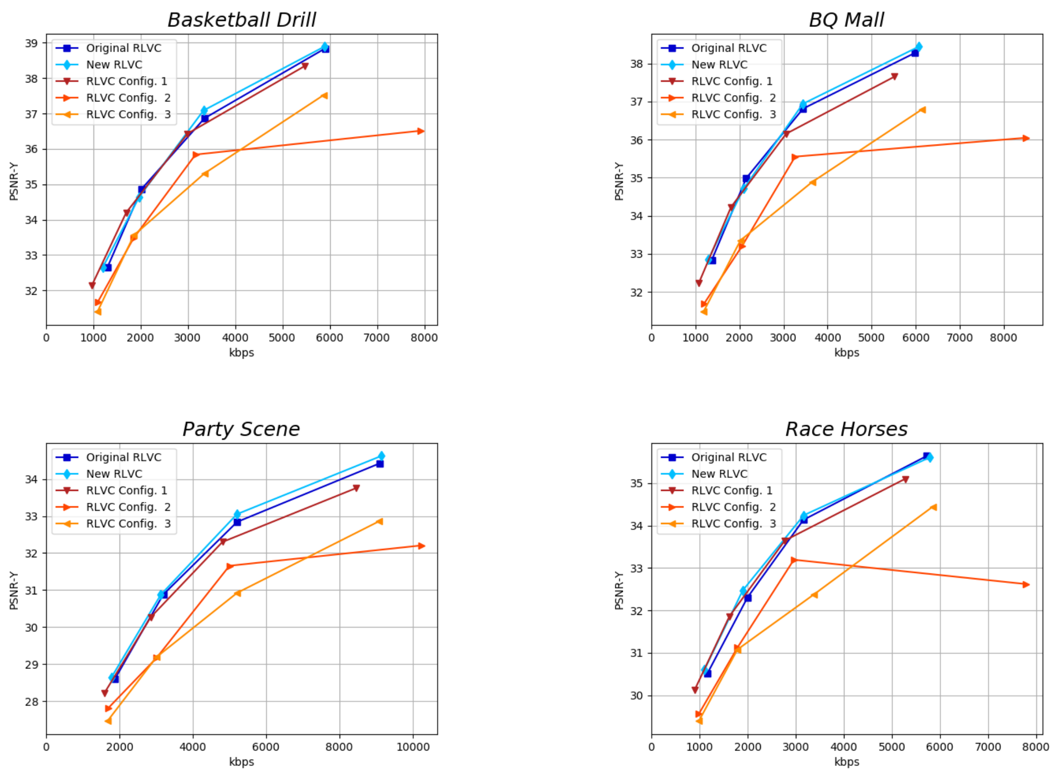

Figure 42: PSNR-Y RD performance for the Class C video sequences.

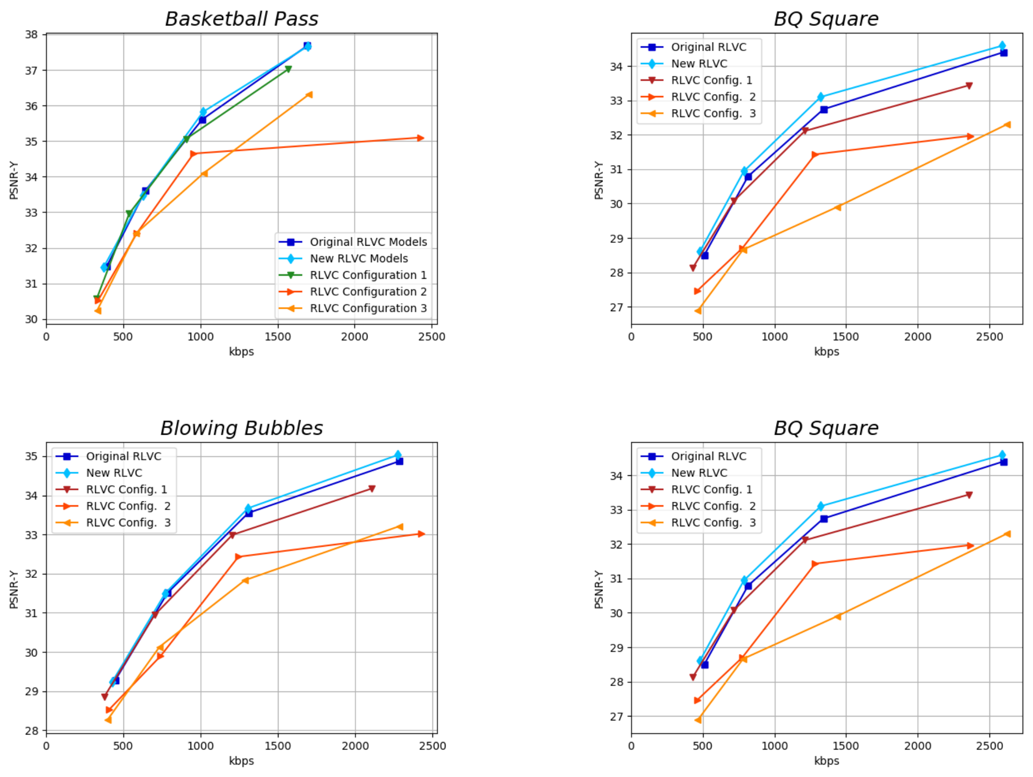

Figure 43: PSNR-Y RD performance for the Class D video sequences.

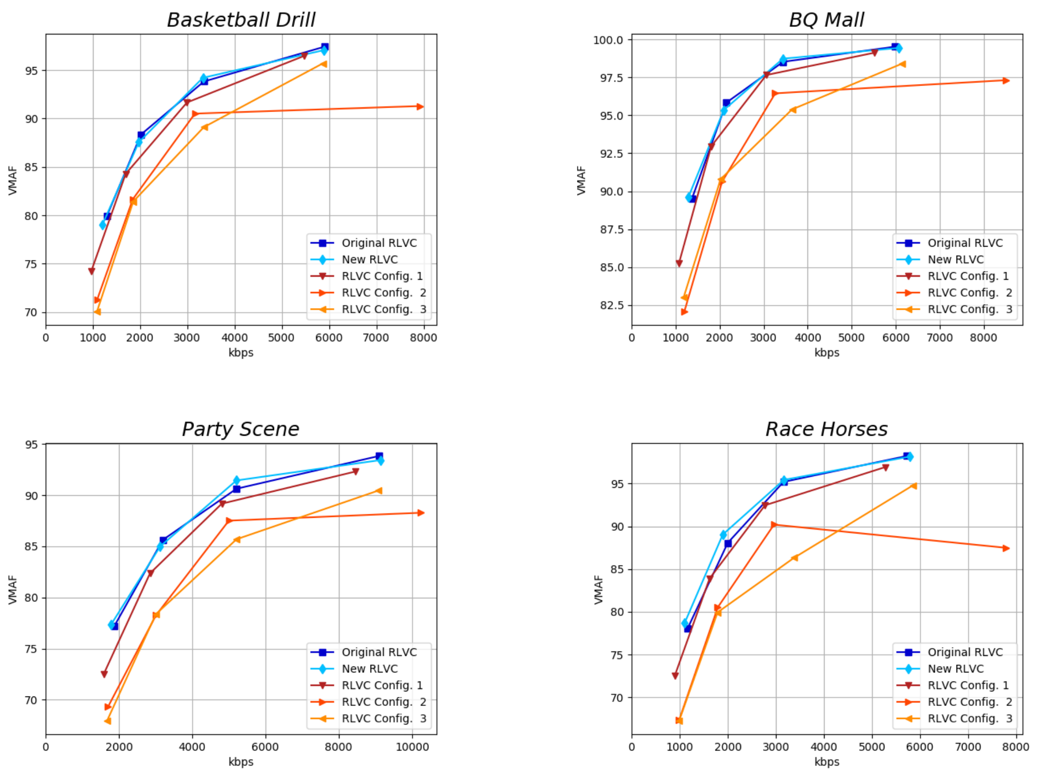

Figure 44: VMAF RD performance for the Class C video sequences.

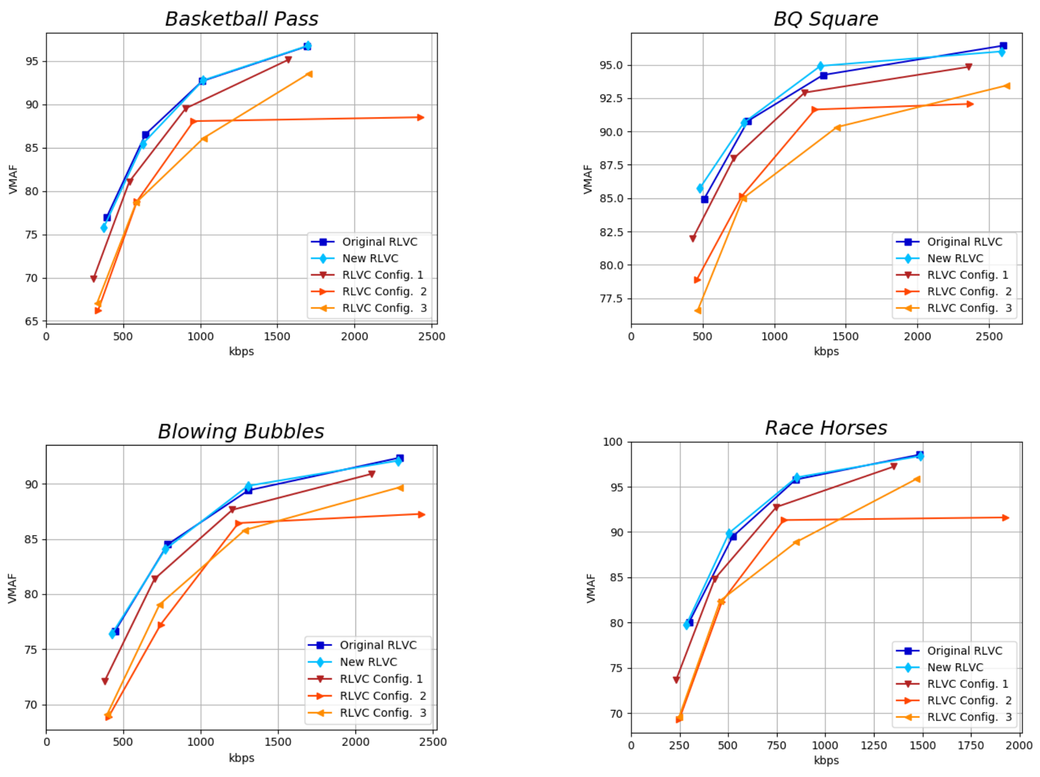

Figure 45: VMAF RD performance for the Class D video sequences.

Table 7 and Table 8 show the PSNR-Y and VMAF BD-Rate, respectively, for the RLVC training configurations defined above, using as reference the RD performance obtained with the pre-trained Original RLVC Models.

Table 7: PSNR-Y BD-Rate (%): Original RLVC models as reference.

Class	Sequence	Newly Trained RLVC	RLVC Configuration 1	RLVC Configuration 2	RLVC Configuration 3
Class C	*RaceHorses*	-5.74	-2.63	103.41	59.85
	BQMall	-1.26	3.65	14.12	63.27
	PartyScene	-4.31	5.13	23.24	56.23
	BasketballDrill	-2.43	-1.36	17.72	39.28
	Average	-3.44	1.20	39.62	54.66
Class D	*Basketball Pass*	-1.90	0.07	5.65	34.89
	BQSquare	-8,71	6.47	26.96	117.33
	Blowing Bubbles	-3,11	5.30	24.12	45.72
	RaceHorses	-3,87	1.57	4.41	43.12
	Average	-4,40	3.35	15.29	60.27

Table 8: VMAF BD-Rate (%): Original RLVC models as reference.

Class	Sequence	Newly Trained RLVC	RLVC Configuration 1	RLVC Configuration 2	RLVC Configuration 3
Class C	*RaceHorses*	-6,52	2.40	112.27	68,92
	BQMall	3,93	-7.83	-24.14	32,35
	PartyScene	0,13	11.57	-29.28	57,15
	BasketballDrill	1,67	8.93	-38.01	46,34
	Average	-0,20	3.77	5.21	51,19
Class D	*Basketball Pass*	2,12	13.48	-74,24	53,11
	BQSquare	-4,02	12.78	-40,89	73,37
	Blowing Bubbles	1,00	13.89	-21,19	42,56
	RaceHorses	-1,98	8.98	-87,45	51,87
	Average	-0,72	12.28	-55,94	55,23

From the results in figures 42, 43, 44, and 45, and tables7 and 8, it is possible to observe:

- **Newly Trained RLVC models versus Original RLVC models:** The Newly Trained RLVC models can achieve a similar, and even better, RD performance as the Original RLVC models provided by the authors. Even though the training followed the recommendations by the RLVC authors, there are several aspects that may account for the slight differences in performance obtained.

Since the neural networks weights are initialized randomly, the starting point for the training of the Newly Trained RLVC models may be different than the starting point of the pre-trained RLVC models; therefore, both models can be optimizing towards rather different minima, which may lead to rather different models and, therefore, slightly different, and in this case better, RD performance.

- **PSNR-Y versus VMAF**: For the VMAF metric, there is a clear decrease in quality when Phase 2 of training is skipped, which is not so evident with the PSNR-Y metric. For the VMAF metric, there is also a smaller gap in quality between the results obtained with the Newly Trained RLVC models and the RLVC Configuration 3 than there is for the PSNR-Y quality metric, which can be verified in the charts and associated BD-Rates. This seems to indicate that the 'mathematical quality' difference detected by PSNR-Y is not equally important according to the more perceptual VMAF metric.
- **Newly Trained RLVC models versus Configuration 1 models:** Comparing the Newly Trained RLVC models with Configuration 1 models, a slight RD performance variation may be noted. With the VMAF quality metric, all sequences show a faintly worse RD performance when Phase 2 is skipped. For the PSNR-Y metric, the results show that the Configuration 1 RD performance is similar to the Newly Trained RLVC models for the lower qualities, but worse for the higher qualities. For the lower qualities, the Configuration 1 results have worse quality but also a smaller bitrate compared with the Newly Trained RLVC models, which ultimately makes the curves align in this part of the chart and have a similar RD performance. However, for the highest qualities of the RLVC Configuration 1, the PSNR-Y starts to worsen and creating a gap between the two curves.
- **Configuration 1 models versus Configuration 2 and 3 models:** Both training Configurations 2 and 3 are proven to always be worse than the remaining configurations, which implies that the training should, at least, not be stopped too early in Phase 1. From Configuration 2 to Configuration 3, a slight decrease in RD performance can be noted, which is due to the fewer 100 000 iterations that Configuration 3 performs in Phase 1.
- **BD-Rate Computation Issues:** While most BD-Rate results in both Table 7 and Table 8 agree with the RD performance results in figures 42, 43, 44, and 45, the results in Table 8 for the RLVC Configuration 2 pertaining to Class C and D do not fit well the RD performance charts. These results show that, on average, for the VMAF quality metric with Class D sequences, the same video quality should lead to an approximate 55% decrease in bitrate; however, this is not observed in Figure 45. This mismatch is due to the irregularity manifested at the higher quality RD point for the "RLVC Configuration 2", which makes it hard to model with a third order polynomial (as done in the BD-Rate computation) and thus, leads to a BD-Rate value which does not express well the real RD performance situation.

In summary, it may be concluded that the author of this Thesis is able to appropriately train RLVC models, i.e., the 'Newly Trained RLVC Models', and thus to 'control' the RLVC framework in training and testing. This implies that a reliable starting point and benchmark for the next potential improvements to the RLVC architecture is available.

Moreover, the study of simpler training configurations for the RLVC model allows concluding that skipping and/or shortening the original training phases is not advisable, as the RLVC RD performance would be penalized.

6 RLVC with Attention Model: a Preliminary Study

After careful thought on the best way to extend the RLVC architecture to improve its RD performance two tools got priority: *attention models* and, more specifically, the so-called *Transformer*.

Attention models were firstly devised and used to improve RNN-based architectures [5], which deal with temporal data, specifically containing gated units similar to LSTM cells; thus it is known that attention models and RNNs work well together, and the RLVC codec, adopted for extension, is an RNN-based architecture. Attention models have also been incorporated into architectures with RNNs to solve computer vision tasks and have shown improvements [59], thus hinting that attention models may also be of use for tasks involving data such as images and video.

The Transformer model [6] emerged after the attention models were implemented into several NLP solutions. The Transformer is a DL model entirely based on attention mechanisms, which proved that, for some specific NLP tasks, namely machine translation, a model can achieve high performance without using RNNs or even CNNs, and in alternative employing only attention mechanisms. The Transformer shows that attention mechanisms alone can learn to faithfully represent temporal inputs and its internal connections, without losing these benefits for larger inputs, which commonly happens for RNN-based architectures when dealing with sentences longer than the ones used in training. The Transformer offers improved results while needing less training time and using a less computationally complex architecture than the RNN-based architectures usually used for NLP tasks. Most recently, the Transformer model has been applied to computer vision problems, like image recognition [7] with great success, and even to video classification tasks [9]. The Transformer (eventually with some adaptations) may be used in a video codec architecture to replace the often used RNNs, exploiting temporal correlation more efficiently, and thus may lead to coding efficiency improvements like those achieved in NLP tasks.

As a steppingstone into using the Transformer in a video coding architecture, it has been decided to integrate an attention model suitable for the processing of images in the RLVC codec, since it uses an RNN-based model, where it is known the attention models may provide some benefits. Since several different types of attention models have been proposed in the literature, the attention model chosen to extend the RLVC architecture will be motivated and described in this chapter.

In this context, this chapter will firstly describe the attention model used to extend the RLVC architecture [49], with the goal of improving its RD performance. Then, the attention model integration in the RLVC architecture and the associated training process will be presented, and finally the RD performance of the extended RLVC (X-RLVC) codec will be studied and compared to the Newly Trained RLVC model RD performance achieved in Chapter 5. As in Chapter 5, the X-RLVC architecture will only be trained for the four different qualities with the PSNR-RGB distortion metric, due to training complexity and time constraints.

Because DL-based video coding research is very computationally intensive, all experiments, especially the training, take a significant amount of time. This has prevented to go in this Thesis beyond a preliminary study of RLVC video coding with attention models. Still the author of this Thesis believes this will be a major research topic in coming years.

6.1 Selected Attention Model

The attention model selected was the so-called Convolutional Block Attention Module (CBAM) [60]. According to its proponents, this attention model is simple but effective and was designed to be easily integrated into any CNN architecture. The CBAM was created to improve architectures that perform image classification, and has shown competitive results [60]. These capabilities may be also important in aiding the RLVC framework to learn which inputs it should dedicate more attention in order to achieve a better RD performance.

In practice, an attention model is applied to an input to identify which of its parts should be given more emphasis, i.e., attention, which translates into multiplying those inputs by higher weights, and the less important inputs by lower weights, so their values can stand out if they should be given more attention, or be drowned out if they are less significant.

Figure 46 shows the architecture of the selected CBAM model. This attention model considers two different types of attention: *channel* wise attention and *spatial* wise attention. The CBAM takes the input tensor, which is three-dimensional (an image in RBG channels, or a feature map of an image already passed through some convolutional layers, which will add depth), and processes the set of input features along the various tensor channels in the channel attention module; after these features are processed along the tensor channel axis in the spatial attention module. Both attention modules are characterized by a set of weights that are applied to the input, by multiplication, thus embedding into each part of the input its associated significance.

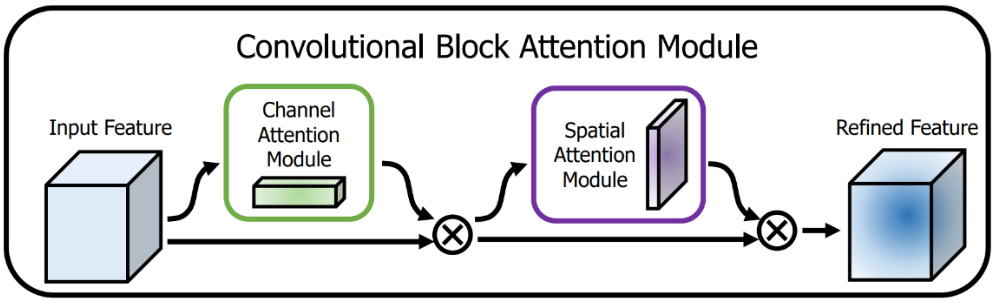

Figure 46: CBAM architecture [60].

<u>Channel Attention Module</u>

The channel attention module architecture is shown in Figure 47. Since each channel of a feature map usually detects a specific feature, the channel attention focuses on which channel will have the most important characteristics for a given group of features. Therefore, average pooling is applied to each channel individually to compute channel statistics, and max pooling is also performed to collect distinctive features over all channels; the results of both pooling operations generate the so-called MaxPool and AvgPool descriptors shown in Figure 47. Both descriptors are passed through the same

Multi-Layer Perceptron (MLP) separately, which is a simple FCNN architecture with at least one hidden layer, and both outputs are merged with an element-wise summation, finally creating the so-called channel attention map (M_c).

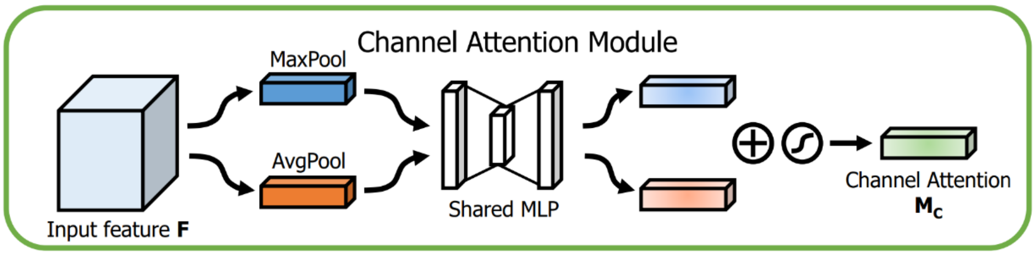

Figure 47: Channel Attention Module architecture [60].

Spatial Attention Module

The input of the spatial attention module is the output of the channel attention module multiplied by the initial feature tensor, as these operations are performed sequentially. In this module, the inter-spatial features are extracted by performing average and max pooling along the channel axis, so that two feature maps are obtained for the average and most distinctive characteristics, respectively; these feature maps are then concatenated. Then, a convolution operation is applied to the concatenated descriptors to create the final spatial attention map (M_S), which includes the weights expressing how much the spatial regions of the input should be amplified or suppressed. The CBAM input, after being channel-wise weighted by the channel attention map, is spatially weighted with the spatial attention map, finally creating the CBAM refined output.

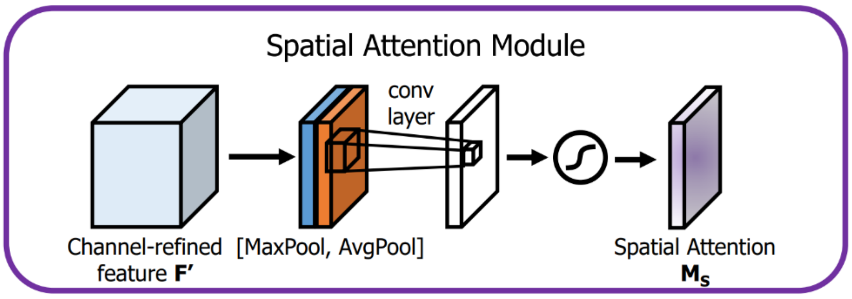

Figure 48: Spatial Attention Module architecture [60].

6.2 RLVC Extended Architecture

To enhance the RLVC architecture, it was decided to include the CBAM module in the motion and residue RAEs since they perform the most important operations in the codec, i.e. the encoding and decoding of the motion and residue information, thus defining the final quality, while taking into consideration information from past frames in the video sequence. Thus, by inserting the attention model in the two RAEs, the encoder-decoder architecture should receive additional assistance in identifying which information should be considered most important, either to encode, or to pass along to the next pair of processed frames, through the ConvLSTM layer states.

The attention model was only included at the encoder side of the RAEs, since this is the position where it should be more beneficial, allowing the attention model to control the information being encoded. However, it would be interesting to study a more symmetric approach where the attention model is also included at the decoder side.

To study the different impacts of the CBAM module in the RLVC codec, two different positions for the model, within the RAEs, were studied. Other positions could also have been studied if time had allowed.

Position 1 – CBAM Before Quantization

In this case, the CBAM module is inserted immediately before the quantization of the latent representations, i.e., after all the RAE encoding layers, see Figure 49. Thus, after reducing the data to its most compact form, the attention model should determine which information is most important, so that the weighted latent representation is processed by the quantization and the decoder.

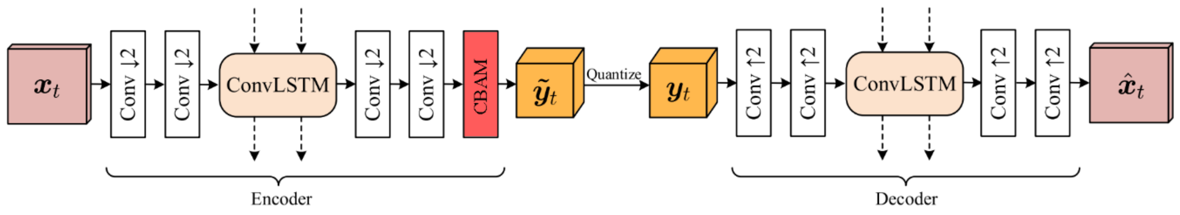

Figure 49: Modified RAE architecture, with CBAM module in Position 1 – before quantization.

Position 2 – CBAM Before ConvLSTM

In this case, the CBAM module is inserted before the ConvLSTM layer at the encoder side of the RAE, see Figure 50. This position should offer benefits to the recurrent part of the RAEs, since it will carefully weight the features information given to the ConvLSTM layer, which will then be used in the coding of the next frames.

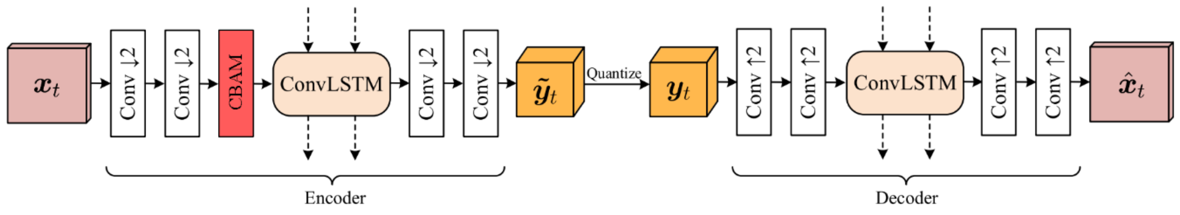

Figure 50: Modified RAE architecture, with CBAM module in Position 2 – before ConvLSTM.

The remaining RLVC architecture, notably the motion estimation and motion estimation networks, and both RPM networks, was not changed at all.

6.3 Training Process

A) Training Configurations

The training of the X-RLVC model will basically include the same phases described in Section 5.2 for the original RLVC model, with the eventual addition of some new phases associated to the CBAM training. Since the Newly Trained RLVC models from Chapter 5 are available, they will be used in some configurations as starting models for the additional training of the X-RLVC with CBAM; this means in practice that the X-RLVC models may refine and extend the previously available RLVC models to avoid starting the training from the beginning.

While there are many interesting training configurations, the initially chosen configurations to train both positions of the CBAM were:

- **Training Configuration 1 (TC1) – Square One with Phase 1:** In this configuration, the X-RLVC model is trained from the start, with no loaded models (except the motion estimation network) as performed in the original Phase 1 training. All network parameters, except the motion estimation network, are randomly initialized, and Phase 1 training progresses as described in Section 5.2.1. Then, when Phase 1 is finished, the obtained models are used to initiate Phase 2 training, as described in Section 5.2.2. The models obtained at the end of Phase 2 training are used to train both corresponding RPM networks, like described in Section 5.2.3. Thus, this training configuration would train the X-RLVC in exactly the same way the original RLVC is trained.
- **Training Configuration 2 (TC2) – Phase 1 Refinement:** In this configuration, the models obtained at the end of training Phase 1 for the networks of the Newly Trained RLVC model are used to initialize the X-RLVC model. This means that the only weights left to randomly initialize are the CBAM weights, and the remaining weights of the X-RLVC model are not randomly initialized. For this configuration, the training starts using the "Total Loss Function" (3) and the "Pairs Within the Sequence" batch configuration since Phase 1 of training had already been performed for the original models; thus, for model refinement with the CBAM, the loss function and batch configuration used are those used at the end of Phase 1. After Phase 1 is finished, the models obtained are used to start Phase 2 training, with all the same hyperparameters as described in Section 5.2.3. The models obtained at the end of this training process are used to train both corresponding RPM networks.
- **Training Configuration 3 (TC3) – Phase 2 Refinement:** In this configuration, the models obtained at the end of Phase 2 training for the Newly Trained RLVC model networks (motion estimation, RAEs, and motion compensation) are used as starting point for training the X-RLVC architecture. Similarly to the previous configuration, only the CBAM weights are randomly initialized. Then, the initialized X-RLVC architecture is trained again, although only with Phase 2 since this training configuration adopts a refinement approach and thus Phase 1 is not performed again. All hyperparameters remain the same as mentioned in Section 5.3.2, except for the learning rate, which starts with the last value used in the original Phase 2 ($1x10^{-6}$) and is then changed in the same way. The RAE models obtained at the end of this training process are then used to train both RPM networks, thus performing the same Phase 3 training as described in Section 5.2.3.

Unfortunately, given the complexity and time constraints, not all described training configurations have been trained and tested. As such, for Training Configuration 2, there is a distinction between results obtained with the models which followed the training configuration fully (that are referred to as "TC 2"), and the results obtained with models that were trained with an incomplete TC2, which performed only the original Phase 1 training and the Phase 1 refinement with CBAM (that are referred to as "TC 2, Only Phase 1 + CBAM"). For Training Configuration 3 the models were trained fully according to the description, for both positions of the CBAM.

A) A Training Problem: Exploding Gradients

During the training process with the X-RLVC model, i.e., with the CBAM, an additional problem emerged, that did not occur while training the Newly Trained RLVC model in Chapter 5.

In fact, during the training of some models, there were instances of exploding gradients. This happens in the backpropagation of gradients, when large derivatives make the gradients increase exponentially, and sometimes gradients even reach numerical overflow values. This causes the network weights and the loss function to diverge, instead of converging to the optimum point, and thus the model becomes unstable and the training does not improve in terms of loss function. The exploding gradient problem may be solved with the so-called gradient clipping control, which implies forcing the gradients to have a maximum value. A regular training of the X-RLVC was observed to learn the gradient norm and then gradient clipping was applied to trim any gradient too large to value closer to the gradient norm. After the implementation of gradient clipping, there were no more instances of the exploding gradient problem. Figure 51 shows the MSE distortion between the original motion flow, created by the motion estimation network, and the decoded motion flow, obtained after decoding by the motion RAE (the MSE values are saved during training, even though they are not directly used by the loss function); the chart in blue shows an example with exploding gradients, which led to an exponential growth of the MSE value, while the chart in green shows the MSE values during a new training, after gradient clipping was applied.

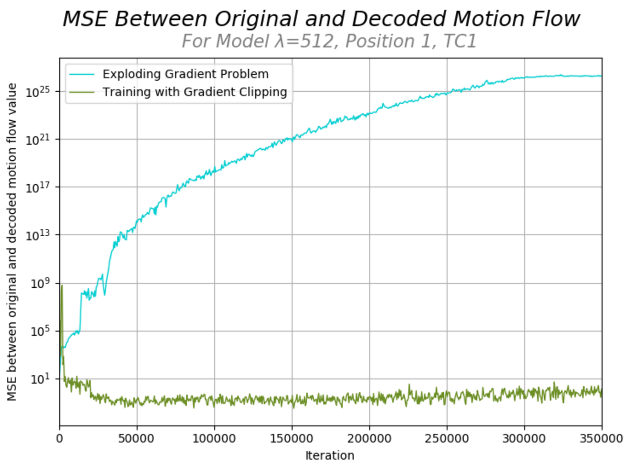

Figure 51: Example MSE values between the original and decoded motion flow, before and after gradient clipping.

6.4 Performance Results and Analysis

After correcting the exploding gradient training problem, the previously described architecture of the X-RLVC was tested, for both Position 1 and Position 2, and for training configurations TC2 and TC3.

6.4.1 Preliminary Results

Figures 52, 53, 54, and 55 show the obtained preliminary RD performance results for the JVET Classes C and D test videos, using again the PSNR-Y and VMAF quality metrics as in previous chapters.

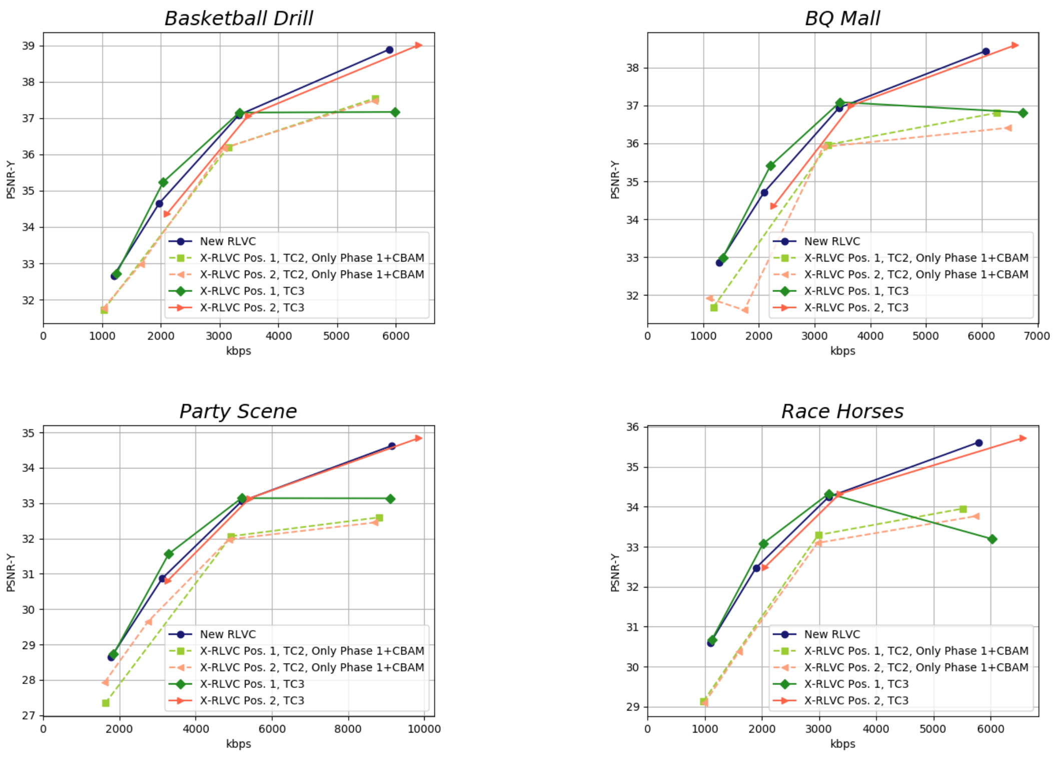

Figure 52: PSNR-Y RD performance for the Newly Trained RLVC models and X-RLVC models: Class C video sequences.

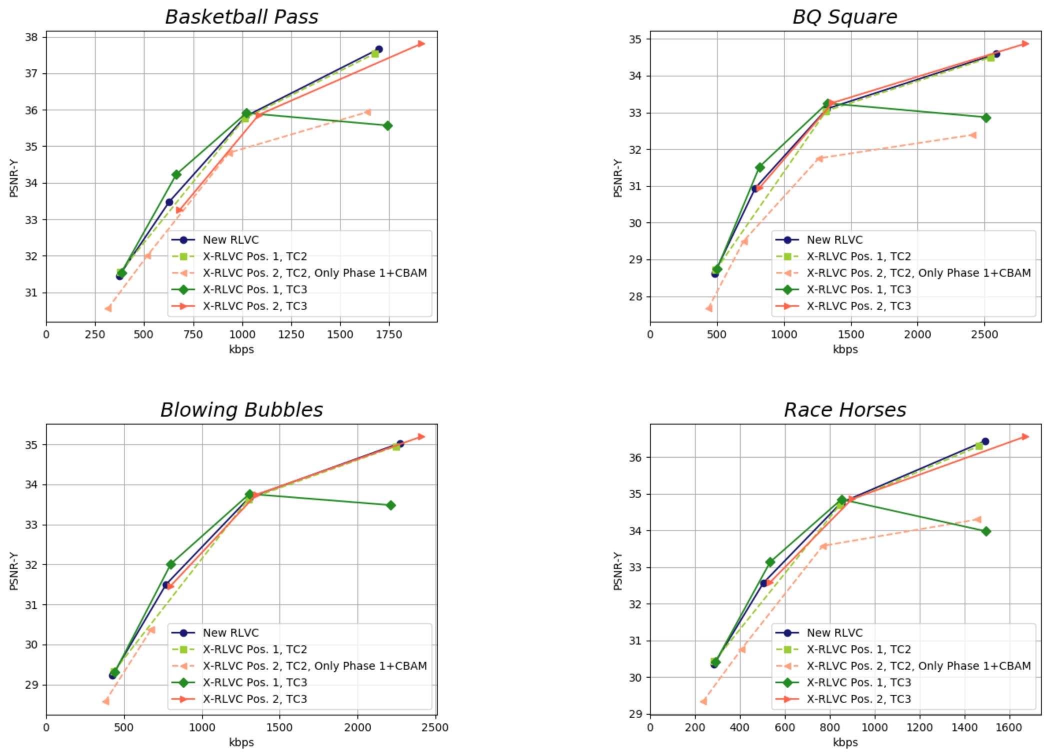

Figure 53: PSNR-Y RD performance for the Newly Trained RLVC models and X-RLVC models: Class D video sequences.

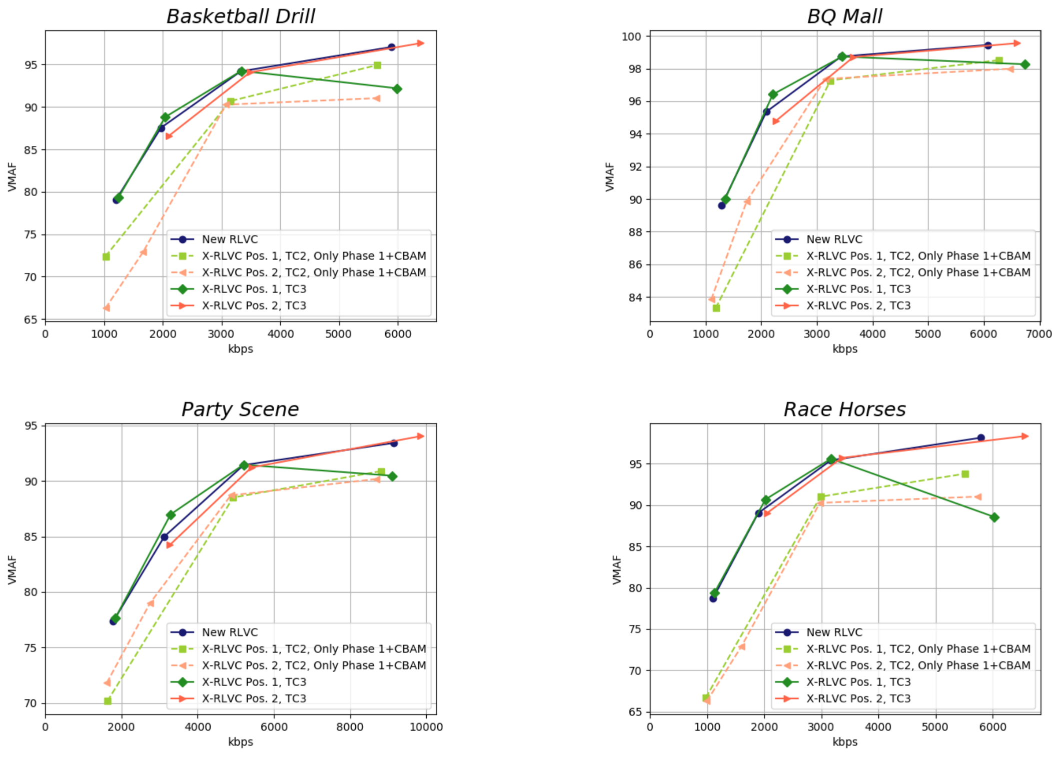

Figure 54: VMAF RD performance for the Newly Trained RLVC models and X-RLVC models: Class C video sequences.

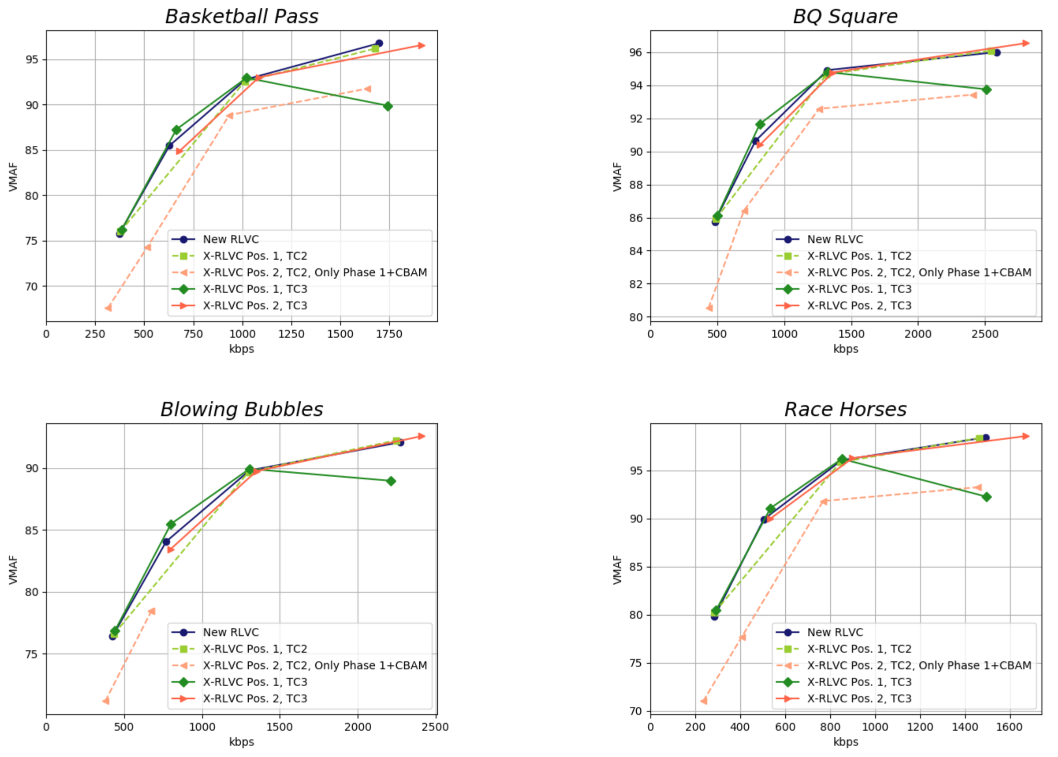

Figure 55: VMAF RD performance for the Newly Trained RLVC models and X-RLVC models: Class D video sequences.

6.4.2 The Highest Quality/Rate Point Problem

From the preliminary results shown in the previous figures, it is possible to observe that there is a clear and unexpected quality deterioration problem for the highest quality/rate model (λ=2048), when the CBAM is in Position 1 of the X-RLVC; the quality achieved is sometimes even worse than that obtained with the model for λ=512. The examination of the loss function during training for this model shows an unexpected progress since its evolution is quite erratic and unstable, even more so than the training example shown in Section 5.3.2, as shown in Figure 56.

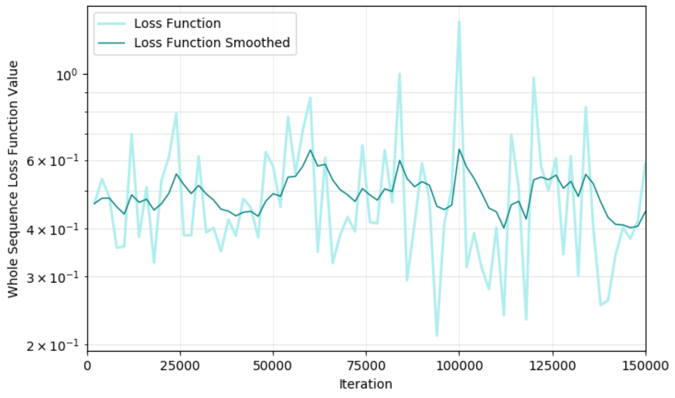

Figure 56: Evolution of the “Whole Sequence Loss Function” while TC3 training the X-RLVC Position 1 solution, for λ=2048.

A) Proposed Solutions for the Problem

To avoid this quality deterioration problem, several options were considered, and some implemented. To evaluate if the following proposed and trained models would yield favorable performance results during the testing stage, a prediction of the model’s efficiency was made based on the progress and final value of the loss function. For a model to achieve a good performance during the testing stage, it is expected that its loss function (for this quality/rate point) should end the training period with a final loss value similar to, or smaller than, 0.4, which is the value achieved with the Newly Trained RLVC model for λ=2048; as such, only the models that satisfied this condition were chosen to move on to the testing stage. The proposed solutions are the following:

- **Smaller learning rate** – The learning rate used for the training of the highest quality model was reduced by 10, compared to the learning rate used for the remaining qualities. The learning rate was decreased with the objective of making the network training take smaller steps, and thus, contradict the erratic evolution previously shown in the loss function. The decreasing of the learning rate resulted in a model that initially seemed to be progressing well, but after some iterations proved to be unable to converge to an optimum point, as shown in Figure 57. A more modest decrease in learning rate originated a slightly less erratic training, but still did not solve the quality deterioration problem. None of the models obtained after this change in training were tested due to the less than optimal evolution of the loss function during training, which does not satisfy the 0.4 loss function threshold previously set.

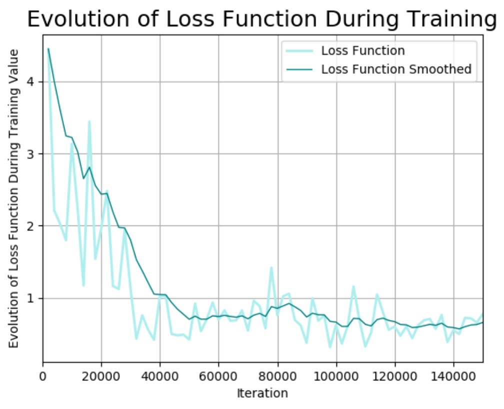

Figure 57: Loss function evolution during training for the “Smaller learning rate” proposed solution.

- **Initialization from λ=1024 model, with CBAM** – Instead of initiating the training from the Newly Trained RLVC model for λ=2048, training can be initiated from the model obtained after training the X-RLVC for λ=1024, i.e., the previous RD point, where the quality deterioration problem does not exist. This guarantees that the X- RLVC training is starting from a full X- RLVC model, closer to the target model, since in this case the CBAM weights are already initiated and not randomly initialized, and the whole architecture would only need a smaller progress to adjust to the new value of λ in the loss function. The evolution of the loss function during the training for this solution did not show promising results, see Figure 58; as a consequence, the model obtained after training was not tested with the test sequences.

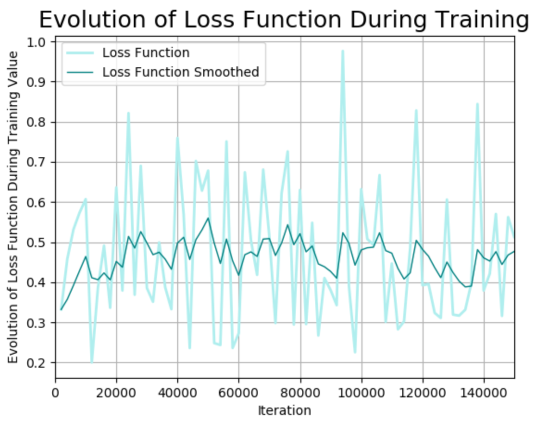

Figure 58: Loss function evolution during training for the “Initialization from λ=1024 model, with CBAM” proposed solution.

- **Freeze motion RAE model, except for the CBAM part** – Given that the Newly Trained RLVC model obtained in Chapter 5 achieves the expected RD performance, and that the CBAM may cause instability in training, another option for further development could be to “freeze” the motion RAE’s model weights already available (except for the CBAM inside the RAE), while the remaining X-RLVC architecture is fine-tuned. This could provide the model with a solid motion RAE model that is known to work, and a CBAM and remaining models that have to adapt to a functioning motion RAE model. This solution was trained, but the loss function, shown in Figure 59, exhibits similar characteristics to the loss function obtained for the “Smaller learning rate” solution; therefore, this solution was also not tested.

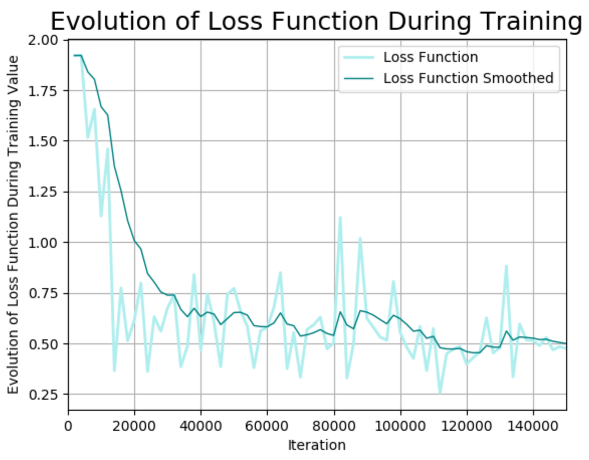

Figure 59: Loss function evolution during training for “Freeze motion RAE model, except for the CBAM part” proposed solution.

- **Remove the CBAM in the motion RAE** – The presence of the fully connected layers in the CBAM architecture may cause training to be more unstable. Thus, another solution to explore is the inclusion of the CBAM only in the residue RAE model and not in the motion RAE model, where it has the potential to cause the most instability due to the coding impact of the motion vectors. During the training of this solution, the progression of the loss function is promising, achieving a final value close to 0.4, as shown in Figure 60; therefore the model obtained at the end of training was effectively testes to check the final RD performance.

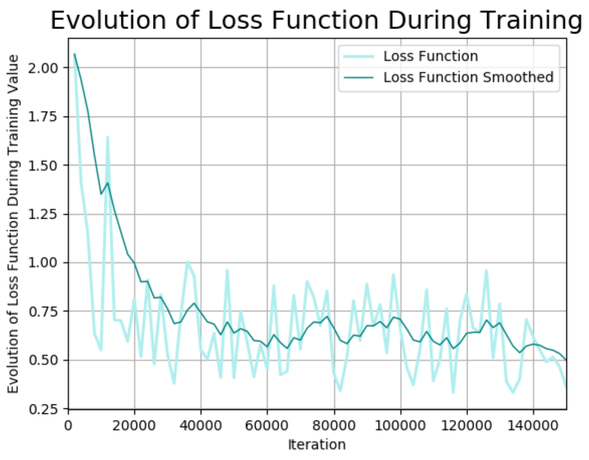

Figure 60: Loss function evolution during training for the “Remove the CBAM in the motion RAE” proposed solution.

- **Residual Connection** – Similarly to what is performed in [61], a residual connection is applied around the CBAM, see Figure 61. This type of residual connections were first applied to deep neural networks with optimization problems due to the backpropagation of gradients. This residual connection does not add extra parameters or computational complexity, and has shown much improvement over networks that do not use residual connections [61]. The residual connection should allow the gradients to flow backwards through the network while skipping the CBAM layers, and thus enable deeper networks to be trained without the risk of vanishing or exploding gradients.

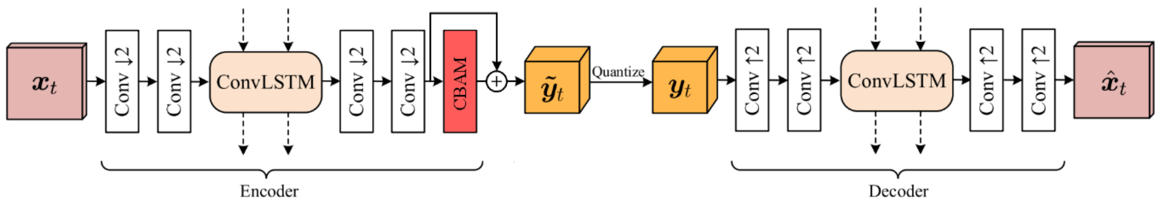

Figure 61: Modified RAE architecture, with residual connection around CBAM in Position 1.

The evolution of the loss function during the training for this further altered X-RLVC architecture is shown in Figure 62. From the loss function evolution, it is possible to observe that this solution shows the most promising results from all the possible described alternatives since the final loss function value achieved by the loss function is the smallest compared to the remaining proposed solutions; thus this solution was tested and the models for λ = 256, 512, 1024 were also trained and tested, in order to obtain a complete RD performance curve for this "Residual Connection" X-RLVC model.

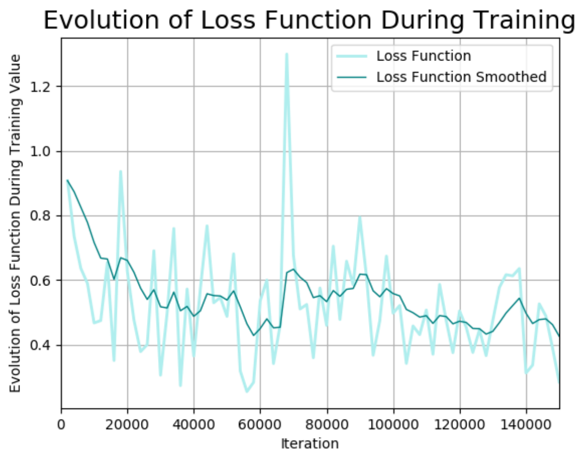

Figure 62: Loss function evolution during training for the "Residual Connection" proposed solution.

Despite the strategies listed above, of which the first was implemented and the second is in training, the training of the X-RLVC with the CBAM in Position 1 and with the hyperparameter λ=2048 was not successful, and the quality degradation can still be observed.

B) New Performance Results for Proposed Solutions

As mentioned previously, only two of the proposed solutions to solve the quality drop at the highest rate point were tested with the Class D test sequences, given the time constraints imposed. Figure 63 and Figure 64 show the results obtained for the last two mentioned solutions, the "Residual Connection" model in green and the "Remove the CBAM in the motion RAE model" in purple, as well as the previously obtained results for the X-RLVC model, all trained with the TC3 training configuration, for the CBAM in Position 1.

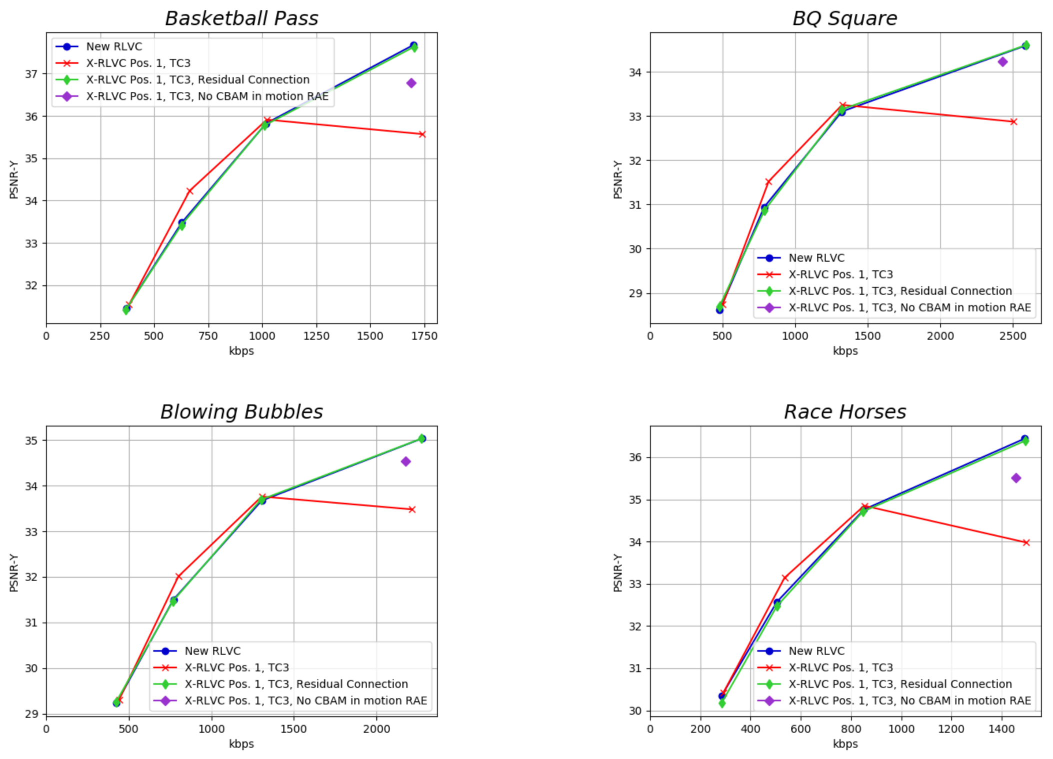

Figure 63: PSNR-Y RD performance for the Newly Trained RLVC models, X-RLVC models and the models for the proposed solutions: Class D video sequences.

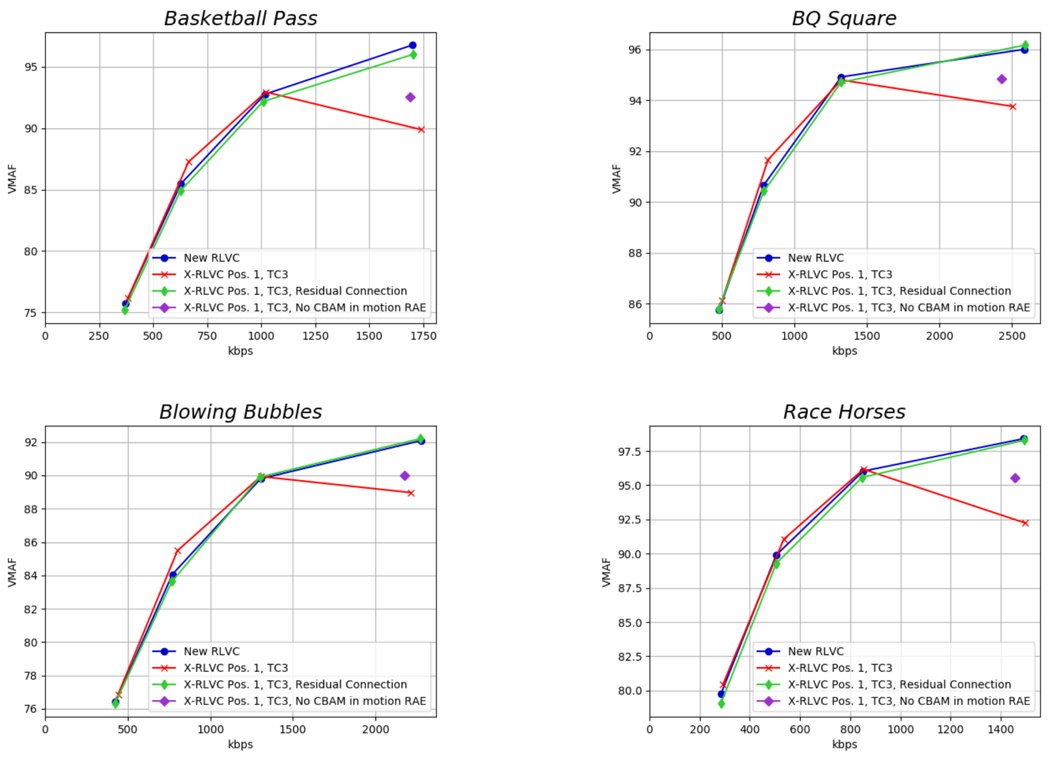

Figure 64: VMAF RD performance for the Newly Trained RLVC models, X-RLVC models and the models of the proposed solutions: Class D video sequences.

6.4.3 Overall RD Performance Analysis

Examining the preliminary RD performance curves presented in figures 52, 53, 54 and 55, the following conclusions may be extracted:

- **RLVC versus X-RLVC –** Generally, no version of the X-RLVC models with any training configuration provides better results over the Newly Trained RLVC models in a consistent way. The X-RLVC Position 1 shows improvement for some rate points (namely for λ=512, and λ=1024 in some cases) but shows worse performance for the highest quality model. The X-RLVC Position 2 has a consistently worse RD performance than the Newly Trained RLVC models.
- **X-RLVC Position 1 versus X-RLVC Position 2 –** The X-RLVC Position 1 shows more promising RD performance improvements, but the X-RLVC Position 2 is more consistent. The X-RLVC Position 1, for the λ values 512 and 1024, shows consistent, even if minor, improvements over the Newly Trained RLVC models. Also for this case, the X-RLVC model with λ=256 proves to have a similar performance as the Newly Trained RLVC model for the same quality. However, the gains for these points do not make up for the poor performance achieved with the λ=2048 model (highest rate). For X-RLVC Position 2, the lowest quality model (λ=256) is not available, since during training it suffered from the exploding gradient problem, and there were no computing resources to initiate training again, after the implementation of gradient clipping. The remaining quality models achieved for the X-RLVC Position 2 do not show overall improvement over the Newly Trained RLVC models, but for some sequences demonstrate a similar RD performance for the higher quality models. Overall, the X-RLVC Position 1 would be more promising if the quality degradation problem for the highest rate could be solved.
- **X-RLVC TC2 versus TC3 –** Given the time constraints, the difference between the training configurations (TC2 and TC3) can only be evaluated for the CBAM in Position 1, since the CBAM in Position 2 with TC2 was not fully trained until the end; as a consequence, the results shown in the mentioned figures skip the Phase 2 training with CBAM. From the results obtained for both training configurations with the CBAM in Position 1, it may be concluded that both these training configurations do not provide much different results. In fact, when comparing the results for the same qualities obtained with the different training processes, the results are approximately the same.
- **X-RLVC Position 1 Top Quality Problems –** It was not possible to obtain 'good' results for the X-RLVC model with the hyperparameter λ=2048, when the CBAM is in Position 1 (see Figure 49), even after trying the strategies mentioned earlier. The large loss function values make the training of this model more unstable and, coupled with an attention model (which can sometimes also have unstable training and different end results depending on the initialization [62]), the combination is prone to erratic and unpredictable training, which can lead to an unexpected poorer performance, as observed for the X-RLVC Position 1, trained in Configuration 1.

For the RD performance curves presented in figures 63 and 64, which regard changes made in the X-RLVC architecture to solve the highest quality/rate point problem:

- **Residual Connection X-RLVC vs Removing the CBAM in the motion RAE model** – Both these solutions provide an improvement in RD performance for the highest quality, with the Residual Connection model showing the best results. The model in which the CBAM was removed from the motion RAE model shows a more significant improvement for PSNR-Y than for VMAF, while the Residual Connection model shows the same performance for both quality metrics.
- **Residual Connection X-RLVC vs X-RLVC** – While the Residual Connection X-RLVC model shows a quality improvement for the critical RD point, this increase in RD performance is not verified for the models corresponding to the other RD points. In fact, the better RD performance for the highest quality points seems to have been achieved at the cost of losing the improved RD performance previously obtained for the lower RD points. Overall, both the Residual Connection X-RLVC and X-RLVC have the same RD performance.

From the obtained preliminary RD performance results with the X-RLVC models, it can be concluded that the insertion of the selected attention model into the RLVC architecture in the previously described positions and its subsequent training does not overall improve yet the RLVC performance. While some improvements happen for specific rate points, the gains are not consistent to claim any overall performance improvement.

With more available time and computational resources, different attention models, extended architectures, and additional training configurations could be tested, to eventually offer more promising RD performance results.

7 Conclusions and Future Work

This chapter will summarize the main contributions and conclusions of this Thesis, as well as suggest possible paths to be followed in related future work.

7.1 Main Contributions and Conclusions

The most relevant contributions of this Thesis are related to reviewing, benchmarking, and extending DL-based video coding solutions, in general, and the RLVC solution, in particular.

The first chapter of this Thesis introduces the context of this work: DL-based video coding; after, the motivation and objectives behind the use of attention models into an already existing DL-based video codec are presented.

Chapter 2 provides insight into the current, traditional video coding standards, while Chapter 3 presents a comprehensive introduction to DL concepts and tools, along with reviews of some of the most relevant DL-based video coding solutions in the literature.

Next, Chapter 4 provides an extensive and detailed benchmarking of the most recent DL-based video coding solutions with publicly available software against the most recent video coding standard, the VVC [52], used in several meaningful configurations. This work has led to a conference paper entitled "Conventional versus Learning-based Video Coding Benchmarking: Where Are We?", which has already been accepted and chosen as a best paper award candidate at the International Workshop on Advanced Image Technology 2022 (IWAIT'2022), which will be held in Hong Kong, in January 2022.

In Chapter 5, the most promising DL-based video coding solution from those previously reviewed and benchmarked in Chapter 4, the RLVC [49], was selected and its training process was implemented as suggested by the authors to create new DL models for the codec, with the goal of replicating the RLVC original model's performance in order to guarantee that extensions of the RLVC codec may be developed in a reliable way.

Finally, Chapter 6 targets extending the RLVC model with a selected attention model, the CBAM [60], to benefit from the advantages of this type of technology. First, the extended codec architecture is presented and after the training methods and the various obstacles that occurred during the training of the extended codec are highlighted. Finally, this chapter offers some preliminary performance results for the extended RLVC solution, notably for several CBAM positions within the RLVC architecture, and for different training strategies. These preliminary experiments did not yet allow obtaining clear RD performance gains with the extended RLVC since many ideas and associated experiments could not be tested due to the time and computational constraints; this was expected considering the ambition and complexity of the objective and the time frame for a MSc Thesis. However, the author still believes this attention model biased coding approach will have success in the future and will be a hot topic of work in the associated research community in the near future since it has great potential to provide performance improvements.

In summary, the main objectives of this Thesis were to perform the benchmarking of the current DL-based video coding models against conventional coding technology, achieved in Chapter 4, and to design, implement and train a DL-based video codec extended with an attention model, based on an already existing architecture from the literature, which would hopefully provide some RD performance improvement. While this second objective was not fully achieved, significant steps have been made in that direction.

7.2 Future Work

Considering the research path followed in this Thesis, interesting future work may take place in the following directions:

- Integration of attention models in DL-based video coding architectures:
  - Integration of different types of attention models into an RNN-based video coding architecture, or even other nor RNN-based architectures;
  - Study of additional positions for the attention model in both sides of the codec architecture (encoder and decoder), that might possibly give more leverage to the video coding process;
  - Definition of different training processes, that may better exploit the attention model and provide better performance results for the final attention-extended model.
- Other research directions for future DL-based video codecs:
  - Integration of the Transformer [6] architecture in already established DL-based video coding architectures;
  - Development of a new DL-based video codec that better fits the Transformer architecture, eventually with appropriate modifications, since this tool has proven to achieve great results even when used for tasks it was not initially designed for;
  - Development of a DL-based video codec based on a GAN architecture since GANs have been implemented and shown very competitive performance results for image coding [20] [63] [64].

In summary, the integration of attention models is still a new endeavor in the DL-based video coding arena, but it may come to provide fruitful performance results, as it has already happened in other related areas, such as image coding [65] [66] [67].

References

[1] Cisco, “Cisco Annual Internet Report (2018–2023) White Paper,” 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed: 30-Dec-2020].

[2] J. Hodgson, “Alternative Social Media Platforms Your Business Should Consider In 2016,” 2016. [Online]. Available: https://allthingsweb.co.uk/blog/alternative-social-media-platforms-your-business-should-consider-in-2016/#. [Accessed: 12-Oct-2021].

[3] P. Pachasirisakun, “Streaming Movie Illustration Vector,” 2020. [Online]. Available: https://www.istockphoto.com/pt/vetorial/streaming-movie-illustration-vector-gm1225052430-360448506. [Accessed: 12-Oct-2021].

[4] DeepAI, “Attention Models.” [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/attention-models. [Accessed: 19-Oct-2021].

[5] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” in *3rd International Conference on Learning Representations*, San Diego, CA, USA, 2015.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in *Advances in Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 5998–6008.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale,” in *International Conference on Learning Representations*, Virtual, 2021.

[8] S. Atito, M. Awais, and J. Kittler, “SiT: Self-supervised vIsion Transformer,” *ArXiv Prepr. arXiv2104.03602*, 2021.

[9] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, “ViViT: A Video Vision Transformer,” *ArXiv Prepr. arXiv2103.15691*, 2021.

[10] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 557–559, 2003.

[11] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[12] J. R. Ohm and G. J. Sullivan, “Versatile Video Coding - Towards the Next Generation of Video Compression,” in *Picture Coding Symposium*, San Francisco, CA, USA, 2018.

[13] Z. Li, C. Bampis, J. Novak, A. Aaron, K. Swanson, A. Moorthy, and J. Cock, “VMAF: The Journey Continues,” *Netflix Technol. Blog*, vol. 25, 2018.

[14] S. Rowe, “Introduction to Neurons in Neural Networks.” [Online]. Available: https://medium.com/artificial-neural-networks/introduction-to-neurons-in-neural-networks-71828d040a65. [Accessed: 28-Dec-2020].

[15] H. Mohamed, A. Negm, M. Zahran, and O. C. Saavedra, “Assessment of Artificial Neural Network for Bathymetry Estimation Using High Resolution Satellite Imagery in Shallow Lakes: Case Study el Burullus lake.,” in *International Water Technology Conference*, Sharm El Sheikh, Egypt, 2015, pp. 434–444.

[16] I. Tabian, H. Fu, and Z. S. Khodaei, “A Convolutional Neural Network for Impact Detection and Characterization of Complex Composite Structures,” *Sensors*, vol. 19, no. 22, p. 4933, 2019.

[17] Peltarion, “2D Convolution Block.” [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block. [Accessed: 02-Jan-2020].

[18] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks.” [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 28-Dec-2020].

[19] A. Dertat, “Applied Deep Learning - Part 3: Autoencoders.” [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798. [Accessed: 28-Dec-2020].

[20] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. Van Gool, “Generative Adversarial Networks for Extreme Learned Image Compression,” in *IEEE International Conference on Computer Vision*, Seoul, South Korea, 2019, pp. 221–231.

[21] J. Brownlee, “A Gentle Introduction to Generative Adversarial Networks (GANs).” [Online]. Available: https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/. [Accessed: 28-Dec-2020].

[22] “RNN or Recurrent Neural Network.” [Online]. Available: https://hackernoon.com/rnn-or-recurrent-neural-network-for-noobs-a9afbb00e860. [Accessed: 28-Dec-2020].

[23] C. Olah, “Understanding LSTM Networks.” [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 28-Dec-2020].

[24] S. Xingjian, Z. Chen, H. Wang, D. Y. Yeung, W. K. Wong, and W. C. Woo, “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting,” in *Adv. Neural Inf. Process. Syst.*, Montréal, Canada, 2015, pp. 802–810.

[25] A. Xavier, “An Introduction to ConvLSTM,” 2019. [Online]. Available: https://medium.com/neuronio/an-introduction-to-convlstm-55c9025563a7. [Accessed: 04-Oct-2021].

[26] G. Lu, W. Ouyang, D. Xu, X. Zhang, C. Cai, and Z. Gao, “DVC: An End-to-End Deep Video Compression Framework,” in *IEEE Conference on Computer Vision and Pattern Recognition*,

Long Beach, CA, USA, 2019, pp. 11006–11015.

[27] A. Ranjan and M. J. Black, “Optical Flow Estimation Using a Spatial Pyramid Network,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, Hawaii, USA, 2017, pp. 4161–4170.

[28] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, “Variational Image Compression With a Scale Hyperprior,” in *6th International Conference on Learning Representations*, Vancouver, BC, Canada, 2018.

[29] J. Ballé, V. Laparra, and E. P. Simoncelli, “End-to-end Optimized Image Compression,” in *5th International Conference on Learning Representations*, Toulon, France, 2017.

[30] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, “Video Enhancement With Task-oriented Flow,” *Int. J. Comput. Vis.*, vol. 127, no. 8, pp. 1106–1125, 2019.

[31] A. Mercat, M. Viitanen, and J. Vanne, “UVG Dataset: 50/120fps 4K Sequences for Video Codec Analysis and Development,” in *Proceedings 11th ACM Multimedia Syst. Conf*, Istanbul Turkey, 2020, pp. 297–302.

[32] C. Y. Wu, N. Singhal, and P. Krähenbühl, “Video Compression Through Image Interpolation,” in *Proceedings of the European Conference on Computer Vision*, Munich, Germany, 2018, pp. 416–431.

[33] A. Habibian, T. Van Rozendaal, J. M. Tomczak, and T. S. Cohen, “Video Compression With Rate-Distortion Autoencoders,” in *Proceedings of the IEEE International Conference of Computer Vision*, Seoul, South Korea, 2019, pp. 7032–7041.

[34] A. Van Den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, “Conditional Image Generation With PixelCNN Decoders,” in *Advances in Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 4790–4798.

[35] K. He, G. Gkioxari, P. Dollar, and R. Girshick, “Mask R-CNN,” in *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy, 2017, pp. 2961–2969.

[36] J. Carreira and A. Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset,” in *Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 4724–4733.

[37] F. Ofli, R. Chaudhry, G. Kurillo, R. Vidal, and R. Bajcsy, “Berkeley MHAD: A Comprehensive Multimodal Human Action Database,” in *Proceedings of the IEEE Workshop on Applications of Computer Vision*, Claerwater Beach, FL, USA, 2013, pp. 53–60.

[38] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, “Learning Image and Video Compression Through Spatial-Temporal Energy Compaction,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Long Beach, USA, 2019, pp. 10071–10080.

[39] S. Niklaus, L. Mai, and F. Liu, “Video Frame Interpolation Via Adaptive Separable Convolution,” in *IEEE International Conference on Computer Vision*, Venice, Italy, 2017, pp. 261–270.

[40] J. Katto and Y. Yasuda, "Performance Evaluation of Subband Coding and Optimization of Its Filter Coefficients," *J. Vis. Commun. Image Represent.*, vol. 2, no. 4, pp. 303–313, 1991.

[41] S. Rowe, "Video Trace Library." [Online]. Available: http://trace.eas.asu.edu/index.html. [Accessed: 30-Dec-2020].

[42] R. Yang, F. Mentzer, L. Van Gool, and R. Timofte, "Learning for Video Compression With Hierarchical Quality and Recurrent Enhancement," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA, 2020, pp. 6627–6636.

[43] F. Bellard, "BPG Image Format." [Online]. Available: https://bellard.org/bpg/. [Accessed: 07-Dec-2020].

[44] J. Lee, S. Cho, and S. K. Beack, "Context-Adaptive Entropy Model for End-to-End Optmized Image Compression," in *Proceedings of the International Conference on Learning Representations*, New Orleans, LA, USA, 2019.

[45] R. Yang, X. Sun, M. Xu, and W. Zeng, "Quality-Gated Convolutional LSTM for Enhancing Compressed Video," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, Shanghai, China, 2019, pp. 532–537.

[46] Xiph.org., "Xiph.org Video Test Media." [Online]. Available: https://media.xiph.org/video/derf/. [Accessed: 07-Dec-2020].

[47] VQEG., "VQEG Video Datasets and Organizations." [Online]. Available: https://www.its.bldrdoc.gov/vqeg/video-datasets-and-organizations.aspx. [Accessed: 07-Dec-2020].

[48] F. Bossen, "Common Test Conditions and Software Reference Configurations," in *9th Meeting of the JCT-VC*, Geneva, Switzerland, 2013, vol. 12.

[49] R. Yang, F. Menzter, L. Van Gool, and R. Timofte, "Learning for Video Compression With Recurrent Auto-Encoder and Recurrent Probability Model," *IEEE J. Sel. Top. Signal Process.*, vol. 15, no. 2, pp. 388–401, 2021.

[50] H. Liu, L. Huang, M. Lu, T. Chen, and Z. Ma, "Learned Video Compression Via Joint Spatial-Temporal Correlation Exploration," in *Proc. AAAI Conf. Artif. Intell.*, New York, NY, USA, 2020, pp. 11580–11587.

[51] E. Agustsson, D. Minnen, N. Johnston, J. Balle, S. J. Hwang, and G. Toderic, "Scale-Space Flow for End-to-End Optimized Video Compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Seattle, WA, USA, 2020, pp. 8503–8512.

[52] B. Bross, J. Chen, J. R. Ohm, G. J. Sullivan, and Y. K. Wang, "Developments in International Video Coding Standardization After AVC, With an Overview of Versatile Video Coding (VVC)," *Proc. IEEE*, vol. 109, no. 9, pp. 1463–1493, 2021.

[53] JVET, "VVCSoftware_VTM." [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/releases#VTM-11.0. [Accessed: 01-

Mar-2021].

[54] R. Yang, L. Van Gool, and R. Timofte, “OpenDVC - An Open Source Implementation of the DVC Video Compression Method,” *ArXiv Prepr. arXiv2006.15862*, 2020.

[55] R. Yang, L. Van Gool, and R. Timofte, “OpenDVC Software Implementation.” [Online]. Available: https://github.com/RenYang-home/OpenDVC. [Accessed: 01-Mar-2021].

[56] R. Yang, L. Van Gool, and R. Timofte, “HLVC Software Implementation.” [Online]. Available: https://github.com/RenYang-home/HLVC. [Accessed: 01-Mar-2021].

[57] R. Yang, F. Menzter, L. Van Gool, and R. Timofte, “RLVC Software Implementation.” [Online]. Available: https://github.com/RenYang-home/RLVC. [Accessed: 01-Mar-2021].

[58] JVET, “Common Test Conditions and Evaluation Procedures for Neural Network-based Video Coding Technology,” *Doc. JVET-U2016-r1*. 2021.

[59] F. Wang and D. M. J. Tax, “Survey on the Attention Based RNN Model and Its Applications in Computer Vision,” *ArXiv Prepr. arXiv1601.06823*, vol. arXiv:1601, 2016.

[60] S. Woo, J. Park, J. Y. Lee, and I. S. Kweon, “CBAM: Convolutional Block Attention Module,” in *European Conference on Computer Vision*, Munich, Germany, 2018, pp. 3–19.

[61] H. Kaiming, Z. Xiangyu, R. Shaoqing, and S. Jian, “Deep Residual Learning for Image Recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[62] A. Zeyer, A. Merboldt, R. Schlüter, and H. Ney, *A Comprehensive Analysis on Attention Models*. 2019.

[63] S. Kudo, S. Orihashi, R. Tanida, and A. Shimizu, “GAN-based Image Compression Using Mutual Information Maximizing Regularization,” in *Proceedings of the Picture Coding Symposium (PCS)*, Ningbo, China, 2019, pp. 1–5.

[64] L. Wu, K. Huang, and H. Shen, “A GAN-based Tunable Image Compression System,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Snowmass Village, CO, USA, 2020, pp. 2334–2342.

[65] H. Liu, T. Chen, P. Guo, Q. Shen, X. Cao, Y. Wang, and Z. Ma, “Non-local Attention Optimized Deep Image Compression,” *ArXiv*, vol. 1904.09757, 2019.

[66] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, “Learned Image Compression With Discretized Gaussian Mixture Likelihoods and Attention Modules,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Virtual, 2020, pp. 7939–7948.

[67] L. Zhou, S. Zhenhong, X. Wu, and J. Wu, “End-to-end Optimized Image Compression with Attention Mechanism,” in *CVPR Workshops*, Long Beach, CA, USA, 2019.